AFRL-RQ-WP-TR-2018-0196

# HyCIRCA: FORMAL SYNTHESIS AND VERIFICATION TECHNIQUES OF AUTONOMOUS CYBER-PHYSICAL SYSTEMS

**Robert P. Goldman, Daniel Bryce, David J. Musliner, and Michael J.S. Pelican**
**Smart Information Flow Technologies**

**Md. Ariful Islam, Frank Pfenning, and Edmund M Clarke**
**Carnegie-Mellon University**

**OCTOBER 2018**
**Final Report**

**THIS IS A SMALL BUSINESS TECHNOLOGY TRANSFER (STTR) PHASE II REPORT.**

**AIR FORCE RESEARCH LABORATORY**
**AEROSPACE SYSTEMS DIRECTORATE**
**WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542**
**AIR FORCE MATERIEL COMMAND**
**UNITED STATES AIR FORCE**

# NOTICE PAGE

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| **1. REPORT DATE** ***(DD-MM-YY)*** October 2018 | **2. REPORT TYPE** Final | **3. DATES COVERED** ***(From - To)*** 18 April 2016 – 31 October 2018 |

**4. TITLE AND SUBTITLE**
HyCIRCA: FORMAL SYNTHESIS AND VERIFICATION TECHNIQUES OF AUTONOMOUS CYBER-PHYSICAL SYSTEMS

**5a. CONTRACT NUMBER** FA8650-16-C-2611
**5b. GRANT NUMBER**
**5c. PROGRAM ELEMENT NUMBER** 65502F

**6. AUTHOR(S)**
Robert P. Goldman, Daniel Bryce, David J. Musliner, and Michael J.S. Pelican (Smart Information Flow Technologies)
Ariful Islam, Frank Pfenning, and Edmund M Clarke (Carnegie-Mellon University)

**5d. PROJECT NUMBER** STTR
**5e. TASK NUMBER**
**5f. WORK UNIT NUMBER** Q1M0

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

| | |
|---|---|
| Smart Information Flow Technologies<br>319 1st Avenue North, Suite 400<br>Minneapolis, MN 55401-1689 | Carnegie-Mellon University<br>5000 Forbes Avenue<br>Pittsburgh, PA 15213 |

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Air Force Research Laboratory
Aerospace Systems Directorate
Wright-Patterson Air Force Base, OH 45433-7542
Air Force Materiel Command
United States Air Force

**10. SPONSORING/MONITORING AGENCY ACRONYM(S)**
AFRL/RQQA

**11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)**
AFRL-RQ-WP-TR-2018-0196

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

**13. SUPPLEMENTARY NOTES**
This is a Small Business Technology Transfer (STTR) Phase II Report. Contractor has waiver STTR Data Rights.
PA Clearance Number: 88ABW-2019-1286, Clearance Date: 27 March 2019

**14. ABSTRACT**
This report was developed under a SBIR contract for topic AF14A-T06 (Formal Synthesis and Verification Techniques for Autonomous Cyber-Physical Systems).

SIFT and CMU developed HyCIRCA, a novel method for correct-by-construction nonlinear hybrid (discrete/continuous) planning and controller synthesis for autonomous systems. HyCIRCA was built by integrating mission planning and controller synthesis from SIFT's Cooperative Intelligent Real-time Control Architecture, its Playbook® Human-Computer Interface, and CMU's dReal and dReach systems for hybrid systems verification. The resulting system provides a substantial advance in efficient controller synthesis for cyber-physical systems.

**15. SUBJECT TERMS**
SBIR Report, controller synthesis, cyber physical systems, hierarchical task network planning, linear systems, temporal logic, unmanned systems, verification

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT:** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** (Monitor) |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | Laura R. Humphrey |
| Unclassified | Unclassified | Unclassified | SAR | 71 | **19b. TELEPHONE NUMBER** (Include Area Code) N/A |

Smart Information Flow Technologies
319 1st Ave North, Suite 400
Minneapolis, MN 55401-1689
Phone: 612-339-7438
Fax: 612-339-7437
Email: rpgoldman@sift.net

January 7, 2019

Dr. Laura Humphrey
AFRL RQQA
Aerospace Systems Directorate
Wright-Patterson AFB, OH 45433-7542

Subject: Contract Number FA8650-16-C-2611, Phase II STTR

Dear Dr. Humphrey:

Smart Information Flow Technologies (SIFT) and hereby waives its STTR Data Rights to all contents of the final report for subject contract. The Government is granted an unlimited nonexclusive license to use, modify, reproduce, release, perform, and display or disclose this report and the data contained herein.

We affirm that we are aware that the report may be released to other contractors to the Government and approve potential release to other contractors.

Sincerely,

[illegible]

Robert P. Goldman
Principal Investigator

## TABLE OF CONTENTS

## LIST OF FIGURES

## 1.0 SUMMARY

In Phase II of HyCIRCA, Smart Information Flow Technology (SIFT) and Carnegie-Mellon University (CMU) addressed the challenge of effectively, reliably, and safely tasking cooperating teams of autonomous cyber-physical systems (CPSs). SIFT's Playbook® interface approach provides high-level, goal-based tasking for multi-agent autonomous missions. SIFT's Cooperative Intelligent Real-time Control Architecture (CIRCA) automatically synthesizes correct-by-construction real-time, closed loop discrete controllers for multi-agent autonomous missions. To fully realize the potential of CIRCA and Playbook, we must extend them to handle CPSes with complex, non-linear dynamics. Professor Ed Clarke's group at CMU has developed a novel approach to verification of non-linear hybrid systems. CMU's dReal system uses δ-complete approximate reasoning methods combined with Satisfiability Modulo Theory (SMT) solvers to verify hybrid system models including non-linear constraints, ordinary differential equations (ODEs), and discrete dynamics. dReach is a wrapper around dReal enabling dReal to solve reachability problems for hybrid automata, a key modeling framework for CPSes. SIFT teamed with CMU to build HyCIRCA, a system to automatically synthesize and verify closed-loop hybrid controllers based on high-level mission specifications.

Contributions of the Phase II work include

(I) new methods for hybrid mission planning based on integrating SHOP2's Hierarchical Task Network (HTN) planning with dReal's SMT solving to handle complex dynamics;

(II) the addition of complex temporal logic properties as constraints on CIRCA's controller synthesis;

(III) multiple abstraction level modeling for controller synthesis with three levels of abstraction:

1. unclocked reactive discrete control;
2. timed automata for hard real time constraints; and
3. hybrid automata for correct supervisory control of complex cyber-physical systems;

(IV) More expressive model-checking for hybrid systems based on translations from Signal Temporal Logic (STL) to SMT.

## 2.0 INTRODUCTION

SIFT's Phase II HyCIRCA project builds on progress in Phase I. In Phase I, SIFT and CMU began integrating CIRCA and dReach/dReal to form the basis of HyCIRCA by developing three new techniques that are critical enablers for HyCIRCA. The key techniques proved out in Phase I were:

- New Playbook translation techniques that translate Playbook's mission goals and constraints into formal representations of mission goals, behaviors, and invariants, based on temporal logic. HyCIRCA synthesizes controllers guaranteed to satisfy these goals and constraints.
- CIRCA uses multi-abstraction modeling to tame the complexity of controller synthesis, generating goal-achieving mission plans in an abstract state space and then verifying their correctness using a more accurate timed automaton model. In Phase I of HyCIRCA, we extended this approach to handle a third level of detail, a complex hybrid systems model that captures the full non-linear dynamics of the domain.
- Another key to CIRCA's controller synthesis is exploiting verifier counterexamples, through culprit identification, to backjump directly to faulty decisions. Culprit identification maps steps in a verifier counterexample into decisions in the controller synthesis process. In Phase I we developed culprit identification techniques that extend to dReal's hybrid system counterexamples.

In Phase II SIFT and CMU extended these capabilities to an end-to-end integrated prototype of HyCIRCA, whose architecture is shown in Figure 1. We tested and evaluate HyCIRCA on a family of scenarios based on the multi-unmanned aerial vehicle (UAV) firefighting scenario we used in Phase I, and additional small test cases that were designed to demonstrate key capabilities.

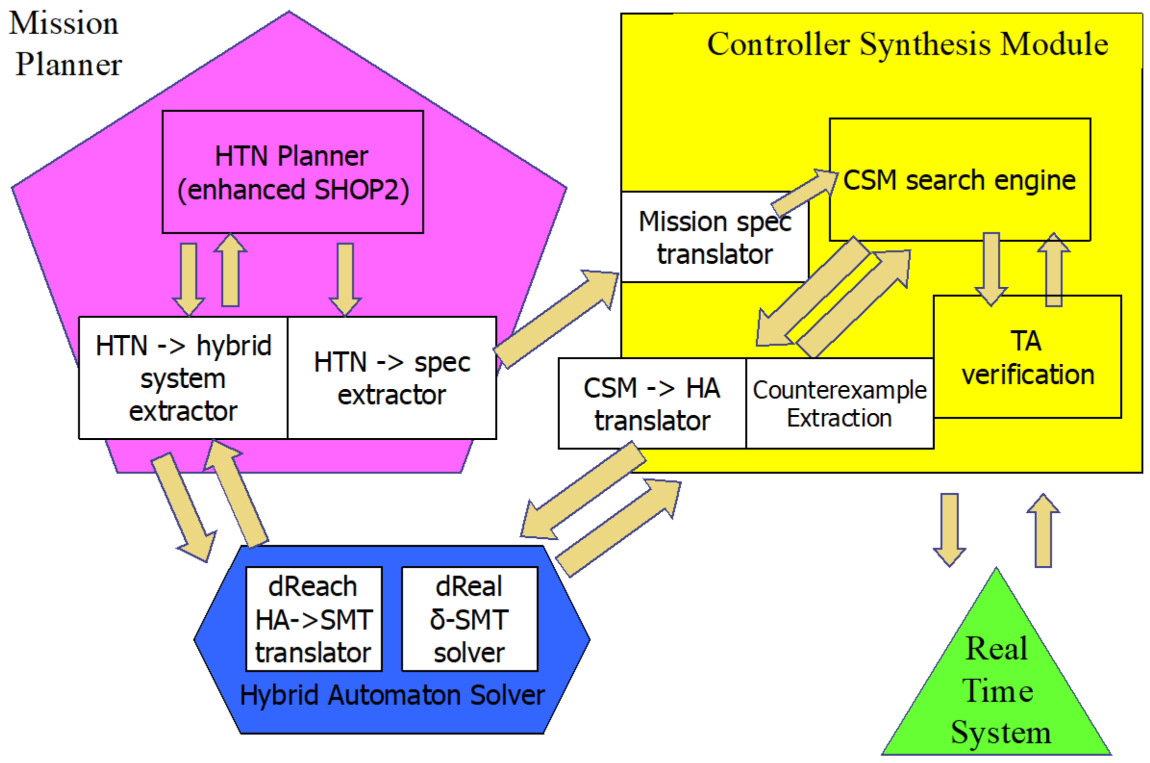


**Figure 1 HyCIRCA system architecture.**

In Phase II we also explored additional research challenges identified in Phase I. Significantly, we extended the integration of dReal hybrid automaton modeling from the controller synthesis process to the mission planner. We developed optimized SMT encodings of hybrid verification and parameter synthesis problems to improve efficiency and further scale HyCIRCA. Notably we developed new techniques, based on Monte Carlo Tree Search, for exhibiting sound solutions to parameter synthesis problems. These new techniques were needed to overcome limitations of δ-soundness, which was only capable of soundly checking *avoidance,* not achievement. We also developed new heuristics of SMT solving specific to planning applications. Meanwhile, CMU developed encodings for Signal Temporal Logic (STL) properties into SMT formulas that could be checked by dReal. This overcame another limitation of dReach/dReal, which were previously only able to check (un)reachability. We had originally planned to make an incremental variant of dReal for checking CIRCA controllers, along the lines that we have used in our work with temporal automata. However, our experience with the high computational cost of dReal and the nature of efficient SMT encodings led us to abandon this approach in favor of an approach in which the hybrid checker would only be used in a post-check of controllers that had already been found to be (approximately) satisfactory by a temporal automaton analysis. This fit well with our approach to checking more complex temporal logic-style goals in the CIRCA controllers.

## 3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

In this section we discuss our methods, assumptions, and procedures. The methods and assumptions are covered in the three background sections, that cover the technologies that were combined, and enhanced, to build HyCIRCA: CIRCA, Playbook, and dReal and dReach. As we introduce these technologies, our assumptions – i.e., the way we modeled systems and the environment for high level hybrid control – will also be discussed. We will end this section by introducing the technologies we have developed in the Phase II research: new methods for Mission Planning, Controller Synthesis and Verification, and SMT encodings for Signal Temporal Logic.

### 3.1 CIRCA Background

This paper provides a brief overview of the research to date that has focused on CIRCA, the Cooperative Intelligent Real-Time Control Architecture. CIRCA is one of the first fully-implemented Artificial Intelligence (AI) planning and execution architectures that supports hard real-time performance guarantees. CIRCA research has explored a broad spectrum of related areas including reliable plan execution semantics, adversarial reasoning, heuristic search guidance, the link between formal verification and planning, meta-control of deliberation in time-constrained domains, probabilistic planning, and multi-agent planning and coordination. In this paper we show how these topics tie together and relate to the general problem of building embeddable intelligent systems that provide formal, provable properties while retaining the complex, unpredictable elements of state-of-the-art intelligent planning and scheduling algorithms.

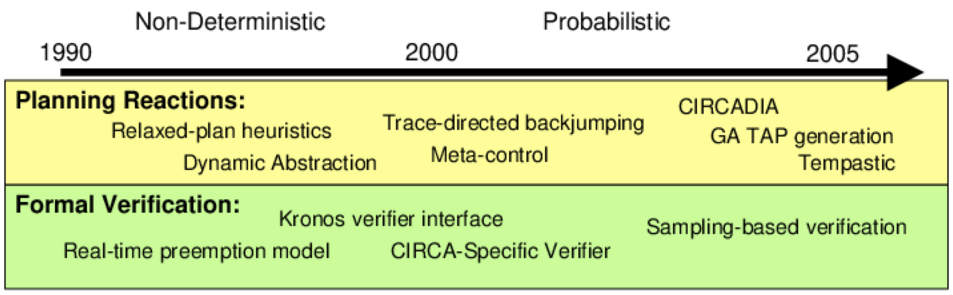


**Figure 2 CIRCA research timeline.**

CIRCA's performance guarantees are constructed and enforced by components that are based on an underlying theory of reactive plan execution; the theory describes how planned reactions interact with an external world (including external sources of change). The CIRCA executive is carefully designed and implemented to enforce the semantics of the plan execution theory.
To set the context, we briefly review the history of CIRCA research. As shown in Figure 2, for over 25 years, CIRCA research has explored numerous topics related to real-time intelligent control based on theoretically-grounded performance guarantees. Most of the CIRCA capabilities described in this paper are still active and available in the current codebase, so we describe them in the present tense, even though some were developed more than 25 years ago.

### 3.1.1 "Classic" CIRCA

In the beginning, we were mainly interested in automatically building real-time control plans to control things like robots in dangerous and adversarial environments. The key aspect of such domains is the potential for catastrophic failure— if the controlled agent (*e.g.*, a UAV) does not respond to some threat (*e.g.*, a surface-to-air missile launch) within a certain time limit, then it risks failing completely. So "Classic" CIRCA is designed to reason about such domains and automatically build and execute reactions that defeat such threats [1][2].
CIRCA is an autonomous, self-adaptive control architecture designed specifically for mission-critical domains. As illustrated in Figure 3, CIRCA combines on-line planning and scheduling systems in its AI Subsystem (AIS) with a very simple, very predictable real-time plan executive, the Real-Time Subsystem (RTS). CIRCA dynamically creates time-constrained reactive control plans -- cyclic loops of Test-Action Pairs (TAPs) -- based on its expectations about future world states and its own potential actions. The RTS is responsible for executing TAP plans in a completely predictable fashion, so that their execution matches the model used by the AIS. The RTS meets this criterion for TAP execution because it has no other function; it simply loops over the cyclic schedule of TAPs, testing and executing them repeatedly. Even communication into and out of the RTS is encapsulated within TAPs, so that all RTS activity is scheduled explicitly.

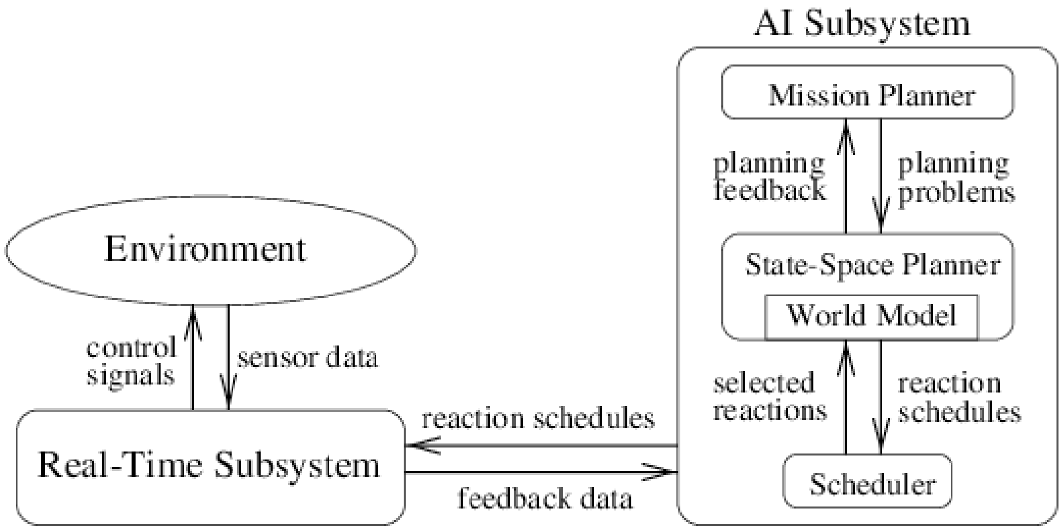


**Figure 3 The CIRCA architecture.**

The Planner and Scheduler, on the other hand, perform the complex, unpredictable reasoning required to develop guaranteed control plans, and the performance of these subsystems must not interfere with the RTS' predictable execution. To achieve this isolation, each control plan executed on the RTS is designed both to achieve system goals and to ensure system safety throughout the range of environmental states that are anticipated during and after the accomplishment of the goals. In other words, the RTS keeps the system safe while the Planner and Scheduler try to build the next control plan; the planning operation is *not* constrained to meet domain deadlines.

### 3.1.2 Theoretical Foundations of CIRCA

The full details of the underlying CIRCA theory and executive design are beyond the scope of this paper but are available in several other publications. An intuitive treatment of the theory and executive design is available in [1], while [3] provides a more formal description in terms of timed automata. Here we briefly overview the theory underlying CIRCA's performance guarantees, to set the context for describing the architectural evolution.

Unlike most planning and scheduling systems that build plans as linear or partially-ordered action sequences, CIRCA builds plans that are actually reactive *controllers*, designed to sense and react to different world states (situations) within strictly-enforced time bounds. Each TAP has a boolean test expression that distinguishes between states where a particular action is and is not to be executed, and a timing constraint specifying the maximum time allowable between TAP executions. In this context, a "state" is a particular assignment of values to features or variables that describe the world and which can be sensed by the CIRCA-controlled agent. A sample TAP and an associated TAP schedule loop are shown in Figure 4. This example was taken from a domain controlling redundant spacecraft inertial reference units (IRUs).When executing a TAP, the RTS evaluates the test expression and, if it returns true, the RTS executes the corresponding action. CIRCA's Scheduler module uses the TAP timing requirements when it builds looping TAP schedules.

```
#<TAP 2>
Tests: (AND (IRU1 BROKEN)
  (OR (AND (ACTIVE_IRU NONE) (IRU2 ON))
      (AND (ACTIVE_IRU IRU1) (ENGINE ON))))
Action: select_IRU2
Max-delay: 2 seconds
```

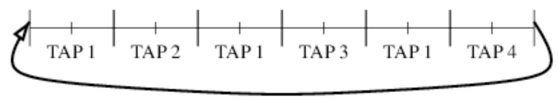


**Figure 4 A sample Test-Action Pair and TAP schedule loop.**

The world model and planning algorithm that the AIS uses to develop TAP plans are detailed in [2]. For our purposes, it is sufficient to understand that the model is a modified state/transition graph in which states correspond to complete descriptions of the world (modulo some level of abstraction), and three types of transitions represent the ways the world can change. *Temporal transitions* represent time and ongoing processes. The timing behavior of a temporal transition is related to the rate of the process it represents: for example, the process of activating a spacecraft's inertial reference unit may take some minimum amount of time to complete. *Event transitions* represent occurrences outside the agent's control, while *action transitions* represent the intentional actions that can be taken by TAPs. Transitions have precondition and postcondition expressions that describe how they can link together states in the planned world model. CIRCA can control the timing behavior of action transitions by setting the timing constraints of the TAPs it builds.

To build plans, CIRCA begins with a set of goal descriptions, a set of initial world states, and a set of transition descriptions that detail the types of events, actions, and processes possible in the world. Some of the transitions are identified as leading to a distinguished failure state, and CIRCA must build a TAP plan that makes failure unreachable while also doing its best to achieve the other goal conditions. The basic planning algorithm conducts a fairly standard type of heuristically-guided forward search with backtracking, expanding the set of reachable states by applying the uncontrollable temporal and event transitions and deciding on an action choice for each state. An action can be planned to *preempt* undesirable transitions (*e.g.*, temporal transitions to failure) by constraining the action to execute quickly enough to definitely occur before the undesirable transition could possibly occur. This notion of preemption is the core aspect of CIRCA planning that allows the system to reason about real-time performance guarantees and system safety.

### 3.1.3 CIRCA's World Model

Several key aspects of this real-time planning and control problem distinguish Classic CIRCA from most AI systems,[1] including:

- Exogenous Events — Unlike most planning systems, CIRCA considers exogenous sources of change in its environment, including adversaries. Because the focus is on making real-time safety guarantees, all exogenous processes and events are assumed to possibly happen at any time they could— in fact, the system takes Murphy's Law to the extreme, expecting that anything bad that can happen will happen, at the worst possible time.
- Time and Preemption — CIRCA tries to build plans that prevent failures through one primary mechanism: disabling the preconditions that allow an adversary (or the environment) to cause a failure. In addition to handling the logical elements of this sort of planning, CIRCA also must ensure that its safety-preserving actions will be taken quickly enough. In other words, CIRCA has to ensure that the right action is taken at the right time, and this timing may be dictated in part by consideration of the uncontrollable environment/adversaries. Unlike most temporal planning models, CIRCA does not label state with specific times; instead, it uses a purely relative (and non-Markov) temporal model that allows the system to compactly represent continuously-executing control loops (e.g., as long as you're flying, any time someone shoots a radar-guided missile at you, deploy chaff and begin evasive maneuvers).
- Nondeterministic Actions— CIRCA's action models can be nondeterministic, having multiple sets of postconditions. For example, the model of a start-engine action may either result in the engine being started or not. Combining the unique CIRCA temporal model with the notion of indexical-functional variables [5] and nondeterministic actions allows CIRCA to efficiently reason about looping plans (reactive controllers) without overly-precise models of system dynamics (e.g., to hammer in a nail, keep hitting it until it is flush)[6].
- Continuous Embedded Operation — CIRCA is also designed to persist through changing missions that cannot be entirely pre-planned, so planning and execution occur concurrently and new plans need to be sent down to the RTS and begin execution without sacrificing the system's safety guarantees. Thus CIRCA can reason explicitly about the safe transfer of control between two different reactive controllers, implemented by different TAP schedules.

### 3.1.4 CIRCA and Formal Verification

CIRCA's world model is non-Markovian in the sense that the abstracted temporal model means that the path of transitions followed to reach a state can affect which transitions are possible out of that state, because of delays. For example, preempting a temporal transition to failure from

[1] See [4] for an overview of Real-Time AI approaches

one state may not disable that failure transition, but instead lead to a new state where it is still applicable; in this case the process represented by that temporal transition will have continued to run, so the safe time remaining in the new state is reduced. Naturally, this complicates the process of reasoning about the temporal model, and motivates our use of formal model checking to verify the required preemption properties that are necessary to ensure that a plan is guaranteed to avoid failure and keep the system safe [7].

Each time the CIRCA State-Space Planer (SSP) makes a heuristic decision about what action should be taken in a state, it uses a verifier to confirm that failure is not reachable and that all the planned preemptions will occur as expected. This means that the verifier will be invoked before the plan (controller) is complete. At such points we use the verifier as a conservative heuristic by treating all unplanned states as if they are "safe havens." Unplanned states are treated as absorbing states of the system, and any verification traces that enter these states are regarded as successful. Note that this process converges to a sound and complete verification when the controller synthesis process is complete.

Incremental Verification: Our earliest efforts to incorporate model checking verifiers used off-the-shelf systems such as Kronos [8][9]. However, because those systems are designed for batch verification of system designs, they are tremendously inefficient when used in the inner loop of the CIRCA planning engine, completely re-building their verification traces as each new action decision was made. Therefore, we implemented a CIRCA-Specific Verifier (CSV) that takes advantage of several key aspects of the CIRCA planning problem and is fully incremental. The CSV system can be orders of magnitude faster than the Kronos-based approach, without sacrificing verification accuracy or precision (in fact, the CSV has a more accurate model of the executive's behavior than the atrophied Kronos interface).

Trace-Directed Backjumping: When the verifier finds that the distinguished failure state is reachable, it can return a trace illustrating a path to failure. By mapping this failure trace onto the search stack choice points, CIRCA can pinpoint the decisions that are responsible for failure, and *backjump* to revise the most recent implicated decision. This backjumping avoids revisiting more-recent but irrelevant decisions and can considerably improve the efficiency of the search *without sacrificing completeness*.

### 3.1.5 Heuristic Search

Despite its temporal abstractions and other advantages, the CIRCA state space is highly exponential and explodes quickly. Our efforts to manage this complexity have resulted in several research contributions:

- Plan Graphs for non-Closed-World Models — As with all state-space searches, heuristic guidance is critical. Fortunately, the early work on plan-graph (or "relaxed plan") heuristics occurred just as CIRCA matured. Based on McDermott's original work [10], we developed our own planning graph heuristic that combined the now-standard relaxation/abstraction elements (*e.g.*, ignoring negative interactions) with CIRCA-specific elements including nondeterministic outcomes and exogenous events.
- Dynamic Abstraction Planning (DAP) — The intuition behind DAP is simple: in some situations, certain world features are important, while in other situations those same

features are not important [11]. By representing only the important features, DAP allows CIRCA to avoid enumerating many unique but functionally-equivalent states. DAP begins with a maximally abstract world model (only distinguishing failure and non-failure states) and incrementally adds more information to a state's representation when necessary to improve the plan. By automatically selecting the appropriate level of abstraction at each step during the planning process, DAP can significantly reduce the size of the search space. It should be noted that our development of DAP is an independent discovery of the Counter Example Guided Abstraction Refinement (CEGAR) technique from model-checking [12].

- "Bad Smell" — Even with backjumping, the SSP might waste time repeatedly attempting to find a solution for "failed" states. Note that, because CIRCA's state space model has non-Markov temporal semantics, the action choices (including reaction timing) that may occur before a failed state can be the cause of an anticipated failure, and it may be possible to revise those earlier action decisions in a way that makes a "failed" state no longer a failure. So these states should not be completely eliminated from the search for a good plan. For this reason, we wanted to control the SSP search so that it would try to avoid states that had previously failed. We gave such states a "bad smell," so that the planner would prefer actions that avoided them wherever possible. This mechanism and its motivation are roughly analogous to aspects of Tabu search [13].

### 3.1.6 TAP Scheduling

The CIRCA TAP scheduling problem is fairly simple, but it has two unique aspects. First and foremost, the tasks being scheduled are automatically generated, so they are not as well-organized and optimized as human-generated tasks might be. One simple but confounding result is that TAP timing specifications do not fall on simple harmonic frequencies, so the least common multiple (LCM) of the TAP periods is generally extremely large. As a result, traditional schedulers that attempt to schedule calendars of task executions out to the LCM of the task periods (such as the Maruti scheduler [14]) will often be completely unable to deal with TAPs.

The second special aspect of CIRCA's scheduling problem is that, instead of a period specification, each TAP is given to the Scheduler with a specification of the maximum acceptable invocation separation. CIRCA specifies invocation separations because synchronous behavior is not necessary for the control tasks it plans. These unique constraints led us to develop novel TAP scheduling approaches that can significantly outperform simple adaptations of existing periodic-task scheduling algorithms [15].

### 3.1.7 Meta Control

The Adaptive Mission Planner (AMP) is responsible for the highest-level control of a CIRCA agent, managing the agent's long-term goals and the agent's deliberation activity. The agent's long-term mission may be divided into phases, each of which requires its own safety-preserving and goal-achieving reactive plan. For example, our UAV scenarios include missions that have phases such as ingress, attack, and egress. The ingress phase is distinguished from the attack phase both by the characteristics of the flight path (*e.g.*, a nap-of-earth stealthy approach vs. a popup maneuver very near a target) and by the expected threats (*e.g.*, the types of missile threats present at different altitudes) and goals (*e.g.*, reaching the target zone vs. deploying a weapon).

The AMP reasons about long-term goals, problem structures, and approaching deadlines to decide what the near-term goals should be, and what problems the near-term reasoning should be focused on. Because the phase plans may need to be created under time pressure as a mission executes, the AMP can make tradeoffs in which mission goals and safety threats are considered in each phase. The AMP's meta-control functions intelligently allocate the CSM deliberation effort to different mission phases, solving the problems with differing levels of safety and goal-achievement depending on how much deliberation time is available [16][17][18].

We take an approximate decision-theoretic approach to the CIRCA deliberation scheduling problem: decision-theoretic, because we attempt to optimally allocate the CSM's reasoning time; approximate because full formulations of the problem are intractable, and some formulations involve an infinite regress. CIRCA's earliest meta-control used coarse-grain modifications to the problems it solved to trade planning time against plan quality [17]. In later work [16], we developed a Markov Decision Process (MDP) model of the deliberation scheduling problem, controlling which of several possible problems the system should work on at any time. Since the MDP may be very large and difficult to solve, we also presented greedy (myopic) approximations to the optimal solution. In those experiments we showed that a discounted myopic approximation technique provided good performance with very limited computational costs. We also compared the performance of the discounted greedy approximation with other strawman agents that attempt to manage deliberation using easy-to-compute heuristics.

### 3.1.8 Distributed CIRCA

We have also investigated methods for extending the real-time performance guarantees that single-agent CIRCA provides to small teams of agents. At the highest level, the AMP's primary responsibility is managing an individual agent's tasks and coordinating with other agents to achieve the overall team mission. The AMP does this by determining what tasks are its responsibilities through negotiation with other cooperating agents, and then arranging to have plans (controllers) generated to successfully address those tasks during the execution of the mission.
In this context, a team of CIRCA agents must arrange to have different agents responsible for different goals and threats, depending on their available capabilities and resources, *e.g.*, Electronic Counter Measure (ECM) equipment and weapons loadout. Using a Contract-Net-like arrangement [19], the AMPs submit bids to handle these responsibilities. For each mission phase, the CIRCA agents must have plans, or controllers, that are custom-designed (either before or during mission execution) to execute the mission phase and make the best possible effort to achieve the goals and defeat the threats associated with the phase. When necessary, the agents can build coordinated plans that communicate at runtime to ensure real-time coordination across a team of agents.

The most critical form of coordination for real-time safety guarantees is *coordinated preemption*, in which a set of complementary reactions executed by distributed agents detect threats and take action to preempt hazardous transitions. Two key issues underlie these "You sense, I'll act" plans:

1. Planned communication – The agents must recognize the need to explicitly communicate (both sending and receiving) at a rate fast enough to satisfy the coordinated preemption timing constraint. In our example, the sensing agent must agree not only to detect the hot spot fast enough, but also to tell the other agent about the opportunity quickly enough. Likewise, the acting agent must focus sufficient attention on "listening" for a message

from the sensing agent at a high enough frequency that it can guarantee to both receive the message and act on the opportunity, all before the deadline.

2. Distributed causal links – The distributed agents must be able to represent and reason about changes to their world that are not directly under their control, but which are predictable enough to be relied upon for a preemption guarantee. For example, in our scenario, the sensing agent must rely on the acting agent to take the appropriate action in time to guarantee that the data collection is performed in time. In complementary fashion, the acting agent must construct a plan that honors its commitment to the acting agent. If one of the agents cannot construct a plan that satisfies its commitments, it must inform the others.

### 3.1.9 Probabilistic CIRCA

As we applied CIRCA in increasingly demanding applications, it became apparent that the architecture's theoretically-strong stance on performance guarantees was not flexible enough to deal with many real-world domains. In some problems, some action choices expose an agent to more possible failure-causing events than preemptive actions can be assured to avoid. More generally, there are many domains that are too dangerous to ever ensure 100% safety; in these domains, we'd like the system to make tradeoffs between mission performance and safety criteria.

Faced with a domain that cannot be made 100% safe, Classic CIRCA fails to find a plan. And even if a 100% safe plan can be found, it may be less than satisfactory. For example, when we built a domain model for Classic CIRCA to control an aircraft in which the landing gear could fail, and there was no way to repair the landing gear or land safely with it broken, the system quickly constructed a safe plan: sit on the runway and don't take off. Unfortunately, this plan did not achieve any of the non-safety-related mission goals.

Trading off some degree of safety in order to achieve important mission-related goals requires that CIRCA make careful choices about which potential failures it is most safe not to be prepared for. Probabilistic CIRCA does this by trimming away enough of the most unlikely transitions to failure that the remaining ones can be guaranteed preempted. When developing this variation of CIRCA, we developed a variety of techniques for estimating probabilities of reaching states and traversing transitions to failure; the non-Markovian aspects of CIRCA in particular make it challenging to assess the probabilities of transitions to failure that persist across sequences of states.
The cumulative probabilities of the transitions that have been trimmed to achieve a subset of the space that can be safely controlled indicate the degree of risk that would be incurred should the control plan be followed. With this information, informed tradeoffs between risk and mission goals are possible [20].

This in turn raises the question of what can or should happen if one of the trimmed transitions actually occurs, putting the agent into a state that its control plan isn't prepared to handle. Our extensions consider all of the states just outside of the control envelope and develop tests to detect when such a state has been reached. A TAP is formed with this test, where the corresponding action involves replacing the current TAP schedule with another one that is intended to at least maintain safety. For example, in our aircraft domain, if the landing-gear failure was trimmed from the initial control plan, the TAP detects this failure and implements a control plan for circling the airport, with the expectation that this will buy time for the AIS to use in formulating an appropriate control plan for recovering

from this situation. Alternatively, the appropriate control plan for this contingency might have been developed ahead of time, in which case it would be swapped in immediately [21].
GSMDPs: To capture a more powerful and precise notion of probabilistic guarantees, we began exploring a modified CIRCA world model in which the fixed worst-case delays associated with transitions were replaced by probability distributions of possible delays. The idea then is to build plans that allow a certain level of safety risk, as long as the overall probability of failure remains below some specified threshold. It turns out that the resulting model is a Generalized Semi-Markov model, and is thus extremely intractable. Even assessing whether a particular controller meets the safety threshold is not analytically computable. So first we developed a sampling-based approach to probabilistic verification, deriving formal bounds on how many simulated plan executions (samples) had to be generated to ensure, with a certain level of confidence, that a plan met the desired safety threshold [22]. We then explored various approaches to extending the CIRCA CSM to build plans in this probabilistic model, as well as other techniques [23].

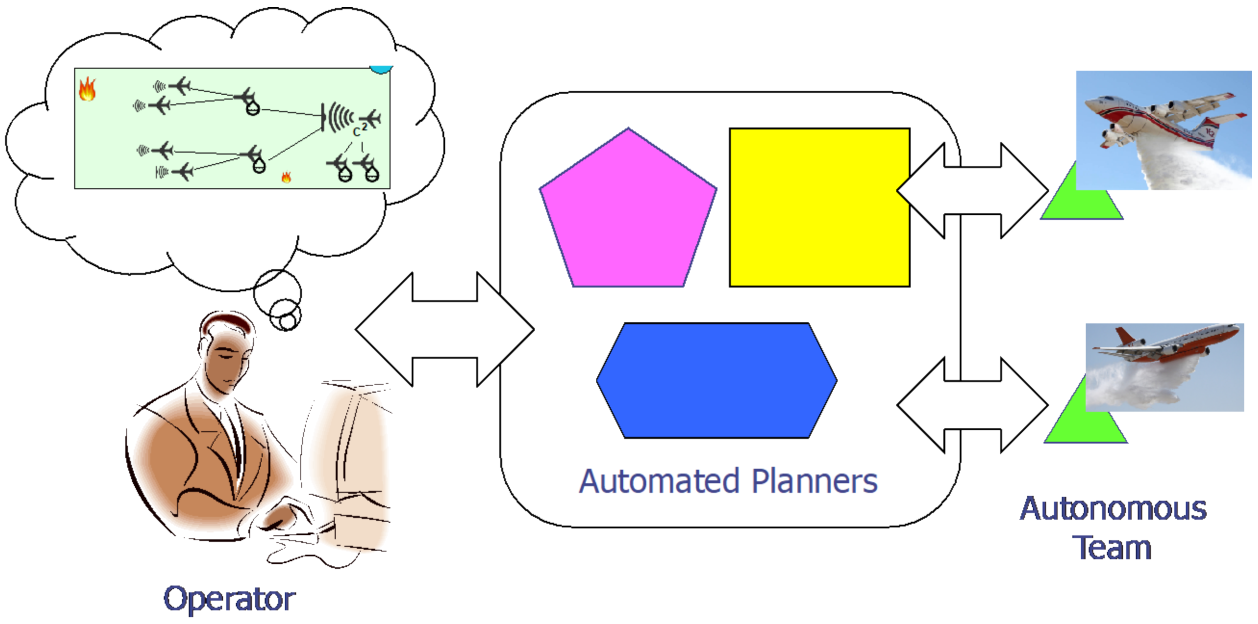
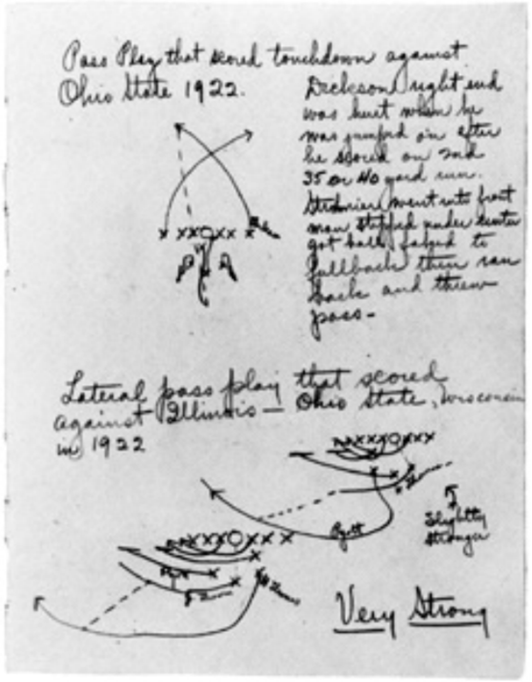


**Figure 5 HyCIRCA architecture.**

## 3.2 Playbook Background

We will leverage SIFT's 15 years of experience developing Playbook-based systems for command and control of teams of manned and unmanned platforms [24][25] to provide the guiding architecture and user interaction metaphor for HyCIRCA as shown in Figure 5. Playbook systems revolve around the concept of "calling a play." Calling a play is an efficient way to capture a commander's intent or goals, including constraints and restrictions s/he wishes to impose. The Playbook metaphor has been a natural way to effectively task teams since at least 1927, when Alonzo Stagg collected plays for his University of Chicago Maroons (see Figure 6). With the commander's high-level intent established, automated or mixed-initiative planning systems then "fill in the details" of the play, designing a plan to accomplish the goals within the specified constraints. Plans can contain hierarchical abstraction: they may specify sub-goals that are assigned to lower-level echelons or units, for those units to treat as goals that trigger local, context-sensitive planning and execution behaviors. The assignment of lower-level goals to units may be either specified from above, or left unspecified and then accomplished via inter-unit negotiation. This hierarchical abstraction supports supervisory control that scales to very large organizations of heterogeneous assets.

We have used our Playbook approach to develop some of the most sophisticated multi-platform command and control capabilities ever demonstrated in simulation or live flight. For example, for DARPA's Heterogenous Urban Robot Teams (HURT) program, SIFT personnel developed a multi-UAV planning and control system that provided high-level Playbook tasking capabilities to multiple human operators commanding teams of up to six military-grade UAVs in Technical Readiness Level (TRL) 7 live flight experiments conducted with Marine training forces. Marine operators could request high-level Plays such as surveil-area and track- target. Our planning and control system automatically allocated platforms to tasks, planned and deconflicted routes, actively controlled real-time plan execution including cross-platform coordination, and replanned in sub-second time as goals changed and mission contingencies occurred.



Pass Play that scored touchdown against Ohio State 1922.
Dickson right end was hurt when he was jumped on after he scored on 35 or 40 yard run. Strohmeier went into front man stepped under center got ball faked to fullback then ran back and threw pass-
Lateral pass play that scored against Illinois — Ohio State, Wisconsin in 1922
Slightly stronger
Very Strong

**Figure 6 1927 playbook for the University of Chicago Maroons.**

Under the hood, Playbook systems depend on advanced automated planning systems to translate the commander's intentions to an executable mission plan and, in most cases, to synthesize low-level controllers for the autonomous members of the commander's team (see Figure 5). For HyCIRCA, we will integrate the HTN planning capabilities of the SHOP2 (Simple Hierarchical Ordered Planner, v. 2) planner, the real-time discrete controller synthesis of CIRCA, and the hybrid reasoning of dReal to create a Playbook system with the unique ability to accept mission specifications expressed in temporal logic and produce autonomous controllers that obey the discrete and continuous constraints on the mission. In the following sections, we describe each of these components in greater detail.

HyCIRCA's top-level Mission Planner (MP) interacts with the operator and the lower-level planners to direct the transformation of an incompletely specified play into a set of executable controllers. The well-proven SHOP2 HTN planner is the foundation of the HyCIRCA MP. SHOP2 is a modern HTN planner with a clean implementation that has performed well in past planning competitions. Another advantage of the SHOP2 planning system is that it is available under a generous open-source license and is maintained at SourceForge (by SIFT). Like other HTN planners, and unlike first-principles planners, SHOP2 searches top-down from a task or set of tasks, rather than chaining together primitive actions (see Figure 7). SHOP2 and other HTN planners decompose complex tasks into more primitive sub-tasks (methods), thus building a plan tree that terminates at leaves corresponding to primitive actions (operators).This method of operation has the advantage of providing an easy way to capture standard operating procedures (SOPs) and trajectory constraints, ensuring that SHOP2 will produce plans that are executable, and avoiding repeated derivations of SOPs from first principles.

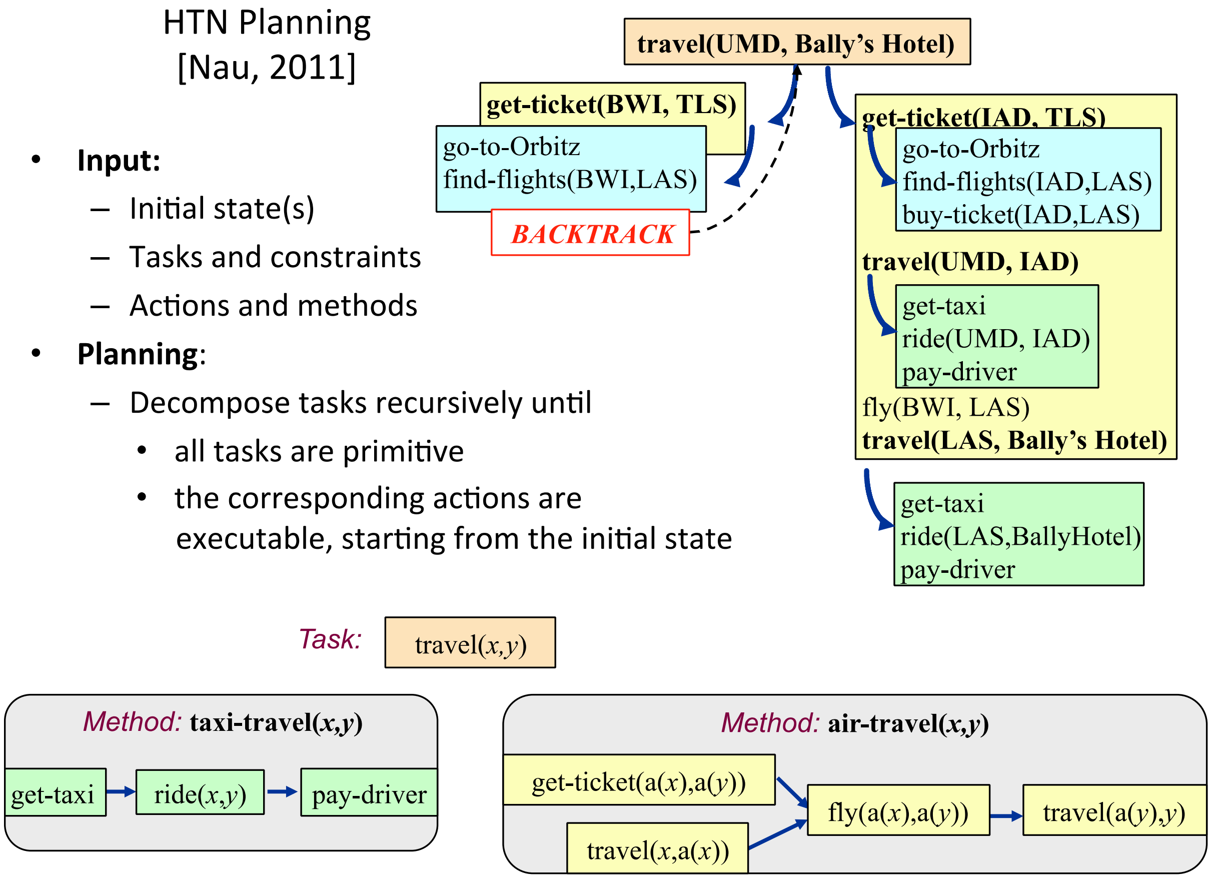


**Figure 7 HTN planners build plans top-down, by task decomposition.**

The HyCIRCA MP will output a set of tasks to be accomplished by the autonomous vehicles and constraints on the tasks. Constraints may include temporal deadlines, resource limits, and synchronization requirements. For example, in an aerial firefighting domain the system might need different controllers for mission phases such as takeoff, ingress, coordinated monitor and extinguish, and landing. Those controllers would be responsible for accomplishing different types of goals and handling different types of failures in each phase. The MP builds problem specifications that describe these goals and possible failures, as well as the environment in which they are to be accomplished, the controlled-asset's capabilities, and any agreements about what other cooperating assets will be doing. The MP then tasks the CSM and dReal to develop a reactive controller to accomplish those goals and handle associated threats. If a controller cannot be found that meets all of the constraints, a culprit or counter example will be provided to the MP, which may automatically modify the candidate plan or request constraint relaxation from the operator.

For HyCIRCA, we will add a significant new capability to for the MP: the ability to specify plan requirements as expressions in temporal logic. For example, UAV operators may wish to express complex goals that have additional structure such as sequential goals ("do A before B"), operating constraints ("approach this fire only from this direction"), etc. In general, operators may wish to express goals that require the full complexity of logics such as Linear Temporal Logic (LTL) [23]. For HyCIRCA, we extended the MP to handle these more complex goals and provide them to the other planners and solvers for verification.

## 3.3 dReal and dReach Background

**dReach** is a bounded model checker for hybrid systems that uses the dReal SMT solver as a subroutine. dReach reads problem instances consisting of a hybrid system and a property, and then uses a variety of translations to SMT. The default dReach approach encodes an SMT problem for each step bound $k \in \{0,1,\dots,n\}$, and solves these until it finds a $k$ where the property is not satisfied or it proves the property is satisfied for each value of $k$. Each SMT problem encodes all possible runs on the hybrid system with $k$ jumps and the negation of a safety property. If unsatisfiable, the negated safety property is not reachable by a $k$ jump run (i.e., safety is guaranteed for I).

**dReal** is an SMT solver designed for verification of Cyber-Physical Systems (CPSs). CPSs are complex systems that combine continuous and discrete behaviors. While the past decades have seen the development of highly scalable formal methods for reasoning about discrete systems, we are facing the hard problem of handling the continuous components in CPSs and their interaction with the discrete part. This is a challenge to the state of the art in both theory and practice. On a theoretical level, to handle the continuous components that evolve in time and space, we have to reason about computations over the real numbers. This is missing from the existing theory in formal verification. In practice, algorithms that have been proposed for reasoning about continuous systems mostly rely on very expensive algebraic methods, which do not scale to anything more than a few variables due to inherent complexity.

In recent work [26][27] at CMU, we proposed a new framework to meet all these challenges. Since the key is to understand continuous systems and computations over the reals, we based our theory on the rigorous foundation of computable analysis [28], the study of computability and complexity questions of continuous functions over the real numbers. Computable analysis provides the theory for understanding numerical algorithms with logical rigor. Note that the use of numerical algorithms, which are ubiquitous in engineering, has always been a missing piece in formal verification. The reason is that, while numerical algorithms can be highly scalable on a wide range of hard problems, they inevitably introduce numerical errors that would render them unusable for formal verification, which aims for definitely correct answers [29]. However, we realized that there are in fact ways to control numerical errors in a formal sense and ensure correctness of numerically-driven procedures for verification. We accomplish this by investigating the notion of numerical perturbations on logic formulas. The traditional decision problem asks whether a logic formula is “true” or “false”. Instead, we ask the question of whether a formula is “true”, or “its perturbed form is false”. This minor change lets us allow imprecision (in the form of a real parameter $\delta > 0$) in the decision algorithms for logic formulas, while controlling it to be one-sided and bounded. We call this notion of decisions with errors the $\delta$-decision problem and the corresponding algorithms $\delta$-complete. We extensively investigated them in [26] and proved key results stating that, while the logic theories over the reals with any interesting functions are either highly intractable or undecidable, the $\delta$-decision versions of the problems avoid undecidability and have a much lower complexity.

$\delta$-complete decision procedures are exactly what we need to advance the state-of-the-art in formal verification of CPSs. These procedures allow us to use the full power of numerical methods, while ensuring that numerical errors never cause correctness problems. For instance, in bounded model checking, we can translate the safety property of a system to a logical formula, P, such that if P is false, then the system is safe. Now, note that in $\delta$-complete decision procedures, errors can only occur when the procedures decide a formula is true. Thus, when we conclude a system is safe, we are absolutely correct, and no numerical errors can affect the answer. On the other hand, when we say a system is unsafe, due to the possible errors in the procedures, we can conclude that either the system is indeed unsafe, or that under some perturbations the system would become safe. In the latter case, since the perturbations are controlled by the parameter $\delta$ which can be arbitrarily small, a minor change in the environment would render it unsafe, and consequently the system should not pass the verification stage anyway, since it is not robust. In this way, we can now take full advantage of scalable numerical methods in our reasoning tools.

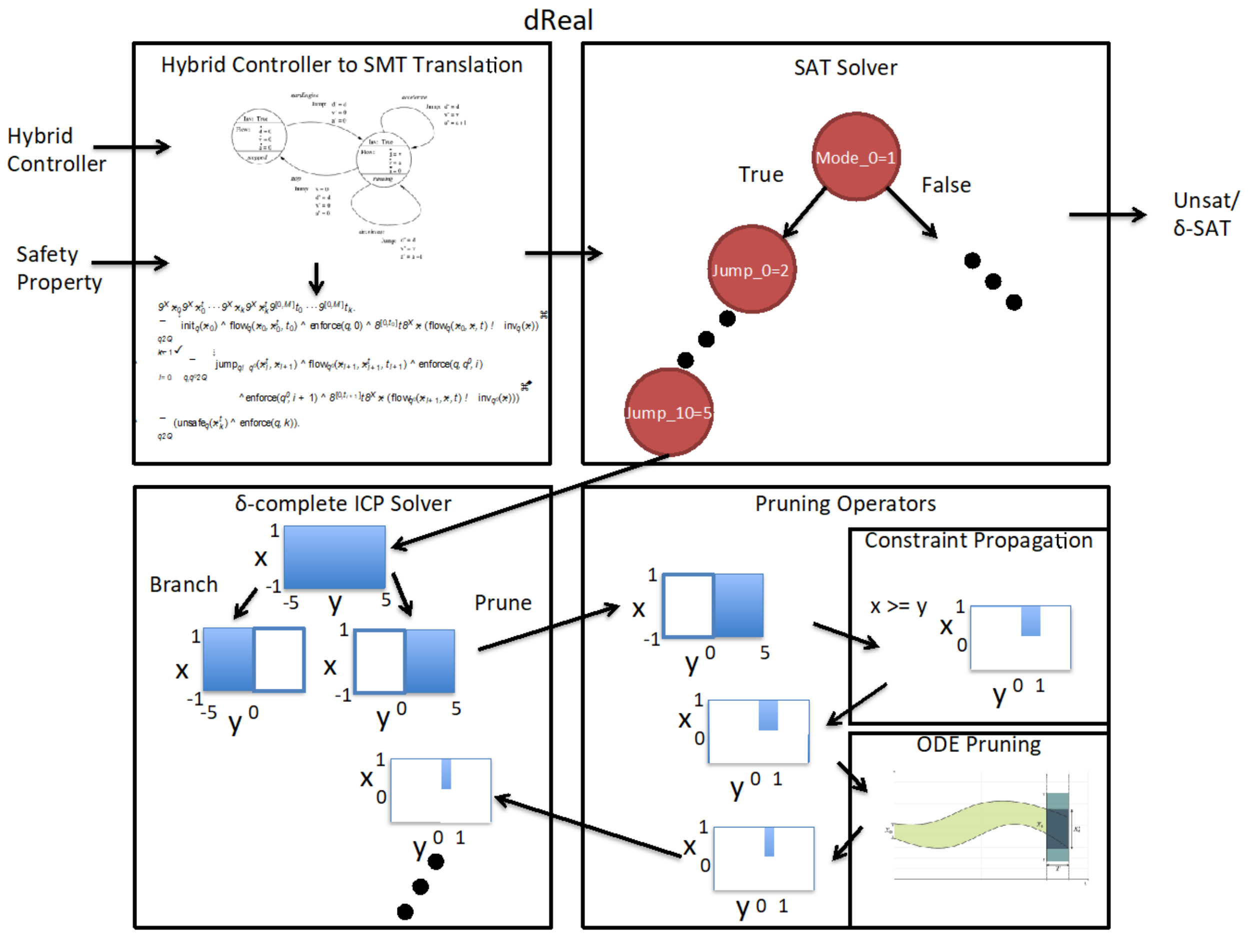


**Figure 8 dReal combines SAT and δ-complete numerical reasoning.**

We solve such verification problems using decision procedures in the framework of Satisfiability Modulo Theories (SMT) [30]. SMT combines powerful Boolean satisfiability solvers with specialized theory solvers, such as those needed to reason about numeric constraints. Our dReal SMT solver for the hybrid decision problem [31][30] has shown very promising results. As shown in Figure 8, dReal implements a tight integration of Boolean Satisfiability (SAT) solving and Interval Constraint Propagation (ICP) solving. It translates a hybrid controller and safety property into an SMT formulation. It solves the SMT problem by first assigning values to Boolean variables in its SAT solver, and then applying ICP to the feasible intervals of numeric variables. In ICP, the numeric intervals are split (via branching) and pruned (via constraint propagation and ODE pruning) until shown δ-satisfiable or unsatisfiable.

dReal proves safety properties for hybrid automata defined as follows:

$$H = \langle X, Q, \{\mathsf{flow}_q(\vec{x}, \vec{y}, t) : q \in Q\}, \{\mathsf{inv}_q(\vec{x}) : q \in Q\}, \{\mathsf{jump}_{q \to q'}(\vec{x}, \vec{y}) : q, q' \in Q\}, \{\mathsf{init}_q(\vec{x}) : q \in Q\}\rangle$$

where $X \subseteq R_n$ for some $n \in N$, $Q = \{q_1,\ldots,q_m\}$ is a finite set of modes, and the other components are finite sets of quantifier-free LRF-formulas. LRF is a logical language over the real numbers, where a logical formula φ is of the form:

$$\varphi := t(\vec{x}) > 0 \mid t(\vec{x}) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi;$$
$$t := x \mid f(t(\vec{x})),\ \text{where } f \in \mathcal{F} \text{ (constants are 0-ary functions).}$$

dReal translates the hybrid system description *H* and the reachability problem for an unsafe set of states *U* into a *k*-step, *M*-bounded (mode occupation duration) SMT problem Reach$_{H,U}$*(k,M)*. If Reach$_{H,U}$*(k,M)* is unsatisfiable, then the system is guaranteed safe (i.e., avoids U). The LRF - formula Reach$_{H,U}$*(k,M)* is defined as:

$$\exists^X \vec{x}_0 \exists^X \vec{x}_0^t \cdots \exists^X \vec{x}_k \exists^X \vec{x}_k^t \exists^{[0,M]} t_0 \cdots \exists^{[0,M]} t_k.$$
$$\bigvee_{q\in Q} \Big(\mathsf{init}_q(\vec{x}_0) \wedge \mathsf{flow}_q(\vec{x}_0, \vec{x}_0^t, t_0) \wedge \mathsf{enforce}(q,0) \wedge \forall^{[0,t_0]} t \forall^X \vec{x}\ (\mathsf{flow}_q(\vec{x}_0, \vec{x}, t) \rightarrow \mathsf{inv}_q(\vec{x}))\Big)$$
$$\wedge \quad \bigwedge_{i=0}^{k-1} \Big( \bigvee_{q,q'\in Q} \Big(\mathsf{jump}_{q\rightarrow q'}(\vec{x}_i^t, \vec{x}_{i+1}) \wedge \mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}_{i+1}^t, t_{i+1}) \wedge \mathsf{enforce}(q, q', i)$$
$$\wedge \mathsf{enforce}(q', i+1) \wedge \forall^{[0,t_{i+1}]} t \forall^X \vec{x}\ (\mathsf{flow}_{q'}(\vec{x}_{i+1}, \vec{x}, t) \rightarrow \mathsf{inv}_{q'}(\vec{x}))\Big)\Big)$$
$$\wedge \quad \bigvee_{q\in Q} (\mathsf{unsafe}_q(\vec{x}_k^t) \wedge \mathsf{enforce}(q,k)).$$

In this encoding, the quantifier superscripts denote the bounds on the quantified variables. The SMT formula encodes that each controller trajectory starts with some initial state satisfying $init_q(\overrightarrow{x_0})$ for some q. In each step, it follows flow$_q$($\vec{x}_i$, $\vec{x}^t_i$, t) and makes a continuous flow from $\vec{x}_i$ to $\vec{x}^t_i$ after time t. When H makes a jump from mode q′ to q, it resets variables following $jump_{q' \rightarrow q}(\overrightarrow{x_k^t}, \vec{x}_{k+1})$. The auxiliary enforce formulas ensure that picking $jump_{q \rightarrow q'}$ in the *i*th step enforces picking $flow_{q'}$ in the *(i + 1)*st step. The universal quantifier for each continuous flow expresses the requirement that for all the time points between the initial and ending time point (t ∈ [0, t$_i$ + 1]) in a flow, the continuous variables $\vec{x}$ must take values that satisfy the invariant conditions $inv_q(\vec{x})$.

While the dReal tool is much more scalable than algebraic approaches, its complexity is still high. Using its full hybrid reasoning capabilities to synthesize controllers is currently only practical for small systems. For HyCIRCA, we address this problem by leveraging the same abstraction approach that CIRCA uses already: using the CIRCA CSM to quickly synthesize a timed automata controller that works in an approximation of the true hybrid model, and then verifying its correctness using the δ-complete decision procedure. As a result, the high-complexity search processes are conducted in the abstraction space, and most of the time the full hybrid model will simply confirm the abstract results. And, when the dReal verification fails, its counterexamples will help efficiently guide the search towards improved mission plans.

Now that we have the necessary preliminaries, we go on to discuss the new developments in Phase II: our work on (1) hybrid mission planning, (2) controller synthesis and verification, and (3) Signal Temporal Logic encoding for SMT.

## 3.4 Mission Planning

"Classic" AI planning systems have traditionally been limited to finite, discrete state spaces. Later, extensions were made to add numerical parameters (e.g., distance traveled) and time. Currently, the cutting edge of research in this area is to add continuous processes to the discrete models, making *hybrid* planners. In this project, we aimed to extend hybrid planning to more complex systems. In this section, we discuss the HyCIRCA approach to planning for *complex hybrid systems* (CHSs). CHSs are systems that have discrete aspects, as in typical planning systems, but that also involve the control of continuous processes. Unlike much previous work [32][33], we do not limit ourselves to systems with piecewise constant rates of change. Building on the dReal SMT solver [31] [30], we concern ourselves with systems whose continuous dynamics can be characterized by non-linear ordinary differential equations (ODEs).
In previous work in this area, we have built a PDDL+-style "first principles" planner [34]. However, in many cases, we find that working from first principles is prohibitively expensive. Existing CHS planners can address only quite small, artificial planning benchmarks. In our current system, HyCIRCA, we have combined the SHOP2 HTN planner [35] with dReal. Using SHOP2, with its expressive power and search control, we generate a durative action [36] plan that is a tentative witness for a CHS plan, and an SMT problem – a *Flow Temporal Network* (FTN). The FTN is a more accurate formulation of the CHS plan, more accurate in the sense of incorporating a more accurate model of the continuous dynamics. This SMT problem is presented to dReal, which either solves the problem – generating a more accurate CHS plan – or rejects it, causing SHOP2 to backtrack and generate a new plan candidate. To further speed HyCIRCA's problem solving, we preprocess the temporal constraints by applying traditional simple temporal network techniques based upon the Floyd-Warshall algorithm. In addition to efficiency challenges, previous work on CHS planning has often been unsound. Because of the complexity of nonlinear continuous inference, previous systems have not always been able to guarantee the soundness of the plans they generated. We have incorporated Monte Carlo Tree Search (MCTS) into HyCIRCA in order to generate sound witnesses for our plans.

Descriptions of the major contributions of HyCIRCA are described as follows. First, a description of the HyCIRCA hybrid architecture combining SHOP2 and dReal. Second, a formal characterization of the *Flow Temporal Network problem*. Third, a description of the method by which SHOP2 generates FTN SMT problems as a side-effect of planning. Fourth, a description of our MCTS technique for finding sound FTN plans. Fifth, an empirical study and comparison, showing the efficiency gains from the HYCIRCA approach.

HyCIRCA applies SHOP2 to synthesize plans consisting of a totally-ordered sequence of atomic actions (instantiated operators). SHOP2 uses an encoding of durative actions (corresponding to FTN activities), called multi-timeline processing (MTP) [35]. SHOP2 uses both Boolean and numeric state fluents in operator preconditions, add effects, and delete effects. To address more expressive hybrid problems, we adopt a generate-and-test approach that compiles candidate plans into a FTN. We annotate the SHOP2 method and operator schemas with additional constraints that are added to the RFTN if the corresponding methods and operators are used by the plan. These include flow statements $f_{A,Y}(Y)$ for each numeric fluent $Y$ upon which an operator $A$ has a continuous effect, and temporal constraints $T$ relating the start and end times of an activity (as part of the operator definition) and between activities (as part of a method definition).

### 3.4.1 Representation

The HyCIRCA representation involves specifying a SHOP2 planning instance, extended with PDDL2.1 Level 4-like continuous effects, instantaneous numeric effects, and conditions (at-start, at-end, and overall). A SHOP2 instance includes an initial state, a goal task, a set of methods, and a set of operators. We add simple temporal constraints to methods (to relate inter-operator start and end points) and to operators (to relate intra-operator start and end points). To operators, we add at-start, at-end, and overall conditions, in addition to at-start, at-end, and continuous effects. SHOP2 ignores these additions to find a feasible plan. We collect the additions to actions and methods appearing in the plan and use them to transform the SHOP2 plan (which may not be valid) into an FTN. If the FTN is satisfiable, then the satisfying action start times and durations correspond to a PDDL+ plan. In the following, we omit the straightforward translation from an extended SHOP2 plan into a FTN, and focus upon the FTN representation.

A Flow Temporal Network (FTN) is a temporal network that defines:

- $\mathcal{A}$, a set of activities with start and end times,
- $\mathcal{T}$, a set of temporal constraints between times,
- $\mathcal{Y}$, a set of real-valued flow variables,
- $\Delta$, a set of relative change constraints,
- $\Theta$, a set of absolute change constraints,
- $\mathcal{F}$, a set of flow constraints,
- $\mathcal{P}$, a set of time point conditions, and
- $\mathcal{I}$, a set of time interval conditions.

Time Points: The initial time point $X_0$ is used to constrain the initial values of variables and serve as an absolute time reference. Each activity $A \in \mathcal{A}$ defines a start $A^{\vdash}$ and end $A^{\dashv}$ time point. The set of all time points is $\mathcal{X}=\{X_0\} \cup \{A^{\vdash},A^{\dashv} \mid A \in \mathcal{A}\}$ where the domain of each timepoint is the interval $[0,T_{\max}]$. The set $\mathcal{T}$ of temporal constraints consists of constraints of the form: $a \leq X - X' \leq b$ where $a$ and $b$ are constants, and $X,X' \in \mathcal{X}$. $\mathcal{T}$ also includes the constraint $X_0 = 0$.

Flow Effects: Each flow variable $Y \in \mathcal{Y}$ is a real variable whose value is bounded by $[Y_{\min},Y_{\max}]$ over the time interval $[0,T_{\max}]$. We adopt the semantics of mixed discrete and continuous change from the hybrid systems literature [37]. This semantics assumes that discrete change happens with zero "rise time". This means that if a discrete change to $Y$ occurs at time $t$, it holds two values at time $t$: a before and an after value. We denote by $Y(t)$, $Y^{\dashv}(t)$, and $Y^{\vdash}(t)$ the value of Y at, before, and after time t, respectively. In the case of a discrete change at $t$, $Y(t)$ is undefined; and if there is no discrete change, $Y(t) = Y^{\dashv}(t) = Y^{\vdash}(t)$.

Flow variables can change discretely at a time point, and continuously between time points. Discrete change can be relative or absolute. For each $X \in \mathcal{X} \setminus X_0$ and $Y \in \mathcal{Y}$, let $\Delta_X(Y)$ denote the relative change due to timepoint $X$ on variable $Y$, and let $\Delta$ denote this set for all such $X$ and $Y$. For each $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, let $\Theta_X(Y)$ denote the absolute change (i.e., assignment) at timepoint $X$ to resource $Y$, and let $\Theta$ denote this set for all such $X$ and $Y$. For each activity $A \in \mathcal{A}$ and continuous variable $Y \in \mathcal{Y}$, let $f_{A,Y}(Y)$ denote the first order ODE defining the continuous effect of $A$ upon $Y$, and let $\mathcal{F}$ denote this set for all such $A$ and $Y$.

We assume that all change is deterministic. To ensure determinism, no time point may effect simultaneous relative and absolute change upon a flow variable. If the FTN does not specify a relative or absolute change at time point $X$ for flow variable $Y$, we assume that there is a relative change of zero. Similarly, if the FTN does not specify a continuous impact upon $Y$ by an activity $A$, then we assume the impact (rate of change) is zero. By assuming determinism, we ensure that if given an assignment to $Y(X_0)$ for all $Y \in \mathcal{Y}$ and each $X \in \mathcal{X}$, there is a unique value for $Y(t)$, $Y^{\dashv}(t)$, and $Y^{\vdash}(t)$ for all $t \in [0, T_{\max}]$, as appropriate.

Flow Conditions: For each time point $X \in \mathcal{X}$ there is a point condition $f^{X}(Y)$ in nonlinear real arithmetic [38] and we denote by $\mathcal{P}$ the set of all such conditions on timepoints. For all $A \in \mathcal{A}$, there is an interval condition $f^{A\vdash\dashv}(Y)$ in nonlinear real arithmetic that must hold between $A^{\vdash}$ and $A^{\dashv}$, and we denote by $\mathcal{I}$ the set of all such conditions.

Solutions to FTN: A solution to an FTN is an assignment to each timepoint variable $X \in \mathcal{X}$, and initial value of each flow variable $Y(X_0)$ for $Y \in \mathcal{Y}$ such that all constraints in $\Delta$, $\Theta$, $\mathcal{F}$, $\mathcal{T}$, $\mathcal{P}$, and $\mathcal{I}$ are satisfied.

Our formulation of FTN semantics resembles that of the PDDL2.1 Level 4 language (temporal actions with continuous effects) for planning. The main difference is that we assume that there is no ε-separation. PDDL2.1 uses ε-separation to ensure that all discrete change is "written" to the state before it is "read" by separating these by ε time units. We believe that this is an executor implementation issue. Our FTN definition does not prohibit problem designers from using this semantics, instead it requires that it be explicitly encoded it as part of the FTN.

### 3.4.2 FTN to SMT Translation

We encode an FTN in first order logic using nonlinear real arithmetic functions, the language "$\mathcal{L}_{\mathbb{R}\mathcal{F}}$", and check its satisfiability. A satisfying assignment to the variables encodes a solution to the FTN, and an unsatisfiable result proves infeasibility.

One of the primary challenges we face is expressing the simultaneous effects of multiple activities upon a variable. Our solution is to encode what we call a *happening timeline*. The happening timeline is characterized by a totally ordered set of time points. Each FTN timepoint maps to one happening. Multiple FTN timepoints can map to the same happening. Discrete change only occurs at happenings, and continuous change occurs between happenings, in a hybrid automaton. Similarly, point and interval conditions are asserted at and between happenings, respectively. The notion of a happening gives us the ability to aggregate all active activities at and between happenings.

**Table 1 Encoding notation.**

| Var. | Bounds | Description |
|---|---|---|
| $A^{\vdash}$ | $[0, T^{\max}]$ | activity start time |
| $A^{\dashv}$ | $[0, T^{\max}]$ | activity end time |
| $h_i$ | $[0, T^{\max}]$ | time of happening $i$ |
| $\gamma_i^A$ | $[0, 1]$ | $A$ is active between happening $i$ and $i+1$ |
| $Y_i^{\vdash}$ | $[Y^{\min}, Y^{\max}]$ | flow value just after happening $i$ |
| $Y_i^{\dashv}$ | $[Y^{\min}, Y^{\max}]$ | flow value just before happening $i$ |

Table 1 lists the real variables used in our encoding, along with their nominal bounds. In our encoding, we refer to the sets of corresponding variables $\mathcal{A}^{\vdash}$, $\mathcal{A}^{\dashv}$, $h_{0:H}$, $\mathcal{Y}_{0:H}^{\vdash}$, $\mathcal{Y}_{0:H}^{\dashv}$, and $\gamma_{0:H}^{\mathcal{A}}$, where the subscripts denote the variables for all happenings between 0 and *H.* For each set of variables, we use a shorthand notation for bounded existential quantification. We omit the bounds, which are listed in Table 1 for each variable. We also group variables into sets. For example $\exists h_{0:H}$ is equivalent to applying the existential quantifier to each element of $h_{0:H}$, as in $\exists h_0 \ldots \exists h_H$.

**Table 2 FTN encoding in $\mathcal{L}_{\mathbb{R}\mathcal{F}}$.**

$$\exists \mathcal{A}^{\vdash} \exists \mathcal{A}^{\dashv}\, \exists h_{0:H} \exists \mathcal{Y}_{0:H}^{\vdash} \exists \mathcal{Y}_{0:H}^{\dashv} \exists \gamma_{0:H}^{\mathcal{A}}.$$

$$\mathcal{T} \wedge f^{X_0}(\mathcal{Y}_0^{\vdash}) \wedge \left( \bigwedge_{A \in \mathcal{A}} \mathsf{hap}^A \right) \wedge (h_0 = 0) \wedge \bigwedge_{i=0}^{H-1} \left[ (h_i \le h_{i+1}) \wedge \left( \bigwedge_{A \in \mathcal{A}} \mathsf{active}_i^A \wedge \mathsf{condition}_i^A \right) \wedge \left( \bigwedge_{Y \in \mathcal{Y}} \mathsf{var}_i^Y \right) \right]$$

We encode a FTN in $\mathcal{L}_{\mathbb{R}\mathcal{F}}$ as listed in Table 2. The encoding is existentially quantified over the variables listed above. The clauses state that the following must be satisfied (each bullet corresponds to a top-level conjunct in Table 2):

- temporal constraints are met,
- point constraints on the initial values of flow variables are met,
- activity begin and end points coincide with a happening,
- the initial happening is at absolute time zero,
- the happenings are ordered,
- activities that start, are active, or end at a happening impact conditions that must be met and
- changes that occur at that happening (and up until the next happening, in some cases),
- conditions (from the previous bullet) must be met at the appropriate time relative to the happenings, and
- changes in flows, both discrete and continuous, must aggregate the impact of activities either at or between happenings, respectively.

We abuse the notation for the set of FTN temporal constraints $\mathcal{T}$ to also denote the conjunction of its elements in the $\mathcal{L}_{\mathbb{R}\mathcal{F}}$ encoding. The *hap*$^A$ clauses correspond to:

$$\mathsf{hap}^A \equiv \bigvee_{i=0}^{H} \left( (A^{\vdash} = h_i) \wedge \bigvee_{j=i}^{H} (A^{\dashv} = h_i) \right)$$

which ensures that each activity start point maps to a happening, and the end point does not map to an earlier happening. For each happening $h_i$, we use the $\text{active}_i^A$ clause to set the indicator variables $\gamma_i^A$ to denote that $A$ is active between happenings $i$ and $i + 1$:

$$\mathsf{active}_i^A \equiv \big( (A^{\vdash} \leq h_i < A^{\dashv}) \leftrightarrow (\gamma_i^A = 1) \big) \wedge$$
$$\big( \neg(\gamma_i^A = 1) \leftrightarrow (\gamma_i^A = 0) \big)$$

The $\text{condition}_i^A$ clause asserts that activities starting, active, or ending at a happening must have their respective conditions satisfied just prior to the happening (start and end) or between it and the next happening (active):

$$\mathsf{condition}_i^A \equiv \left( (A^{\vdash} = h_i) \rightarrow f^{A^{\vdash}}(\mathcal{Y}_i^{\dashv}) \right) \wedge$$
$$\left( (\gamma_i^A = 1) \rightarrow \forall t_i \in [h_i, h_{i+1}].f^{A^{\vdash\dashv}}(\mathcal{Y}_{t_i}))) \right) \wedge$$
$$\left( (A^{\dashv} = h_i) \rightarrow f^{A^{\dashv}}(\mathcal{Y}_i^{\dashv}) \right)$$

The $\text{var}_i^{YA}$ clause aggregates the discrete ($\text{jump}_i^{YA}$ clause) and continuous ($\text{flow}_i^{Y}$ clause) changes for a particular flow variable $Y$:

$$\mathsf{var}_i^Y \equiv \mathsf{jump}_i^Y \wedge \mathsf{flow}_i^Y$$

$$\mathsf{jump}_i^Y \equiv \left( \bigvee_{\mathcal{X}' \subseteq \mathcal{X}} \left( \bigwedge_{X \in \mathcal{X}'} (X = h_i) \right) \rightarrow \left( Y_i^{\vdash} = Y_i^{\dashv} + \sum_{X \in \mathcal{X}'} \Delta_X(Y) \right) \right) \vee$$
$$\left( \bigwedge_{\Theta_X(Y) \in \Theta} (X = h_i) \rightarrow \big( Y_i^{\vdash} = \Theta_X(\mathcal{Y}_i^{\dashv}) \big) \right)$$

$$\mathsf{flow}_i^Y \equiv \left( Y_{i+1}^{\dashv} = Y_i^{\vdash} + \int_{h_i}^{h_{i+1}} \sum_{A \in \mathcal{A}} \gamma_i^A f_{A,Y}(\mathcal{Y})(t)dt \right)$$

The $\text{jump}_i^{YA}$ clause states that the value of a flow variable after a happening is either the sum of all activities with additive effects upon Y or an assignment. The $\text{flow}_i^{Y}$ clause states how a flow variable will change continuously between happenings $i$ and $i + 1$ by summing the rates of change for all active activities.

### 3.4.3 Monte Carlo Tree Search (MCTS) in dReal SMT

We check the satisfiability of the $\mathcal{L}_{\mathbb{R}\mathcal{F}}$ encoding of the FTN with an extension of the dReal SMT solver. The primary challenge that we address is that dReal can detect δ-satisfiability or unsatisfiability and must be modified to check satisfiability. Checking satisfiability of LRF is undecidable, whereas checking the satisfiability of a δ-relaxation is decidable. Intuitively, a δ-relaxation numerically perturbs an $\mathcal{L}_{\mathbb{R}\mathcal{F}}$ formula by replacing equalities f(x) = 0 by inequalities f(x) ≤ δ. If the original formula is satisfiable it implies the δ-relaxation is satisfiable, but not vice versa. In planning, we seek a satisfying assignment to an $\mathcal{L}_{\mathbb{R}\mathcal{F}}$ formula to ensure that specific action start times and durations guarantee plan correctness. We add a generalization of its constraint search algorithm based upon Monte Carlo Tree Search (MCTS).

dReal: We extend dReal to find satisfying assignments by building upon its approach for Interval Constraint Propagation (ICP), called branch and prune [39]. ICP is a type of constraint processing that represents a feasible interval for each numeric variable (e.g., [-∞, ∞]). We denote by a box, a set of intervals, one for each variable. If the lower and upper bound are equivalent for each variable, the box corresponds to a satisfying assignment. A box is unsatisfiable if at least one variable in a box has an empty interval. Algorithms like branch and prune will select a box and a variable to branch. Branching involves dividing an interval into a number of subintervals (most often two) and creating a new box for each. After branching, applying the constraints to prune the box often results in smaller intervals. Search algorithms prioritize which box to select, which variables to branch, how to branch, and how to prune by propagating constraints.
dReal establishes unsatisfiability ("unsat") and δ-satisfiability by recursively branching and pruning until either all boxes are unsatisfiable, or at least one box is δ-satisfiable. A δ-satisfiable box can include intervals that have non-zero width (i.e., the upper and lower bounds are not equal). It is sometime possible to extract a satisfying assignment from a box by iteratively assigning each variable to a value in its feasible interval. We frame this series of assignments as a random playout.

Our MCTS algorithm for ICP called Branch Prune Playout (BPP) follows the pseudocode in Algorithm 1. BPP is based upon the UCT algorithm [40]. It uses the select method to find a leaf node b of the search tree corresponding to an unexpanded (unbranched) box. It uses the expand method to generate a child b′. If b′ is not unsat (i.e., not empty), then the playout method constructs a box b′′ that is either empty or a point. If it corresponds to a point (i.e., satisfying assignment), then BPP exits with a sat solution. Otherwise, BPP uses the backup method to update the value of each search node on the path from b′ up to b0. We omit detailed pseudocode for many of the methods referenced by Algorithm 1 because they are commonly used in MCTS algorithms and we did not make significant modifications.

The playout method in Algorithm 2 involves assigning each variable i with interval width (denoted |b′′[i]|) greater than zero to a value in the interval. After each assignment, the algorithm uses the prune method on a box, which applies generalized arc consistency. If the box b′′ becomes unsatisfiable, then we update the value for the leaf box b to the average width just prior to becoming unsatisfiable. This value rewards playouts that on average have boxes with large width prior to becoming unsatisfiable.

BPP is a direct generalization of branch and prune that combines a systematic branching strategy with stochastic playouts. As BPP splits boxes, the boxes become smaller. If a problem is

satisfiable, then the branching and playouts can focus on regions that are likely to be satisfiable when other regions are shown unsatisfiable.
The algorithms are presented below:

**Algorithm 1:** Branch-prune-playout

```
repeat
    $b \leftarrow$ select($b_0$);
    $b' \leftarrow$ expand($b$);
    if $\neg$unsat($b'$) then
        $b'' \leftarrow$ playout($b'$);
        if $b''$ is a point then
            return sat;
        end
    end
    backup($b'$);
until unsat($b_0$) or out of time;
return unsat;
```

**Algorithm 2:** playout($b$)

```
$b' \leftarrow b$;
while $b'$ is not a point do
    $b'' \leftarrow$ prune($b'$);
    if $\neg b''.empty()$ then
        Pick dimension $i$ where $|b''[i]| > 0$;
        Uniformly sample $v$ where $v \in b''[i]$;
        Assign $b''[i] = v$;
        $b' \leftarrow b''$;
    else
        $unsat(b'') \leftarrow true$;
        $value(b) = \frac{visits(b)value(b)+|b'|}{visits(b)+1}$;
        return $b''$;
    end
end
$value(b) \leftarrow 1$;
return $b'$;
```

### 3.4.4 Heuristic Improvements

The SHOP2 plans include a network of temporal constraints as a subset of the set of continuous constraints. To speed up solving in dReal, we extracted temporal constraints from the SHOP2 plan and expressed them as a simple temporal network (STN). We then applied a specialized STN constraint propagation algorithm (based on the Floyd-Warshall algorithm) to tighten the constraints. We then encode the *tightened* temporal constraints in the dReal formulation before running the SMT solver. Our experimentation showed promising speed-ups, because solving these STN-based problems – really an STN augmented with continuous constraints from the non-linear dynamics – can be done by simply making a limited number of ordering decisions and then propagating constraints that are implied by these decisions.

## 3.5 Controller Synthesis and Verification

In Phase II, the first issue we worked on in controller synthesis was an automated translation from the plans of the CIRCA CSM to hybrid automata compatible with dReach. This enabled the use of dReach to check correctness of the CSM-generated controllers. When descriptions of continuous processes were available to refine the CSM models (whose process information is limited to durations and did not feature continuous change except in clocks), this permitted more accurate checking of the CIRCA controllers.

### 3.5.1 Translating CIRCA CSM to dReach

SIFT have developed a method for automatically translating CIRCA models to dReach models ('.drh' files). The software module translates CIRCA-generated timed automaton controller models into a form suitable for checking by dReach and dReal. Using this translation, dReach will be able to check the safety of CIRCA controllers with respect to complex hybrid dynamics. In addition to the code to perform the translation itself, SIFT also developed an extension to the CIRCA model notation. This extension allows modelers to incorporate additional continuous process information, to be used by dReach and dReal, in CIRCA model files.

Note that the dReach input format used in this checking relies on an extension to dReach allowing it to accept multiple hybrid automata in its input, instead of only a single automaton, and perform the product composition. That extension was developed by SIFT's Dr. Bryce, on a different project, and is being leveraged for HyCIRCA. Dr. Bryce has worked with Dr. Soonho Kong, one of the dReal maintainers, to get the code for networks of automata into the dReal3 release. This enhancement is a key enabler for our work because: (1) handling networks of hybrid automata enables STL model-checking by enabling the use of techniques based on compiling STL formulas into automata and (2) the models needed to verify CIRCA-generated controllers involve composing automata representing a program in the CIRCA real-time subsystem (which executes the controllers) and the environment (plant).

In August 2016, we drafted a third version of the CIRCA to dReach translation supporting an enhanced version of the hybrid CIRCA modeling language we have called HyCML (hybrid CIRCA modeling language). Based on experiments with earlier iterations of the language and translation, we have replaced the continuous-exit-conds construct in HyCML with hybrid guards and invariants: "continuous-invar" and "continuous-guard." This settles an ambiguity in the interpretation of the HyCML with respect to whether "continuous-exit-conds forces or only enables the transition to the next state or mode of the system.

One challenge to HyCML was the difference between the more rigid PDDL+ event semantics and CIRCA's more flexible semantics, which follow those of the timed automata community. In the PDDL+ community events follow "must" semantics – they occur as soon as their preconditions are satisfied. CIRCA's model follows "may" semantics – events *can* occur when their preconditions are satisfied but *may* not. Although CIRCA's "reliable temporals" follow "must" semantics, they still are not urgent – *i.e.* they are not obligated to occur as soon as possible, only before some upper bound. PDDL+ events, the start or stop of a process, always occur as soon as possible.

In this effort, we extended the SMT encoding for networks of automata to use Boolean variables for the mode of each automata at each step. It was previously encoded with integers. This sped up many of the HSCC benchmarks because more of the consistency checking was done in SAT vs ICP.

We integrated dReach hybrid reachability verification with the CIRCA Controller Synthesis Module (CSM) search. The dReach checks are made as a second verification step following successful verification with the CSM timed automaton verification, in order to minimize the use of the very expensive HA checker. Using the previously developed translator for CIRCA models to dReach models which we used for hybrid verification in a post-processing step (rather than integrated with the search), we created an inner loop translation and verification step. An example of the translation to SMT is given in Appendix 0.

SIFT added *culprit extraction* from dReach reachability models and translation to CSM decisions. Culprit extraction maps elements of the verifier counterexamples to decisions in the search stack. Culprit extraction supports CSM backjumping – fixing verification failures by jumping directly to implicated planning decisions, avoiding expensive and wasteful chronological backtracking.

Building on integrated hybrid verification, we developed code to extract the sequence of planner states and transitions that lead to failure when the dReach verifier determines that failure is reachable. This trajectory through the planner's state space can then be analyzed to identify the set of planner decisions that led to the failure. The planner can then jump to the most recent relevant decision, ignoring any more recent decisions that are not involved in the failure.
To test and demonstrate the advantages of backjumping, we created a "distracted-driver" domain related to our prior driving-downtown domain. In the new variant, after choosing to start driving downtown, the driver chooses to perform some goal-achieving tasks such as making a phone call and listening to a podcast, all while driving downtown. Although these actions achieve independent goals of the agent, they do not contribute to the success or failure of the agent's goal to drive downtown. When the planner discovers that this goal cannot be achieved, it should be able to immediately jump back to that initial decision (start driving) and retract it.

The distracted-driver domain revealed interesting characteristics of some hybrid domains and provoked a new research direction. The interesting twist in this domain is that there is no required order or schedule for the distraction activities. This leads to very unconstrained variables when the SMT searches for a counterexample solution. Furthermore, large counterexamples have many possible, and largely equivalent, assignments to continuous variables. The dReal algorithms for interval constraint propagation (ICP), which search for possible δ-satisfiable boxes, are often slow in such cases, because the variables are under-constrained.

To solve these problems, we developed a generalization of the branch-and-prune ICP algorithm which additionally uses Monte Carlo sampling to assign variables. This technique, which is based upon Monte Carlo Tree Search (MCTS), not only provides satisfiable assignments (not δ-satisfiable boxes) but speeds up counterexample generation when the continuous variables are under-constrained. In contrast with branch-and-prune, our MCTS based ICP algorithm can find satisfying assignments without many costly iterations of branching and pruning. Later, we apply this technique to mission planning, as well (see Sections 3.4.3 and 4.1).

We also made miscellaneous related improvements to the CSM. Integration with dReach counterexamples revealed a bug in the existing heuristic in which it rejected possible paths with "loops" to goals in which a variable value would need to be established more than once. In fact, these are only "loops" over the value of a single variable in a state, not a loop of fully specified system states. A fix we developed, tested, and committed to the CIRCA codebase, accompanied by a correctness test to prevent regression in the future.

### 3.5.2 Extending Goal Verification in CIRCA CSM

Previously, the CSM had a two-phase verification process: in the first phase, controllers were checked for safety (represented as *un*-reachability of a distinguished failure state) and in the second (optional) phase, controllers were checked for goal achievement. This enabled the CSM to exhibit "anytime" behavior: as soon as the first controller was generated and checked, it could be safely employed, but further planning could be used to improve performance (goal achievement).

Note that "goal verification," as opposed to safety verification, is not done incrementally. The CSM runs its timed automaton (TA) verifier (reachability checker) after the synthesis process. However, goal verification is only done when synthesis is complete. This is because if a partial controller design – covering only part of the state space – failed to preserve safety – then any extension of the controller would *also* fail the safety property. However, this form of monotonicity does *not* hold when verifying reachability properties: a partial controller that does not reach a goal state (satisfy a reachability property) may be extended to a full controller that does.

At first, goal verification did not support the counterexample guided revision that safety verification does. Recall that the CSM, when it finds a safety violation, uses the counterexample trace to guide *backjumping* [41]. This was partially because the CSM's verifier did not produce counterexamples with loops (needed to provide counterexamples to reachability goals), and because the code for counterexample-to-backjumping translation, or *culprit extraction*, did not handle counterexamples with loops.

We fixed both of these issues early in Phase II: the CSM's TA verifier has been augmented to find looping counterexamples to reachability goals, and the culprit extraction algorithm has been extended to handle such counterexamples. The culprit extraction method directly carries over to handling counterexamples to reachability goals generated by hybrid verification using dReal.

Improvements still need to be made to the counterexample-guided backjumping process for it to be fully useful. Since safety verification is performed incrementally, any counterexample discovered is guaranteed to contain the most recent synthesis decision. This limiting assumption is not true in the case of goal verification, which is not performed incrementally, but only after a full controller has been synthesized. For this reason, when the CSM simply uses the same backjumping code as before, it may wastefully explore parts of the state space that are not involved in the counterexample. To fix this, we had to rewrite the backjumping code to avoid states that do not appear in the counterexample trace. Examples of the utility of backjumping with goal verification are given in Section 4.2.

In work on STL checking, we investigated work on model-checking Metric Interval Temporal Logic (MITL) properties, since procedures for this task can be generalized to STL (see Section 3.6). Unfortunately, the existing methods for such model-checking involve a very expensive translation from MITL property to automaton. This translation requires the addition of large numbers of new clocks, as a function of the time bounds on the properties [42]). Given that HyCIRCA has tight resource constraints, we argue that the fact that arbitrary MITL checking is expensive and complex, and that specific classes of properties are of greatest relevance, licenses us to choose a useful subset of properties – or rather *property templates* or *classes* that the CSM will be capable of checking on its controllers. We have focused on property classes for a limited subset of Computation Tree Logic (CTL).

This approach was prompted by literature review. We started the table of UAV mission-relevant properties from Humphrey, Wolff, and Topcu [43] (see Figure 9). The properties in this table use the "Until" modal operator, $\mathsf{U}$; "Next," $\bigcirc$ and also the (derived) "Eventually," $\Diamond$; and "Always," $\Box$. We also did a literature review covering related works, including Finucane, *et al.* [44] and Konrad and Cheng [45].

| **Property Name** | **Formula** |
|---|---|
| Safety | $\Box\neg p$ |
| Reachability | $\Diamond p$ |
| Coverage | $\Diamond p \wedge \Diamond q \wedge \Diamond r$ |
| Recurrent Coverage | $\Box\Diamond p \wedge \Box\Diamond q \wedge \Box\Diamond r$ |
| Sequencing | $\Diamond(p \wedge \Diamond(q \wedge \Diamond r))$ |
| Avoidance | $\neg q \,\mathsf{U}\, p$ |
| Avoidance with Reachability | $(\neg q \,\mathsf{U}\, p) \wedge \Diamond q$ |
| Sequencing with Avoidance | $\neg q \,\mathsf{U}\, p \wedge \neg r \,\mathsf{U}\, q \wedge \Diamond r$ |
| Previously | $\bigcirc^{-1} p$ |
| Never After | $\Box(p \rightarrow \bigcirc\Box\neg q)$ |

**Figure 9 Mission goals and properties from Humphrey, *et al.***

We compared the existing goal verification capabilities of the CSM with those in the paper by Humphrey, *et al.,* reaching the following conclusions:

- Safety: *In CIRCA.* Verifiable safety guarantees have been a CIRCA design goal from the very beginning. CIRCA's incremental model checker checks partial plans during search to identify unsafe plans early in the planning process and provide backjumping guidance.
- (*Weak*) Reachability: *In CIRCA.* CIRCA's model checker identifies reachable states in the course of verifying the safety of the plan. Any state encountered by the verifier is

reachable in the sense of the CTL "exists" operator, $\exists$ F g, *i.e.*, there exists some path from the initial state to the goal state.

- (*Strong*) Reachability: *In CIRCA.* For the HyCIRCA project, we have added a goal verification step to the planning process that can decide whether a set of states satisfying a goal feature is "always" reachable in the sense of the CTL operator, $\forall$ F g.
- (*Repeated)* Reachability: *In CIRCA.* The goal verification implemented for the HyCIRCA project can also verify that there always exists a path to the goal, $\forall$ GF g.
- Coverage: *Not in CIRCA.* It will be simple to add this to the CSM by exploding the conjunctions into individual checks.
- Recurrent Coverage: *Not in CIRCA.* As with coverage, the CSM could be relatively easily extended to handle recurrent coverage goals.
- Sequencing: *Not in CIRCA.* Sequencing, avoidance, and avoidance with reachability would require the CSM to support "until" operators, which it does not yet do.
- Avoidance: *Not in CIRCA.*
- Avoidance with Reachability: *Not in CIRCA.*
- Sequencing with Avoidance: *Not in CIRCA.*
- Previously: *Not in CIRCA.* We believe we will not need this. At any rate, "previously" as a temporal modality has odd semantics in a continuous time model.
- Never After: *Not in CIRCA.* This also requires some thought because of the continuous time nature of the CSM models.

We have also implemented maintenance goals which don't correspond directly to the goal types on this list. Maintenance goals are "positive" safety goals ($\forall \Box \varphi$), as opposed to the safety goals which the CSM has always supported ($\forall \Box \neg \varphi$).

While this literature review provided useful guidance, it does not provide *exactly* the set of property classes we need for the CSM. The reason is that mission goals [4] are not appropriate for the controllers of *individual platforms* that are concerned with correct execution of individual parts of missions, and with correct reaction to disturbances. For this reason, we derived from the above a reduced set of the most important property classes for CIRCA-style outermost-loop control of autonomous platforms operating cooperatively.

To better instruct the development of new verifiable goal types for the CIRCA CSM, we have reviewed a previously constructed domain and attempted to extract useful goals and classes of goals. During Phase I of HyCIRCA, we developed a CIRCA domain for a fire-fighting airtanker. For several problems within this domain, the CIRCA CSM can find plans for the airtanker that verifiably satisfy mission goals, such as extinguish-fire (satisfied when a fire has been spotted and sufficient retardant has been dumped upon it) and airtanker-landed (satisfied when the airtanker returns safely to an airport). The domain also sketches out some ideas for spotter plane planning, which suggests goal classes for those cooperative agents, as well.

Recall that the properties needed here differ from the ones developed by Humphrey, *et al.* because the latter were intended to characterize the behavior of the team performing a mission *as a whole,* and that role is played by the mission planner (SHOP2) in the context of HyCIRCA. The controllers whose synthesis we focus on here are for individual agents that must perform their role in the mission plan while responding to disturbances. So, for example, these controllers do not choose which waypoints to go to – they get them from the mission plan – but they provide the behaviors needed for waypoint following, sequencing between waypoints, and reacting to disturbances.

Individual platform goals inspired by our firefighting domain include:

- Return safely to base, an achievement goal (either strong or weak depending on whether loss of an autonomous vehicles is considered catastrophic).
- Extinguish fire, an achievement goal (either strong or weak depending on the urgency of extinguishing the fire relative to possible costs). [Airtanker goal]
- Bounded extinguish fire after notification, a bounded response goal, in which the airtanker verifiably attempts (if weak) or guarantees (strong) to put out a fire within T time units of notification by a spotter. [Airtanker goal]
- Notify and receive acknowledgement, which is satisfied if the spotter receives a message received acknowledgement from an airtanker which will attempt to extinguish a fire. This will be represented as a bounded response goal. [Spotter goal]
- Overfly after retardant dump, which is satisfied if the spotter flies over the fire zone to assess the effects of a retardant dump. This would likely be a weak achievement goal, since other opportunities to assess the effects may be available. [Spotter goal]
- Zone coverage, which is satisfied when the spotter can verify that it will visit each of zone where fires may occur during its mission. [Spotter goal]
- Zone avoidance (or zone deconfliction), which is satisfied if the spotter can guarantee that it will not interfere with the activities of an airtanker in a fire zone. [Spotter goal]
- Zone avoidance with reachability, which is satisfied if the spotter avoids the fire zone when the airtanker is in the zone but can verifiably guarantee in a fire zone that it will eventually reach the zone to perform after action assessment.

Focused by our initial efforts to define a useful set of temporal goal properties, we evaluated the state of the goal implementation in the CSM. Currently, goals are implemented as goal objects, that support methods for verification, which may be either strong (for all paths) or weak (there exists a path), in the style of CTL. The current goal implementation is idiosyncratic and sometimes unintuitive, particularly when there are multiple goals or goals include conjunctions of multiple feature values. For example, if a domain includes multiple, inconsistent single-achievement goals, the planner's heuristic may not discover paths to more than one goal. We have developed a straightforward approach to feature-encoding single-achievement goals which would avoid this difficulty.

Furthermore, the current goal implementation does not directly support disjunctive goals. Once again, we have an approach that would overcome this problem, but it must be implemented as a post-processing step in domain definition.
One interesting wrinkle facing the implementation of new goal structures in the CSM is that CIRCA's domain language does not provide a way to instantly establish a feature value depending which depends on the multiple other feature values. These are often referred to as "rules," "axioms," or "derived predicates" in other planning formalisms. In CIRCA we can

encode these rules as instantaneous transitions, but there is a cost in search efficiency because the planning engine does not understand that these transitions are unavoidable and sometimes futilely attempt to "preempt" a deduction during search. As part of our work on HyCIRCA, we have added an axiom facility to the CIRCA CSM.
We have augmented HyCIRCA's CSM to support the following limited class of properties:

- The goals are in a limited form of CTL, with only a single, outermost $\forall$ or $\exists$ quantifier: *i.e.*, all goals are either "strong," or "weak," respectively.
- We are as yet using only *qualitative* CTL: the temporal modalities do not have time bounds. That means that we cannot have, for example, response goals with deadlines. (We will work to relax this limitation next).
- Because the CIRCA controllers operate in dense time, we do not support the "next" temporal modality.
- Temporal formulas are limited to the following set:

$$\phi \equiv \phi_1 \wedge \phi_2$$
$$\phi \equiv \phi_1 \vee \phi_2$$
$$\phi \equiv \psi \rightarrow \phi_1$$
$$\phi \equiv \Box\phi_1$$
$$\phi \equiv \Box\psi$$
$$\phi \equiv \Diamond\phi_1$$
$$\phi \equiv \Diamond\psi$$
$$\psi \equiv p \wedge \psi$$
$$\psi \equiv p$$

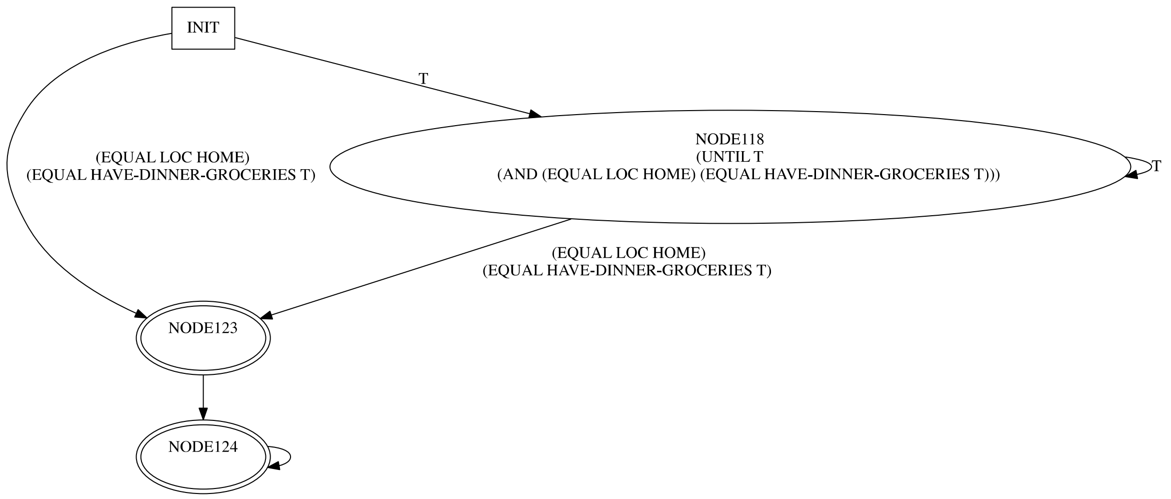
…where the $\phi$ are temporal formulas, $\psi$ are propositional formulas, and $p$ are ground propositions.

As described earlier, the previous implementation of CSM goals (properties) was based on an *ad hoc* set of classes. With the above set of classes in hand, we moved to implement a translation from goal classes to Büchi automata for goal verification. We implemented the LTL to Büchi Automaton translation algorithm of Gerth, *et al.* [46] For illustrations, see Section 0, and the accompanying illustrations (in Appendix 0).

We modified this representation to adapt it to the logic of the CIRCA CSM, which uses feature value assignments (e.g., $location = home$), rather than propositions ($at(home)$). This enables the translation to automata to handle HyCIRCA expressions like:

```
(:finally (:implies (:equal trigger-feature trigger-value)
              (:finally (:equal response-feature response-value))))
```

yielding Büchi automata like the ones in Figure 10 and Figure 11.



INIT
T
(EQUAL LOC HOME)
(EQUAL HAVE-DINNER-GROCERIES T)
NODE118
(UNTIL T
(AND (EQUAL LOC HOME) (EQUAL HAVE-DINNER-GROCERIES T)))
T
(EQUAL LOC HOME)
(EQUAL HAVE-DINNER-GROCERIES T)
NODE123
NODE124

**Figure 10 Büchi automaton for "eventually achieve groceries at home" property.**

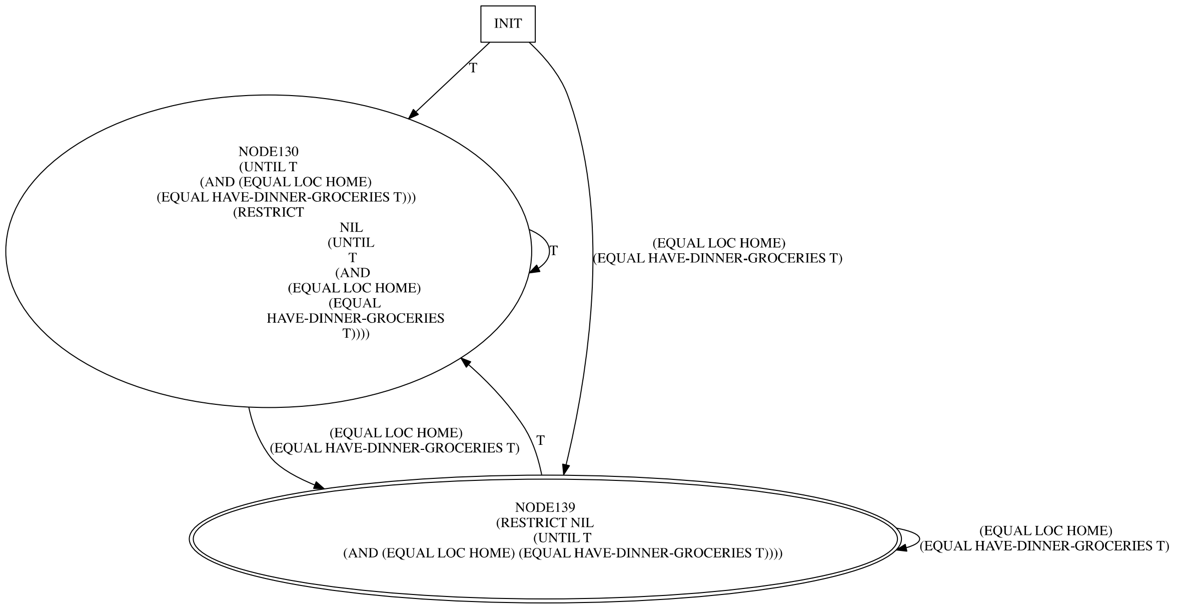


**Figure 11 Automaton for *always* eventually achieving groceries at home.**

The final piece in the puzzle was to integrate Büchi automaton construction with the CSM's CIRCA specific verifier. In a sense, this is simply a straightforward matter of integrating the new goal-checking automaton with the other automata used in verifying (so far only for safety) the CIRCA controllers. In practice, this involved a painstaking construction involving the synthesis of labels on the jumps in the automata in order to properly synchronize the previously-existing automata with the Büchi automata for property checking.

## 3.6 Signal Temporal Logic Encoding

CMU led the development of the fundamental Signal Temporal Logic (STL) model-checking capability. The key idea is to convert the negation of the invariant – the formula to be verified – into a hybrid automaton. Then we reduce the problem to a reachability analysis on a network of hybrid automata, where the network contains only two automata: the automaton for negation of the property, and automaton for the hybrid system to be verified. This second hybrid automaton may itself be the product of multiple sub-automata for different subsystems. We can then perform the reachability analysis on a network of hybrid automata on dReach/dReal, and it follows that the hybrid system satisfies the property, if dReach/dReal returns unsat – that is, if it determines that the combined automata cannot reach their accepting state(s).

The STL Syntax is given in Table 3. $\mathcal{U}_I$ is a timed until, with $I$ a (half-)open or closed interval, and the special interval $[0, \infty)$ being "untimed interval."

**Table 3 STL syntax.**

| |
|---|
| $\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$ |
| Eventually: $\Diamond_I \varphi = \top \mathcal{U}_I \varphi$ |
| Globally: $\Box_I \varphi = \bot\ \mathcal{U}_I \neg\varphi$ |
| $I$ is an interval of the form: $< a, b >$, and $0 \le a < b < \infty$ |

A key research focus is the translation of formulas with nested temporal modalities into hybrid automata. In Phase 1, CMU developed techniques that covered invariants without nested modalities. Complications in translation to automata arise through the combination of nested modalities and time bounds on temporal logic quantifiers. To correctly translate an STL formula into an automaton, we need to enforce the time bound on each mode on the automaton.

SIFT have assisted CMU in formulating STL properties as a network of hybrid automata. Led by Dr. Bryce, we fixed several minor bugs in the network-of-automata specification to SMT translation process in dReach.

We have also worked on verification of STL formula for hybrid systems. For a hybrid system, S, and STL property, φ, this involves two steps:

1. Translating the STL property φ to a hybrid automaton, $A_{\neg\varphi}$
2. Running some bounded reachability analysis on $S \times A_{\neg\varphi}$

The above two steps are mostly orthogonal. SIFT has extended dReach so that it can compute the product of hybrid automata (HA) and perform bounded-time reachability analysis on the product. The CMU team worked on STL translation. Any translation that takes into account the full semantics of STL is challenging, because of some potential difficulties regarding the interpretation of the resulting automata and their compatibility with respect to the interpretation of standard hybrid automata.
We did some literature review on this topic. We found some relevant research papers on translating MITL/MTL logic into timed automata (TA) [47][42][48]. STL is an extension of MTL/MITL. The main difference is that MITL/MTL is expressed over boolean signals, whereas

STL is expressed over real-valued signals. Based on our understanding from literature review, we proposed the following steps to translate STL into timed automata (TA). First, we restrict ourselves in the first phase to one of the following two STL segments:

1. G(φ) segment, where φ is a past STL formula with only Historically*(0,a)*, Once*(0,a)* and Since*(0,a)* temporal operators and
2. Future segment, where we allow only Always*(0,a)*, Eventually*(0,a)* and Until*(0,a)* temporal operators.

The past STL restriction helps by yielding deterministic TA that are easier to follow. In both cases, we suggest restricting temporal operators to have only (0, $a$) bounds because such a restriction implies that only one clock is needed per temporal operator, hence again it will be simpler to understand the correctness of our construction and to debug it. The extension to support all kinds of bounds, such as ($a$, $b$), is purely technical if we have a bounded model checker for formulas that have only (0, $a$), because it can readily be extended to the general case. In late 2016, based on literature research, CMU began a collaboration with Marcello Bersani and colleagues at the *Politecnico di Milano.* Dr. Bersani had done work on more practical translations of MITL to SMT [49], [49], [50]: the translations described earlier, by Nickovic, Maler, and Piterman, were of theoretical interest only as a proof-of-concept. The resulting translations were not efficient enough for practical use. The work by Bersani, *et al.* follows the Bounded Model Checking (BMC) approach to verifying MITL formulas. The key idea is to encode the MITL formula as an SMT formula and solve it in state-of-the-art SMT solver like Z3 or dReal. In their work another language CLTLoc (Clocked, Timed Logic) is used as an intermediary, because it already has a translation algorithm. This process is illustrated in Figure 12.

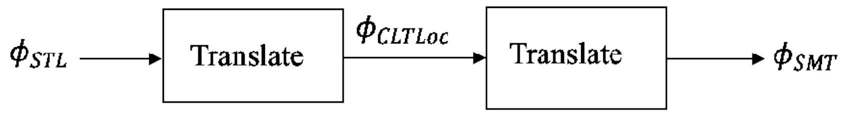


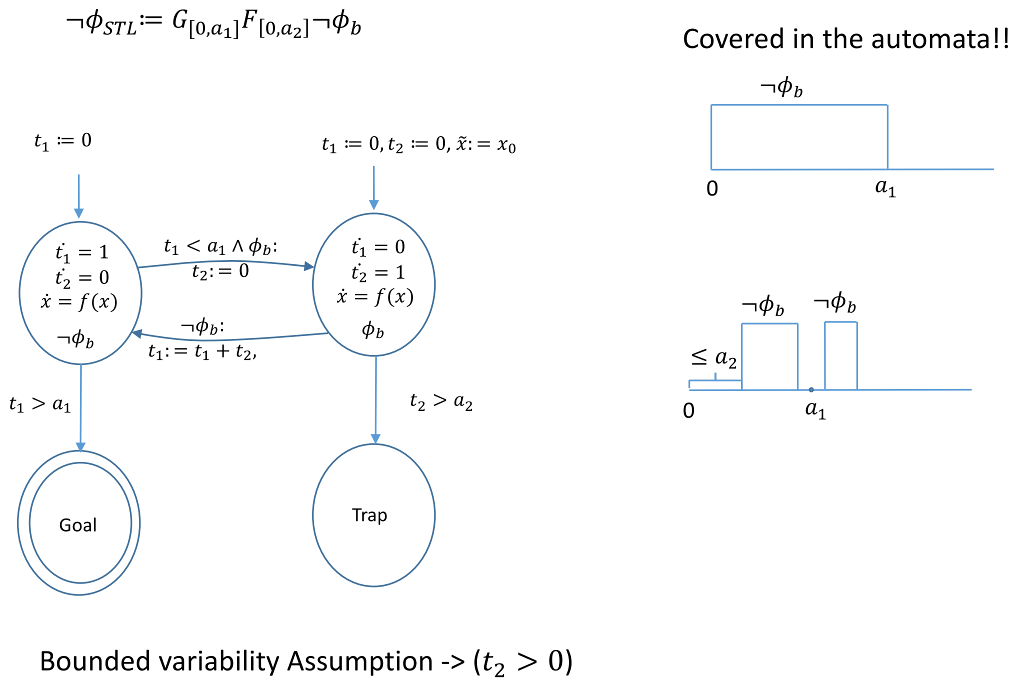
**Figure 12 Two-step translation process, STL to SMT.**

The development process, then, moves to a two-stage process: developing an algorithm for the translation from STL into CLTLoc and then translating CLTLoc to dReal. The previous work by Bersani translated CLTLoc to formulas for Z3 and was not sufficient to handle the complex continuous dynamics in HyCIRCA.

Using this technique, CMU were able to successfully translate an STL formula with *nested* modal operators – $F_{[0,5]}G_{[0,3]}x \leq 0$ – and check it with dReal, against several hybrid automata, under different sets of initial conditions. The encodings developed through the two-phase translation process, however, turned out to yield disappointingly poor performance when checked with dReal. There were a large number of quantified integration expressions that are very difficult to check. As a result, CMU revitalized its earlier approach, based on translating SMT properties into Büchi automata. This yields much tighter translations, but not fully generalized. It is not able to handle arbitrary nested temporal quantifiers, but only formulas such as $FG\varphi$ and $F(\varphi_1 \longrightarrow G\varphi_2)$, and variations that exchange $F$'s and $G$'s. Previously, CMU had shelved the automaton-based approach because of technical issues concerned with unbounded variability. However, the other techniques – including QTSolver, Bersani's tool – all rely on an assumption of bounded variability. So the automaton-based approach is not obviously unsuitable.

Further, even the limited nested quantifications the approach can handle are of pragmatic value. CMU tested this method earlier on properties from the Toyota powertrain example (see Section 4.3), as well as to the Brusselator test problem and some simple UAV problems from SIFT.

CMU developed proofs of correctness for the translation of nested quantifications like $F_{[0,a]}G_{[0,b]}\varphi$ and $G_{[0,a]}F_{[0,b]}\varphi$ into automata. This provides a useful extension of the verification capabilities of dReal. Previously, dReal has offered only safety checking, being able to verify only $G_{[a,b]}\neg\varphi$. We give illustrations in Figure 13, Figure 14, Figure 15, Figure 16, and Figure 17.



$\neg\phi_{STL} := G_{[0,a_1]}F_{[0,a_2]}\neg\phi_b$
Covered in the automata!!
$\neg\phi_b$
0
$a_1$
$t_1 := 0$
$t_1 := 0, t_2 := 0, \tilde{x} := x_0$
$\dot{t_1} = 1$ $\dot{t_2} = 0$ $\dot{x} = f(x)$ $\neg\phi_b$
$t_1 < a_1 \wedge \phi_b$: $t_2 := 0$
$\dot{t_1} = 0$ $\dot{t_2} = 1$ $\dot{x} = f(x)$ $\phi_b$
$\neg\phi_b$: $t_1 := t_1 + t_2,$
$t_1 > a_1$
$t_2 > a_2$
Goal
Trap
$\neg\phi_b$
$\neg\phi_b$
$\leq a_2$
0
$a_1$
Bounded variability Assumption -> $(t_2 > 0)$

**Figure 13 Verifying $\phi_{STL} := F_{[0,a_1]}G_{[0,a_2]}\phi_b$.**

Lemma 3:

$$(\xi(t_1) \nvDash G_{[0,a_2]}\phi_b) \rightarrow (\xi([t_1, t_1 + t_2]) \nvDash G_{[0,a_2]}\phi_b$$

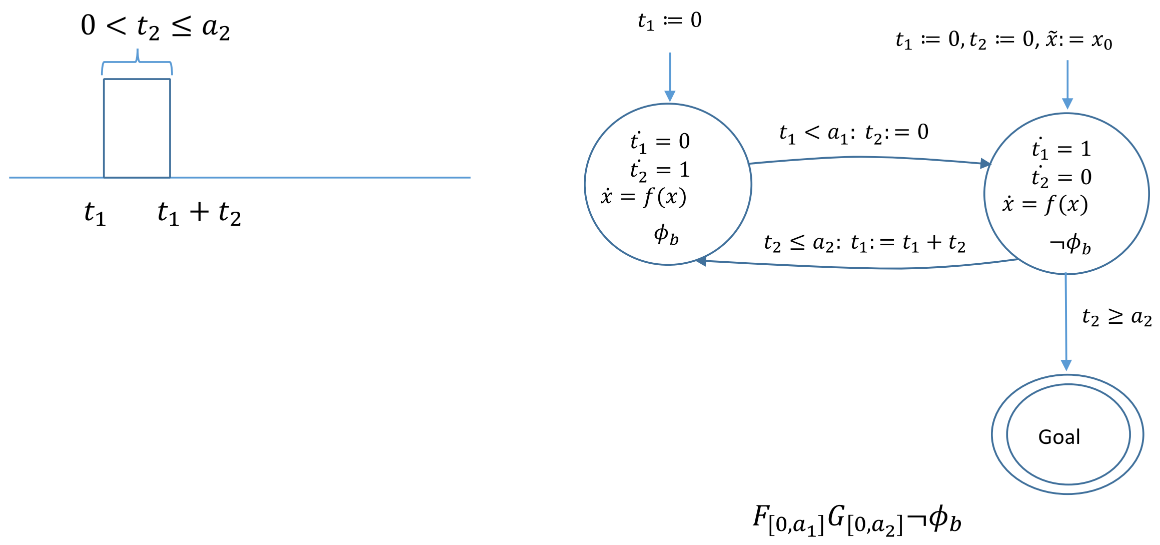
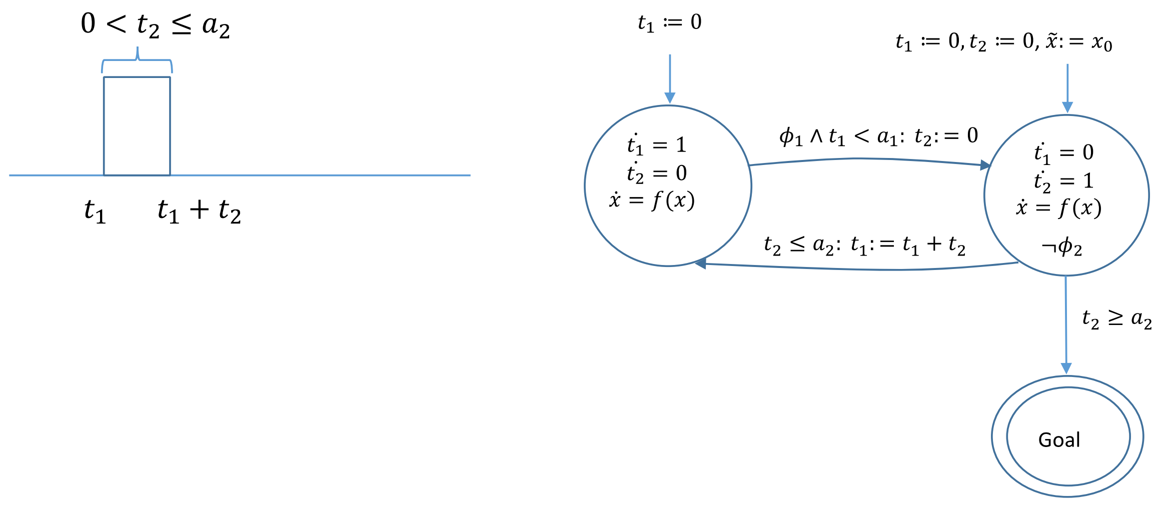


**Figure 14 Verifying $\phi_{STL} \coloneqq F_{[0,a_1]}G_{[0,a_2]}\phi_b$.**

$$\neg\phi_{STL} \coloneqq F_{[0,a_1]}(\phi_1 \wedge G_{[0,a_2]}\neg\phi_2)$$

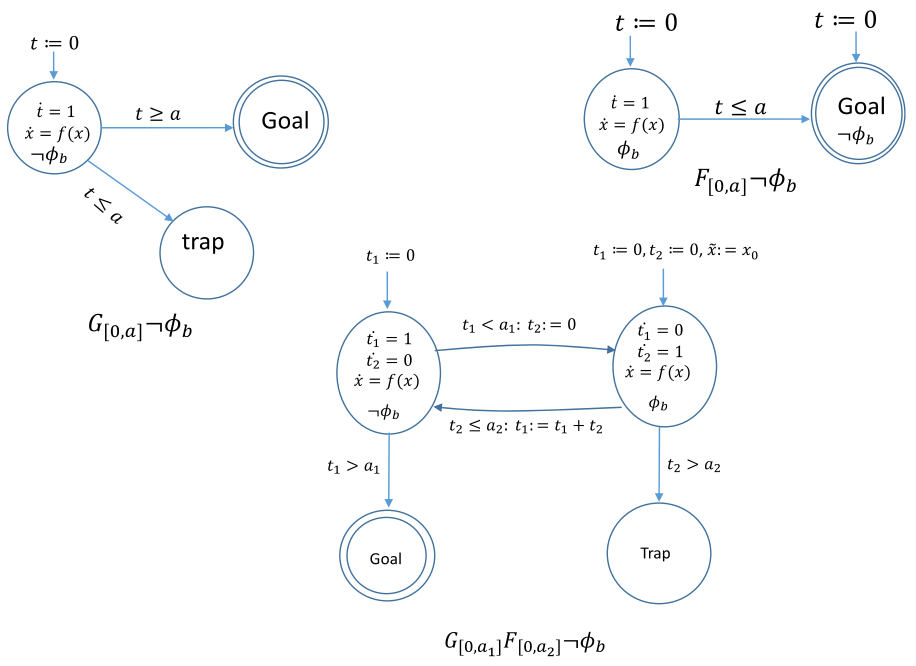
Lemma 3:

$$(\xi(t_1) \nvDash G_{[0,a_2]}\phi_b) \rightarrow (\xi([t_1, t_1 + t_2]) \nvDash G_{[0,a_2]}\phi_b$$



$0 < t_2 \leq a_2$
$t_1$
$t_1 + t_2$
$t_1 \coloneqq 0$
$t_1 \coloneqq 0, t_2 \coloneqq 0, \tilde{x} := x_0$
$\dot{t_1} = 1$ $\dot{t_2} = 0$ $\dot{x} = f(x)$
$\phi_1 \wedge t_1 < a_1$: $t_2 := 0$
$\dot{t_1} = 0$ $\dot{t_2} = 1$ $\dot{x} = f(x)$ $\neg\phi_2$
$t_2 \leq a_2$: $t_1 := t_1 + t_2$
$t_2 \geq a_2$
Goal

**Figure 15 Verifying $\phi_{STL} \coloneqq G_{[0,a_2]} \rightarrow (\phi_1 \rightarrow F_{[0,a_1]}\phi_2)$ (1).**



$t \coloneqq 0$
$\dot{t} = 1$ $\dot{x} = f(x)$ $\neg\phi_b$
$t \geq a$
Goal
$t \leq a$
trap
$G_{[0,a]}\neg\phi_b$
$t \coloneqq 0$
$t \coloneqq 0$
$\dot{t} = 1$ $\dot{x} = f(x)$ $\phi_b$
$t \leq a$
Goal $\neg\phi_b$
$F_{[0,a]}\neg\phi_b$
$t_1 \coloneqq 0$
$t_1 \coloneqq 0, t_2 \coloneqq 0, \tilde{x} := x_0$
$\dot{t_1} = 1$ $\dot{t_2} = 0$ $\dot{x} = f(x)$ $\neg\phi_b$
$t_1 < a_1$: $t_2 := 0$
$\dot{t_1} = 0$ $\dot{t_2} = 1$ $\dot{x} = f(x)$ $\phi_b$
$t_2 \leq a_2$: $t_1 := t_1 + t_2$
$t_1 > a_1$
$t_2 > a_2$
Goal
Trap
$G_{[0,a_1]}F_{[0,a_2]}\neg\phi_b$

**Figure 16 Verifying $\phi_{STL} \coloneqq G_{[0,a_2]} \rightarrow (\phi_1 \rightarrow F_{[0,a_1]}\phi_2)$ (2).**

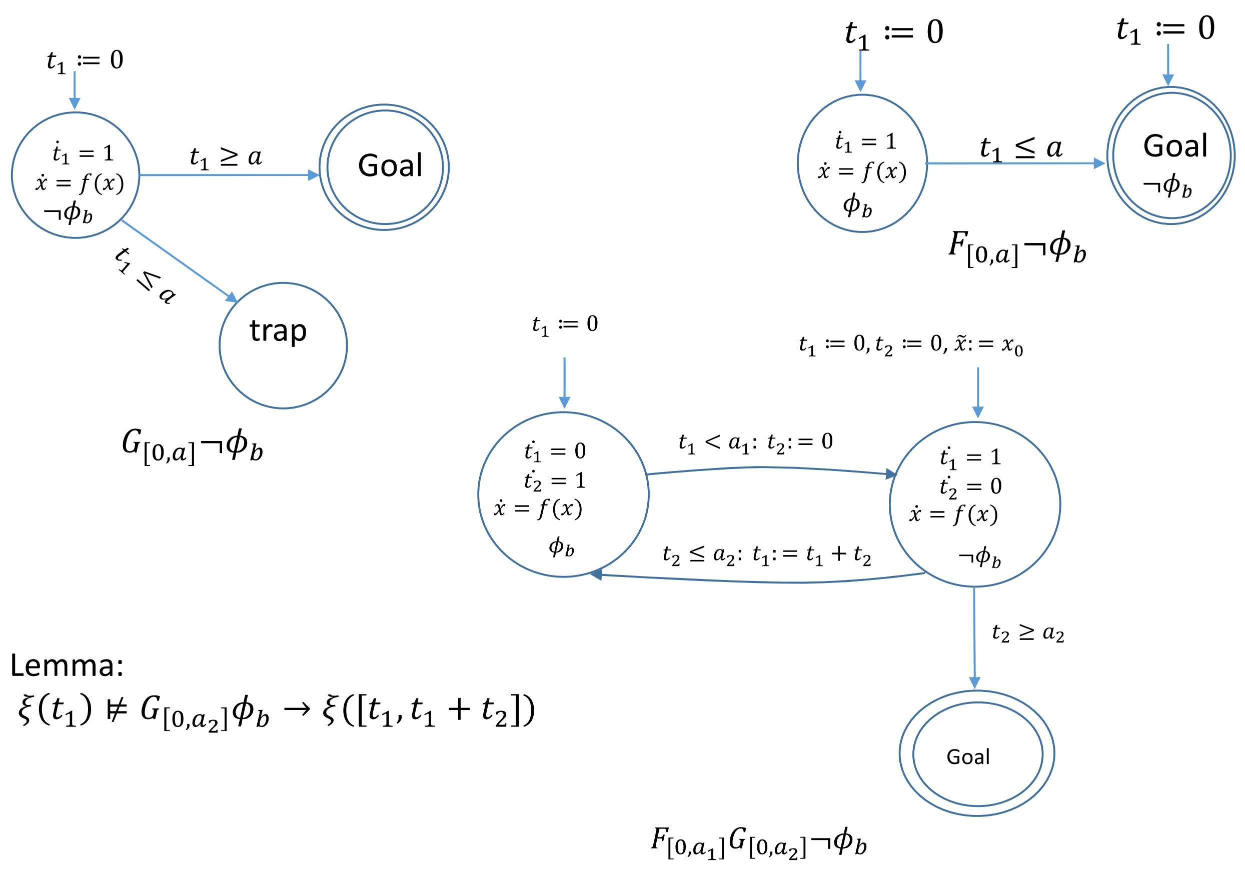


**Figure 17 Verifying $\boldsymbol{\phi_{STL} := G_{[0,a_2]} \to (\phi_1 \to F_{[0,a_1]}\phi_2)}$ (3).**

Experiencing difficulty with checking, because of the problems with integration mentioned above, CMU (with Bersani) worked on eliminating quantification in the translation from STL into CTLoc. Specifically, The STL translation into CLTLoc formulas uses point and interval evaluation of an atomic proposition formula $\varphi_A$:

- $i \models H_{\phi_A}$ : $\phi_A$ holds in left-open and right-open interval $I_i = (t_{i-1}, t_i)$
- $i \models P_{\phi_A}$ : $\phi_A$ holds in point $I_i = [t_{i-1}, t_i]$

where:

$H_\phi$ is equivalent to $\forall t \in (t_i, t_f) \to \phi(t)$ and $\phi(t)$ has form: $g(x(t) \bowtie 0, \bowtie \in \{<, \leq, >, \geq\}$.
The universally quantified formula $H_{\varphi A}$ is the bottleneck of this encoding. Using a technique developed by Cimatti, *et al.*, we recursively rewrite these formulas to eliminate the quantifiers. If *g(x)* is differentiable, and $\dot{g}(x)$ is finitely variable:

$$T(\forall t \in [t_{i-1}, t_i] \to g(x(t) \bowtie 0)) = \begin{cases} g(x(t_{i-1}) \bowtie 0 \wedge g(x(t_i) \bowtie 0)) & \text{if } g(x) \text{ is linear} \\ g(x(t_{i-1}) \bowtie 0 \wedge g(x(t_i) \bowtie 0)) \wedge & \\ T(\text{Const}(\dot{g}, t_{i-1}, t_i)) & \text{otherwise} \end{cases}$$

where:

$$\text{Const}(\dot{g}, t_{i-1}, t_i) = (\forall t \in [t_{i-1}, t_i](\dot{g}(x(t)) \leq 0)) \vee (\forall t \in [t_{i-1}, t_i](g'(x(t)) \geq 0))$$

Consider an example where the system dynamics are $\dot{x} = 2x - 3$ and the STL formula to check is $F_{[0,1]}(x \leq 1)$. Here $g(x) = x - 1$ and $\dot{g} = \nabla g \dot{x} = 2x - 3$. The translation of STL formulas into CLTLoc formulas will contain subformulas like this:

$$H_{(x-1\leq 0)} \Leftrightarrow \forall t \in [t_{i-1}, t_i].(x - 1 \leq 0)$$

An example of the step-by-step recursive translation procedure is as follows:

$\forall t \in [t_{i-1}, t_i](x - 1 \leq 0)$

$$
\begin{aligned}
&= (x(t_{i-1}) - 1) \leq 0) \wedge (x(t_i) - 1) \leq 0) \wedge \\
&\quad ((\forall t \in [t_{i-1}, t_i].2x(t) - 3 \geq 0) \vee (\forall t \in [t_{i-1}, t_i].2x(t) - 3) \leq 0)) \\
&= (x(t_{i-1}) - 1) \leq 0) \wedge \\
&\quad (x(t_i) - 1) \leq 0) \wedge \\
&\quad ((2x(t_{i-1}) - 3 \leq 0) \wedge (2x(t_i) - 3 \leq 0) \vee \\
&\quad (2x(t_{i-1}) - 3 \geq 0) \wedge (2x(t_i) - 3 \geq 0))
\end{aligned}
$$

## 4.0 RESULTS AND DISCUSSION

In this section, we describe the results of experiments and tests of various components of the HyCIRCA system. We proceed from top to bottom. First, we discuss a set of experiments to compare the HyCIRCA approach to planning for hybrid systems with the state of the art in the field. Then we move on to discuss developments in controller synthesis in HyCIRCA. Here we discuss tests of a number of new facilities added to the original CIRCA CSM, including backjumping for goal verification (in addition to safety verification), checking of safety with respect to a complex hybrid model of the plant and disturbances, and generation of controllers that satisfy temporal logic specifications more expressive than CIRCA's original goals of achievement. Finally, we discuss the work done by CMU on extending hybrid system verification to a substantial subset of Signal Temporal Logic (STL).

### 4.1 Mission Planning Evaluation

Our experimental design tests the hypothesis that a two-phase approach to hybrid systems planning can provide substantial savings by pruning the space that needs to be considered by the hybrid SMT solver. To do this, we compare the performance of our HyCIRCA Mission Planner (HMP) against other hybrid systems planners: CoLin and SMTPLAN. We additionally tried to check against the DiNO hybrid systems planner, but we had great difficulty building it, and were never able to replicate the results that the developers reported in their paper. We corresponded with one of the authors (the supervisor of the student who built the system), but he was unable to help us. For this reason, we ended up comparing with only CoLin and SMTPLAN, but used the test domains from all three planners.

We compare the planners' performance on a variety of hybrid planning domains, some previously-existing benchmarks, and some newly created. There is not, as yet, a clearly identified set of test problems for evaluating hybrid systems planners: this is partly due to the fact that existing planners still differ substantially in their capabilities. For this reason, we have adopted benchmarks used in previous hybrid planning papers and added problems that highlight HMP's capabilities. In some cases, we have created families of problems to explore sensitivity of performance to particular parameters of interest. The domains fall into the following set of classes: (1) Benchmark problems from the SMTPLAN distribution: the car problem and a number of variants of the generator problem. (2) Benchmark problems from the DINO distribution: DINO is distributed with the above problems, as well as new solar rover and powered descent domains. (3) Our own benchmark problems from a UAV firefighting domain.

The car domain is the simplest of the domains: it involves getting to a target location, at rest, using step-valued accelerate and decelerate operators. The family of generator problems all involve running a generator for a fixed duration, without running out of fuel, requiring that the plan schedule refueling actions without overflowing the generator's fuel tank. In the nonlinear variant, the rate of filling accelerates over time, instead of being a constant function of time. The Toricelli variant uses a more complex fill rate based on Torricelli's law.

DINO adds the solar rover and power descent domains. DINO's approach to hybrid systems planning is based on discretization of the continuous mechanics, and the solar rover domains are intended to stress the discretization. The nonlinear variant adds a nonlinear function modeling battery charging to the base domain. The powered descent problem also seems aimed to stress discretization – it involves using a planetary lander's thruster to make a gentle landing by controlling acceleration due to gravity while simultaneously managing power (as in the solar rover domain).

Our firefighting domains are quite simple domains that involve following a course through a set of waypoints, while managing fuel usage. For the purposes of this comparison, we have simplified this to a one-dimensional problem; as we will see, even this can be difficult for some planners.

For each planner/problem pair, we have conducted FIXME trials, and report mean runtime and standard deviation. We also test the resulting plans using VAL [51]to verify that they are correct.

**Evaluation Domains** The linear generator problem involves filling a generator with reserve fuel canisters so that it may run for a given period of time. The challenge to the problem instances is that plans may simultaneously fill the generator with all but one canister, lest the generator will overflow. Successful plans reserve at least one canister until the point where the generator will not overflow, and then refuel using the remaining canisters. SHOP2 solves this problem with methods that use all but one canister at the start of the generate action, and the final canister at the end of the plan (ending at the same instant as the generate action). Results are shown in Figure 18.

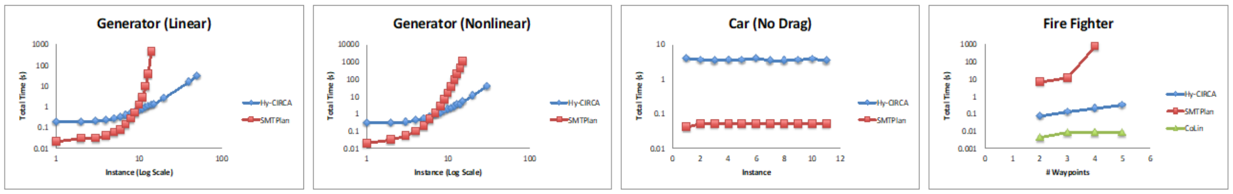


**Figure 18 Comparison results.**

Figure 18 illustrates results comparing HyCIRCA with SMTPlan and CoLin on four planning domains. We compare with SMTPlan on all problem domains, but only CoLin in the Fire Fighter domain. CoLin can only address linear change in our simplified Fire Fighter domain. The figures report the total time in seconds to generate a plan for increasingly challenging problem instances. The Generator (linear and nonlinear) and Fire Fighter instances require plans with more actions as the problems increase in scale. The Car instances require the same number of actions per instance but increase the number of available actions. The Fire Fighter domain increases the number of waypoints an air asset must fly to extinguish a fire, while tracking its linearly changing distance from the starting point.

The Generator instances illustrate that HyCIRCA has a small overhead associated with SHOP2 search, SMT translation, and dReal solving that is not justified in smaller instances. As the instances become larger, the overhead associated with applying HTN planning to select actions is well justified. HyCIRCA outperforms SMTPlan because SMTPlan couples its search over action choice and numeric reasoning, where HyCIRCA decouples the two. The Car instances illustrate that the overhead associated with HyCIRCA is not a significant factor even in smaller instances, unlike Generator. The Fire Fighter domain illustrates how HyCIRCA is competitive with CoLin, a purpose-built planner for linear numeric change, and is significantly better than SMTPlan. We believe the primary difficulty in the Fire Fighter domain is how the durative actions associated with flying between waypoints have a *computed* duration. Planners such as SMTPlan that must couple action selection with temporal and resource reasoning may perform more search. Decoupling action selection from action scheduling, in this case, provides a more effective organization of the search space.

The overall theme we observe in these experiments is that separating action selection from temporal and resource reasoning can have a dramatic impact on scalability. Furthermore, structuring action selection search with an HTN planner that applies domain knowledge can be particularly helpful.

## 4.2 CSM Improvements

### 4.2.1 Hybrid model-checking

As described above, HyCIRCA would initially check its controllers for safety, using its original, cheaper, timed automaton model-checker. If a controller was found to be safe, in the timed abstraction of the full hybrid system, the hybrid system would then be translated into a hybrid automaton so that dReach and dReal could check it for correctness using the richer model, featuring continuous nonlinear processes.
An example from the UAV firefighting domain is the “low fuel light” scenario. This checks whether a controller that will abort its mission when it receives a low fuel warning can operate safely (i.e., never running out of fuel and falling out of the sky). CIRCA uses a simple, temporal abstraction of the problem, in which the abstraction is made that the UAV will have some amount of flight *time* left after the low fuel indicator. The hybrid model is more accurate, representing the fact that the low fuel indicator actually indicates a fixed amount of fuel remains, and that whether or not the UAV lands safely is a function of distance, fuel consumption, and fuel remaining.

The model is captured in a *lifted* form in HyML (the HyCIRCA Markup Langauge). Some example components follow. First, the controller cannot predict when the low fuel warning will come on (a discrete controller, it does not monitor actual fuel level):

```
(def-event low-fuel-warning-turns-on
  :documentation "Low fuel warning turns on."
  :continuous-invar "(fuel >= LOW_FUEL)"
  :preconds ((low-fuel-warning f)
             (landed f))
  :postconds ((low-fuel-warning t)))
```

The temporal abstraction is captured by the following failure process:
```
(def-temporal failure-no-reaction-to-low-fuel
  :documentation "CIRCA fails if it doesn't start emergency return
                  w/in 20 ticks of low-fuel-warning."
  :continuous-invar "(fuel >= 0)"
  :continuous-guard "(fuel <= 0)"
  :preconds ((low-fuel-warning t)
             (flight-mode nominal)
         (landed f))
  :postconds ((failure t))
  :min-delay 20)
```

Note that the minimum delay component captures the temporal abstraction, but the continuous invariant and guard provide a more accurate model based on fuel level.
Finally, we have an expression which captures a simple model of fuel flow:

```
(def-drh-text define-fuel-flows
  :text "#define FLIGHT_FLOWS(on) \\
  d/dt[velx] = 0; \\
  d/dt[vely] = 0; \\
  d/dt[posx] = velx; \\
  d/dt[posy] = vely; \\
  d/dt[fuel] = - FUEL_ECO * (sqrt(velx^2+vely^2) + 0.1 * on);"
)
```

If the CSM can successfully construct a controller, and check it using the temporal abstraction, the information in the HyML domain description is used to compose an SMT model suitable for dReach. An example of this model is given in Appendix B.

### 4.2.2 Heuristic improvements

As we worked to extend the CIRCA CSM, we addressed limitations in the CSM's overall search heuristic, which guides the search for a safe (initially) and goal-achieving (later) controller. We tested these improvements on the "windy grid" domain.

The windy-grid domain involves a simple moving agent trying to reach a goal location in a grid by turning in a particular direction and then moving forward, while also subject to "wind" in each grid location that may push the agent towards falling off the edge (failure). Because the wind and the agent's actions race against each other, the agent can sometimes actually perform a movement action in a state for which it was not intended (a race condition, hazard, or "ghost" action). The multi-model verifier's job is to ensure that failure states are not reachable, even in the presence of race conditions; if it is reachable, the planner backtracks to make a different choice.

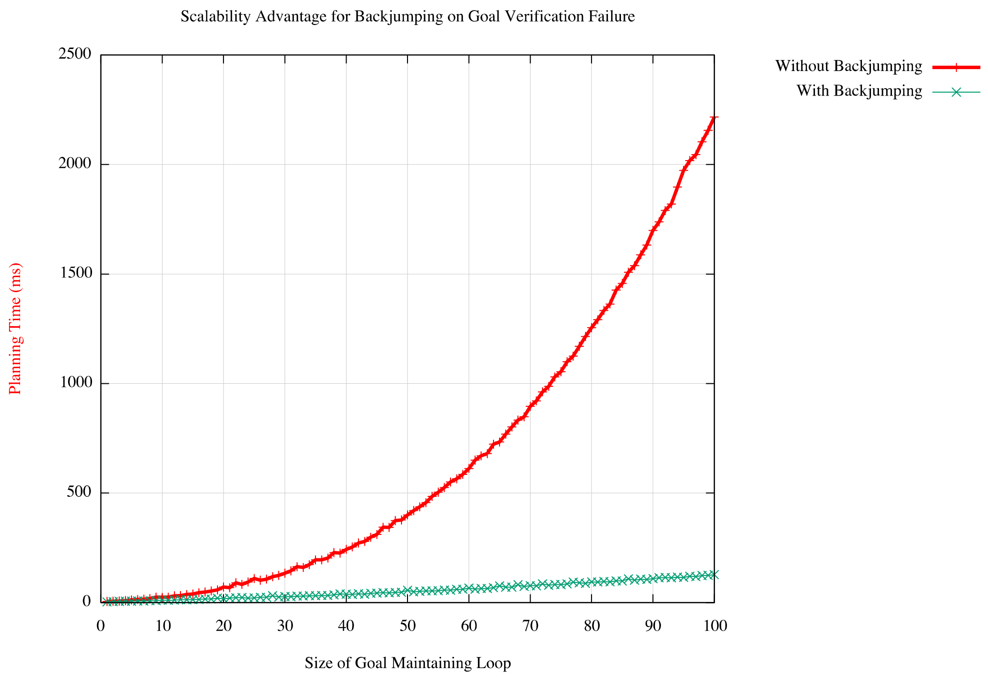


**Figure 19 Run time for "trap" domain with and without backjumping.**

The heuristic improvements included making the heuristic better at anticipating race conditions that might lead to immediate failure. Earlier versions of the heuristic were also not correctly accounting for path-dependent costs and could yield poor decisions that caused a lot of backtracking in these problems. We achieved dramatic performance improvements on windy-grid and other domains, with no degradation in the dozens of other CIRCA test domains. With improvements to the path-cost tracking, the control- synthesizer (planner) now solves all scales of the windy-grid domain with only a single backtrack (or none at all, if the domain is encoded without conditional postconditions). For example, the solving time for the 9x9 windy-grid domain with conditional postconditions improved from 96s to 30s, and without conditional postconditions it dropped from 27s to just 3.4s.

### 4.2.3 Backjumping

When it uses counterexample guided backjumping, the CSM can skip over many planning decisions that are irrelevant to a failed goal verification (i.e., one that found a counterexample to the desired property). If the relevant decision occurred early in the planning process, backjumping can save significant time compared to chronological backtracking. To illustrate the potential of this technique, we created a demonstration domain in which the planner may make bad decisions early in the planning process.

As shown in Figure 19, without backjumping planning time increases super-linearly with the size of a goal-maintaining loop of states, but with backjumping enabled, planning time increases by a small linear factor of the domain size.

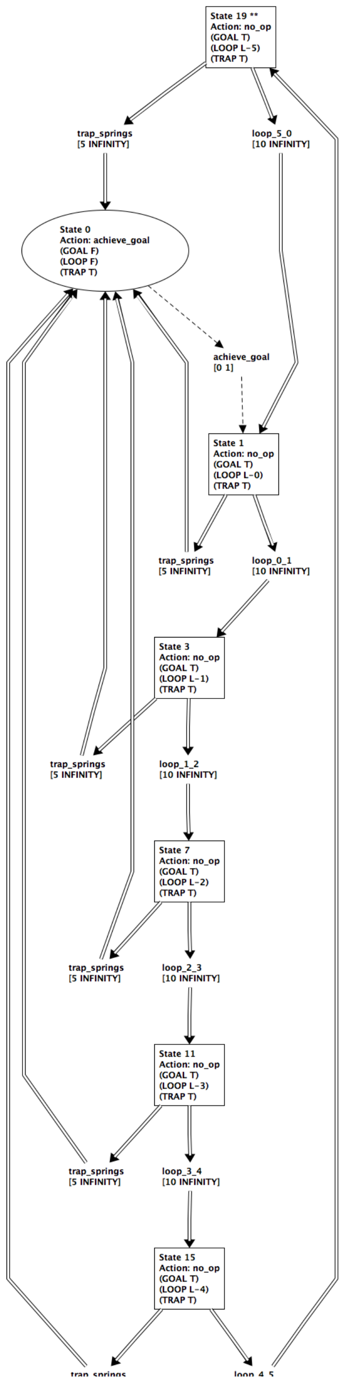


**Figure 21 Safe controller that fails to achieve goal.**

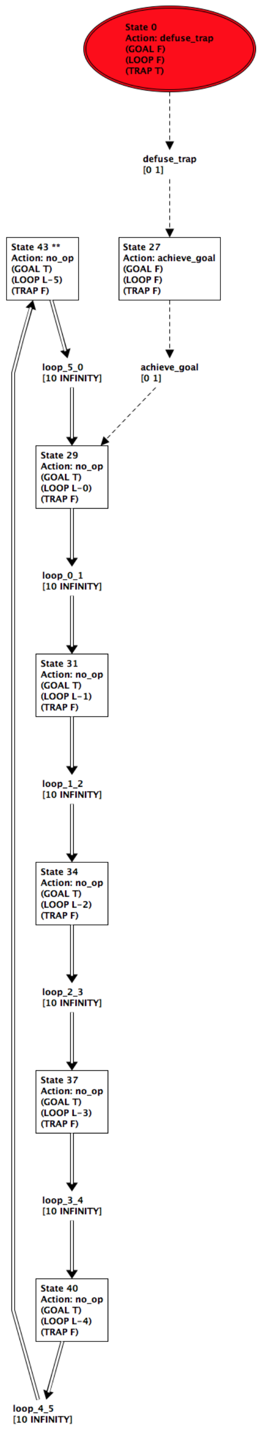


**Figure 20 A safe controller that *does* achieve goal.**

### 4.2.4 Goal Verification

Figure 21 and Figure 20 show the effects of goal verification. Figure 21 shows the initially synthesized, safe controller that fails to achieve the goal. The goal here is to achieve and maintain GOAL = True. Note that the controller is guaranteed to safely avoid having the system be destroyed by the trap, after it springs.

This scenario demonstrates the difference between strong and weak achievement goals and the benefit of culprit-directed backtracking for goal achievement. A “strong” achievement goal in traditional CIRCA terminology is equivalent to an “eventually always” goal in LTL terms. Similarly, a “weak” goal corresponds to an “eventually” goal (which may be satisfied if the goal condition is satisfied for only a finite period). Before the HyCIRCA project CIRCA’s CSM did not correctly verify “strong” achievement goals or provide counter-examples to direct backtracking for goal verification. The first graph shows the first plan that the CSM’s will produce. Its search heuristic eagerly achieves the goal from the initial state, but leaves a “trap” in place that will eventually defeat the goal conditions. (The uncontrollable trap has been constructed so that it cannot be rectified once the goal has been achieved, a bit like leaving your office key at home before driving to work.)

This plan is verifiably safe and does contain a reachable state that satisfies the goal condition so it is accepted by the first phase of plan generation. However, the goal verifier recognizes that the “always” requirement is not met. It also produces a counter example that the planner can use to backtrack over the numerous loops states to the initial state in which the planner made the wrong action choice. After backtracking, the CSM produces the plan shown in the second figure, in which it uses the “defuse_trap” action to defeat the long-running trap transition before it achieves the goal conditions.

As described in Section 3.5.2, we implemented a translation from our limited set of goals to Büchi Automata. The resulting automata are given as a series of figures at the end of the report, see Appendix Figure 29 to Figure 39.

**Property Checking** Here we give an example of the integration of property automata with CIRCA’s controller verification. First, Figure 23 shows a candidate controller, synthesized by CIRCA’s CSM. Recall that the CSM generates memoryless and un-clocked controllers. This is for a very simple controller whose objective is to bring groceries home but must deal with a mischievous child who can steal and eat the groceries. However, once the groceries are delivered to the home, the system’s goal is satisfied (even if the kid then eats the groceries). Note that this figure shows the addition of a synthetic feature, `HAVE-DINNER-GROCERIES T LOC HOME` for the conjunction “`HAVE-DINNER-GROCERIES` and `LOC HOME`”, which is added to the state space, and managed by an axiom.
To verify this controller, the CSM’s model checker composes the multiple models shown in Figure 22 to Figure 25.
Figure 22 shows the evolution of the system’s state, which is affected by controlled and uncontrolled (disturbance) actions. Figure 23 shows the CIRCA-generated controller, which senses the world and, based on sensory inputs, asynchronously interacts with the environment by triggering controlled processes. Figure 24 is a model for a single uncontrolled/disturbance action, in this case a child who, at arbitrary times, may steal groceries and eat them. In general, there will be many such disturbance models. Figure 25 is the automaton that captures the simple achievement goal of reaching home with

groceries (not stolen by the kid). Finally, Figure 26 is the product automaton, assembled by combining the previous sub-models.
These models show the functioning of the world, as affected by the controlled and uncontrolled processes; the controller; an example uncontrolled process; and the automaton that captures the goal. The CSM's verifier computes the product automaton (on the fly), shown in Figure 26 and, in this case, determines that there exists a path that leads to the solution. The transition from `RTA-State 11` to `RTA-State 12` is the end of a path to a goal-achieving state. The location vector `Location (6 1 1 2 2)` – where each element in the vector is the index of a state in one of the component automata – represents the goal satisfying state by the final "2" in the vector which corresponds to the goal-satisfying state in the goal automaton.

SEE ATTACHMENT FIGURE 22

**Figure 22 State model.**

SEE ATTACHMENT FIGURE 23

**Figure 23 Controller model.**

SEE ATTACHMENT FIGURE 24

**Figure 24 Uncontrolled process model.**

SEE ATTACHMENT FIGURE 25

**Figure 25 Automaton for simple achievement goal.**

SEE ATTACHMENT FIGURE 26

**Figure 26 Product automaton, constructed on the fly for verification.**

## 4.3 STL Verification

CMU worked on a benchmark problem from the powertrain controller domain [52]. Specifically, they worked to verify the throttle component of the controller. The throttle generates an input signal for the controller (see **Figure 27**). This input signal needs to satisfy a specific input profile that can be specified as an STL formula:

$$\begin{aligned} \text{rise}(a) &\equiv (\theta = 8.8^{\circ}) \wedge \Diamond_{(0,\epsilon)}(\theta = a) \\ \text{fall}(a) &\equiv (\theta = a) \wedge \Diamond_{(0,\epsilon)}(\theta = 8.8^{\circ}) \end{aligned}$$

The main challenge is to convert an STL formula to a corresponding timed hybrid automaton (hybrid automaton with time bounds for dwell in states). In our current approach, we first convert the formula to a Büchi automaton (without any time bound). Next, we convert the Büchi automaton as a timed automaton by adding clock variables for each temporal operator. The main challenge is to compute the correct invariant set for each mode of the timed automaton. As the formula for rise and fall is simple, we can, however, compute the invariant set in these cases CMU extended their coverage to more of the powertrain controller, specifically to the requirement constraint of normal operating mode (See Figure 28):

> In normal operating mode of the powertrain model, the controller uses both feedback PI control and feed-forward control to regulate the A/F ratio. Assume at time t, $lambda(t)$ is the A/F ratio and $\lambda_{ref}$ is the reference A/F ratio that the controller needs to maintain. For convenience, we define the normalized error signal, $\mu$, as: $\mu(t) = \frac{\lambda(t)-\lambda_{ref}}{\lambda_{ref}}$.

Given the above definitions, the normal operating mode requirement can be expressed as STL formulas as follows:

$$\square_{(\tau_s,T)}|\mu| < 0.05$$

$$\square_{(\tau_s,T)}\Big(\text{rise}(a)|\,\text{fall}(a) \Rightarrow \square_{(\nu,\frac{\zeta}{2})}|\mu| < 0.02\Big)$$

The main challenge in verifying such requirements is the complexity of the formula, in particular the nested temporal operators. We have not automated the conversion of such formulas to automata, since, to the best of our knowledge, no such algorithm exists.

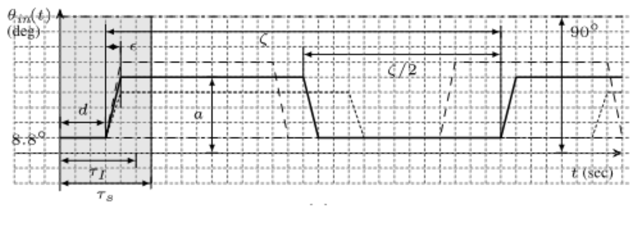


**Figure 27 Plot of throttle input signal.**

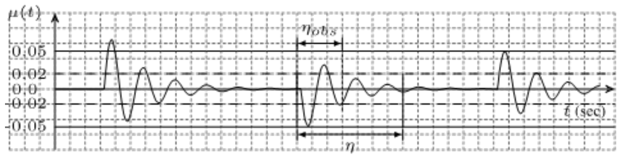


**Figure 28 Powertrain operating mode behavior.**

The CMU team also verified the stability property of the Brusselator[53], [54] model. This model is an example of an autocatalytic, oscillating chemical reaction. The dynamics of the model is defined as:

$$\dot{x_1} = f_1(x) = 1 - (b+1)x_1 + ax_1^2x_2$$
$$\dot{x_2} = f_2(x) = bx_1 - ax_1^2x_2$$

where $x = [x_1, x_2]$ are state vectors, $f = [f_1, f_2]$ is the vector field and *a* and *b,* are model parameters with *a, b> 0.*

The equilibrium state or fixed point ($x_{eq}$) of a dynamical system is defined as: $f(x_{eq}) = 0$.
The local asymptotic stability of a system around its fixed point $x_{eq}$ is defined as:
$If\ for\ any\ \delta > 0, \quad such\ that\ |x(0) - x_{eq}| < \delta \rightarrow \lim_{t\rightarrow\infty}(x(t) - x_{eq}) \rightarrow 0$
We encoded this stability property as an STL formula as follows:
$G_{[0,a_1]}F_{[0,a_2]}\left(|x(0) - x_{eq}| < \delta \rightarrow |(x(t) - xeq))| < \epsilon\right)$.

Instead of zero, we consider a small positive number $\epsilon$, as we are using bounded-time encoding. We then verify this property on dReach/dReal by combining the automaton corresponding to the negation of the formula with the Brusselator model. Similarly, we also encoded local Lyapunov stability as : $F_{[0,a_1]}G_{[0,a_2]}\left(|x(0) - x_{eq}| < \delta \rightarrow |(x(t) - xeq))| < \epsilon\right)$ and then verified it on dReal/dReach.

## 5.0 CONCLUSIONS

In this report, we have described SIFT and CMU's work on the Phase II HyCIRCA STTR. In particular, we have reviewed our contributions in the area of hybrid planning; controller synthesis; and model-checking verification for complex hybrid systems with non-linear continuous dynamics. The key contributions of this work are the following: (I) new methods for hybrid mission planning based on integrating SHOP2's HTN planning with dReal's SMT solving to handle complex dynamics; (II) the addition of complex temporal logic properties as constraints on CIRCA's controller synthesis; (III) multiple abstraction level modeling for controller synthesis with three levels of abstraction: (1) unclocked reactive discrete control; (2) timed automata for hard real time constraints; and (3) hybrid automata for correct supervisory control of complex cyber-physical systems; (IV) More expressive model-checking for hybrid systems based on translations from Signal Temporal Logic (STL) to SMT.

The HyCIRCA Phase II opens a number of areas for further investigation. Our work in hybrid planning, integrating the SHOP2 HTN planner with dReal suggests that carefully staging problem solving can provide substantial improvements in efficiency. It also shows the limitations of existing SMT solvers for nonlinear systems: taking advantage of specific features of our planning problems was critical to achieving acceptable performance. Also while our approach to generating an abstract mission plan using HTN, and then performing parameter synthesis with dReal provided impressive speed-up, there are challenges if the abstract plan does not have the *downward refinement property* – i.e., when the SHOP2 plan is *not* a conservative abstraction and may not have a sound refinement. Note that our work in controller synthesis does not have this problem – if a controller does not have an extension, we have mechanisms to efficiently repair it – our counterexample guided backjumping.

The controller synthesis work in HyCIRCA likewise raises a number of interesting questions. One is, what is the right relationship between the abstract and base models? Should the abstract model be conservative, coming as close as possible to the downward refinement property, so that we avoid the time cost of checking controllers that are unsound? The cost of such a policy is that we may lose many controllers that are sound, but that *look* unsound in the abstract representation. Another question is whether we can perform parameter synthesis for hybrid controllers in the CIRCA framework, making them more powerful than the purely discrete, unclocked controllers that CIRCA currently generates.

Finally, there are a large number of questions about STL verification for hybrid systems. Key questions include what is the boundary between practical and intractable? Are there subsets of STL that are effectively checkable, even if not actually tractable? What limitations need to be made for STL verification to be practicable, and are there interesting controllers in this space? Finally, we considered at one time whether we could use controllability theory to find parts of the control space that are provably controllable, even if we cannot generally verify correct control behaviors.

## 6.0 REFERENCES

[1] D. J. Musliner, “CIRCA: The Cooperative Intelligent Real-Time Control Architecture,” University of Michigan, Ann Arbor, 1993.

[2] D. J. Musliner, E. H. Durfee, and K. G. Shin, “World Modeling for the Dynamic Construction of Real-Time Control Plans,” *Artif. Intell.*, vol. 74, no. 1, pp. 83–127, Mar. 1995.

[3] R. P. Goldman, D. J. Musliner, and M. J. S. Pelican, “Exploiting Implicit Representations in Timed Automaton Verification for Controller Synthesis,” 03, pp. 225–238.

[4] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul, “The Challenges of Real-Time AI,” *IEEE Comput.*, vol. 28, no. 1, pp. 58–66, Jan. 1995.

[5] P. E. Agre and D. Chapman, “Pengi: An Implementation of a Theory of Activity,” in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, Seattle, Washington, 1987, pp. 268–272.

[6] D. J. Musliner, “Using Abstraction and Nondeterminism to Plan Reaction Loops,” 1994, pp. 1036–1041.

[7] D. J. Musliner, R. P. Goldman, and M. J. Pelican, “Using Model Checking to Guarantee Safety in Automatically-Synthesized Real-Time Controllers,” in *ICRA*, 2000.

[8] S. Yovine, “Kronos: A Verification Tool for Real-Time Systems,” *Springer Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1/2, Oct. 1997.

[9] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, “The tool Kronos,” in *Hybrid Systems III: Verification and Control*, 1996, pp. 208–219.

[10] Drew McDermott, “Using Regression-match graphs to control search in planning,” *Artif. Intell.*, vol. 109, no. 1-- 2, pp. 111–159, Apr. 1999.

[11] R. P. Goldman, D. J. Musliner, K. D. Krebsbach, and M. S. Boddy, “Dynamic Abstraction Planning,” 1997, pp. 680–686.

[12] E. M. Clarke *et al.*, “Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems.,” *Int J Found Comput Sci*, vol. 14, no. 4, pp. 583–604, 2003.

[13] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[14] S. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala, “The MARUTI Hard Real-time Operating System,” *SIGOPS Oper Syst Rev*, vol. 23, no. 3, pp. 90–105, Jul. 1989.

[15] D. J. Musliner, “Scheduling Issues Arising from Automated Real-Time Scheduling,” University of Maryland Department of Computer Science, CS-TR-3364, UMIACS-TR-94-118, Oct. 1994.

[16] R. P. Goldman, D. J. Musliner, and K. D. Krebsbach, “Managing Online Self-Adaptation in Real-Time Environments,” in *Self-Adaptive Software: Applications*, pp. 6–23.

[17] D. J. Musliner, “Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis,” in *Lecture Notes in Computer Science*, SV, 2001.

[18] R. P. Goldman, D. J. Musliner, and K. D. Krebsbach, “Deliberation Scheduling for Hazardous Missions,” in *submitted to AAAI’02*, 2002.

[19] R. G. Smith, “The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver,” *TCOMPUTERS*, vol. C–29, no. 12, pp. 1104–1113, 1980.

[20] E. Atkins, E. H. Durfee, and K. G. Shin, “Plan Development Using Local Probabilistic Models,” pp. 49–56.

[21] E. M. Atkins, E. H. Durfee, and K. G. Shin, “Detecting and Reacting to Unplanned-for World States,” pp. 571–576.

[22] H. L. Younes and D. J. Musliner, “Probabilistic Plan Verification through Acceptance Sampling,” in *Proc. AIPS-02 Workshop on Planning via Model Checking*, Toulouse, France, 2002, pp. 81–88.

[23] H. L. S. Younes, D. J. Musliner, and R. G. Simmons, “A Framework for Planning in Continuous-time Stochastic Domains,” in *ICAPS*, 2003, p. 10.

[24] C. A. Miller, R. P. Goldman, H. B. Funk, P. Wu, and B. Pate, “A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles,” in *American Helicopter Society 60th Annual Forum Proceedings*, Baltimore, MD, 2004, pp. 2146–2157.

[25] C. Miller and R. P. Goldman, “‘Tasking’ Interfaces; Associates that Know Who’s the Boss,” in *Proceedings of the Fourth USAF/RAF/GAF Conference on Human/Electronic Crewmembers*, Kreuth, Germany, 1997.

[26] S. Gao, J. Avigad, and E. M. Clarke, “Delta-Complete Decision Procedures for Satisfiability over the Reals,” in *IJCAR*, 2012, pp. 286–300.

[27] S. Gao, M. Ganai, F. Ivancˇic, A. Gupta, S. Sankaranarayanan, and E. M. Clarke, “Integrating ICP and LRA Solvers for Deciding Nonlinear Real Arithmetic Problems,” p. 9.

[28] V. Brattka, P. Hertling, and K. Weihrauch, “A Tutorial on Computable Analysis,” in *New Computational Paradigms*, S. B. Cooper, B. Löwe, and A. Sorbi, Eds. New York, NY: Springer New York, 2008, pp. 425–491.
[29] D. J. Musliner, M. J. S. Pelican, and R. P. Goldman, “Incremental Verification for On-the-Fly Controller Synthesis,” *Electron. Notes Theor. Comput. Sci.*, vol. 149, no. 2, Feb. 2006.
[30] S. Gao, S. Kong, and E. M. Clarke, “dReal: An SMT Solver for Nonlinear Theories of Reals,” in *CADE*, 2013, pp. 208–214.
[31] S. Gao, S. Kong, and E. M. Clarke, “Satisfiability modulo ODEs,” in *FMCAD*, 2013, pp. 105–112.
[32] J.-A. Shin and E. Davis, “Processes and continuous change in a SAT-based planner,” *Artif. Intell.*, vol. 166, no. 1–2, pp. 194–253, Aug. 2005.
[33] S. Bogomolov, D. Magazzeni, A. Podelski, and M. Wehrle, “Planning as Model Checking in Hybrid Domains,” in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2014, pp. 2228–2234.
[34] D. Bryce, “A Happening-Based Encoding for Nonlinear PDDL+ Planning,” p. 8.
[35] D. Nau *et al.*, “SHOP2: An HTN Planning System,” *J. Artif. Intell. Res.*, vol. 20, pp. 379–404, 2003.
[36] M. Fox and D. Long, “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains.,” *JAIR*, vol. 20, pp. 61–124, 2003.
[37] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani, “Verifying Industrial Hybrid Systems with MathSAT,” *Electron. Notes Theor. Comput. Sci.*, vol. 119, no. 2, pp. 17–32, Mar. 2005.
[38] D. Jovanović and L. de Moura, “Solving Non-linear Arithmetic,” in *Automated Reasoning*, vol. 7364, B. Gramlich, D. Miller, and U. Sattler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 339–354.
[39] P. Van Hentenryck, D. McAllester, and D. Kapur, “Solving polynomial systems using a branch and prune approach,” *SIAM J. Numer. Anal.*, vol. 34, no. 2, pp. 797–827, 1997.
[40] L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” in *Machine Learning: ECML 2006*, vol. 4212, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293.
[41] J. Gaschnig, “Performance measurement and analysis of certain search algorithms,” Carnegie-Mellon University, CMU-CS-79-124, 1979.
[42] O. Maler, D. Nickovic, and A. Pnueli, “From MITL to Timed Automata,” in *Formal Modeling and Analysis of Timed Systems*, 2006, pp. 274–289.
[43] L. R. Humphrey, E. M. Wolff, and U. Topcu, “Formal Specification and Synthesis of Mission Plans for Unmanned Aerial Vehicles,” in *Proceedings of the AAAI Spring Symposium: Formal Verification and Modeling in Human-Machine Systems*, 2014, pp. 116–121.
[44] C. Finucane, G. Jing, and H. Kress-Gazit, “LTLMoP: Experimenting with language, temporal logic and robot control,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, pp. 1988–1993.
[45] S. Konrad and B. H. Cheng, “Facilitating the construction of specification pattern-based properties,” in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, pp. 329–338.
[46] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, “Simple On-the-fly Automatic Verification of Linear Temporal Logic,” in *Protocol Specification, Testing and Verification XV*, P. Dembiński and M. Średniawa, Eds. Boston, MA: Springer US, 1996, pp. 3–18.
[47] D. Nickovic, “Checking Timed and Hybrid Properties: Theory and Applications,” Theses, Université Joseph-Fourier - Grenoble I, 2008.
[48] D. Ničković and N. Piterman, “From MITL to Deterministic Timed Automata,” in *Formal Modeling and Analysis of Timed Systems*, vol. 6246, K. Chatterjee and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 152–167.
[49] M. M. Bersani, M. G. Rossi, and P. San Pietro, “On the Satisfiability of Metric Temporal Logics over the Reals,” *Electron. Commun. EASST*, vol. 66, 2013.
[50] M. M. Bersani, M. G. Rossi, and P. San Pietro, “Deciding Continuous-Time Metric Temporal Logic with Counting Modalities,” in *Proc. 7th International Workshop on Reachability Problems*, Berlin, Heidelberg, 2013, pp. 70–82.
[51] R. Howey, D. Long, and M. Fox, “VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL,” in *Proceedings ICTAI*, 2004, pp. 294–301.

[52] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, “Powertrain control verification benchmark,” in *Proceedings of the 17th international conference on Hybrid systems: computation and control - HSCC ’14*, Berlin, Germany, 2014, pp. 253–262.
[53] R. Singh, “Brusselator as a Reaction Diffusion System,” The Institute of Mathematical Sciences, Chennai, India, 2008.
[54] Brusselator, “Brusselator --- Wikipedia, The Free Encyclopedia,” 2016.

## LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

**ARMOR** AFRL Space Vehicle Directorate's next-generation satellite program on Advanced Reconfigurable Modular technology for improved Operational Resilience.

**BMC** Bounded Model Checking. Checking a property for correctness only over a bounded scope. The bounding might be metric (e.g., over a bounded interval of metric time) or discrete (e.g., permitting only a bounded number of discrete jumps in the trace of a hybrid automaton), or both.

**CBJ** Conflict-Directed Backumping. Search method that tracks conflicts (culprits) that cause search failure and uses that information to jump back to the most recent relevant decision when backtracking, instead of simply the last decision.

**CIRCA** Cooperative Intelligent Real-time Control Architecture, SIFT's architecture for hard real-time intelligent control.

**CODE** DARPA's Collaborative Operations in Denied Environments program.

**CPS** Cyber-physical system: a system involving both continuous processes and discrete logic (typically in control).

**CSM** Controller Synthesis Module: the component of CIRCA that automatically generates closed-loop, hard real-time controllers for asynchronously-operating systems.

**DBM** DARPA's Distributed Battle Management program.

**dREACH** CMU's front-end for the dREAL SMT solver, allowing users to describe hybrid automata in a convenient notation that is automatically translated to dREAL input for reachability solving.

**dREAL** CMU's SMT solver for δ-complete reasoning.

**HA** Hybrid Automaton. An automaton model combining continuous variables and discrete state transitions.

**HAMMER** The Highly Autonomous Mission Manager for Event Response architecture for satellite autonomy, which SIFT is developing with AFRL funding for the ARMOR program.

**HTN** Hierarchical Task Network. An AI planning method; SHOP2 is an HTN planner.

**MCTS** Monte Carlo Tree Search.

**MITL** Metric Interval Temporal Logic.

**ODE** Ordinary differential equations.

**Playbook®** SIFT's approach to supervisory control of automation, based on the notion of "calling a play," in which a vocabulary of actions is shared between automation and users, user intent is expanded into a full, executable mission plan through the use of a Hierarchical Task Network (HTN) planner, and execution is managed by a smart, plan-aware executive.

**SHOP2** The HTN planner used by HyCIRCA. Originally developed at the University of Maryland, the award-winning, open source SHOP2 planner is now maintained by SIFT.

**SIFT** Smart Information Flow Technologies is a small research company specializing in intelligent automation, verification, and human-centered systems.

**SMT** Satisfiability Modulo Theories. A reasoning system that couples one or more special-purpose theory solvers (e.g., for non-linear equations) with a satisfiability solver.

**STL** Signal Temporal Logic: a temporal logic that admits propositions over continuous-valued variables.

**TA** Timed Automaton. A finite-state automaton enhanced with real-valued clocks used to model system dynamics.

**TL** Temporal Logic.

## Appendix A: Büchi Automata for CIRCA CSM goals

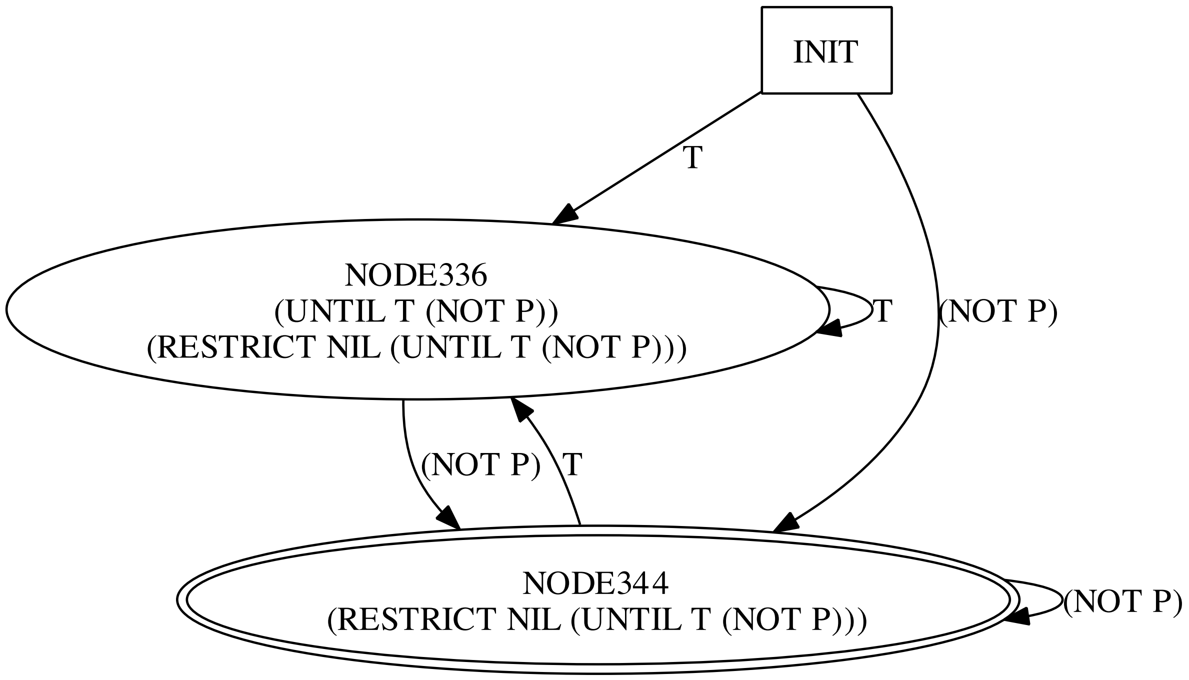

INIT
T
NODE336
(UNTIL T (NOT P))
(RESTRICT NIL (UNTIL T (NOT P)))
T
(NOT P)
(NOT P)
T
NODE344
(RESTRICT NIL (UNTIL T (NOT P)))
(NOT P)

**Figure 29 Achieve and hold automaton.**

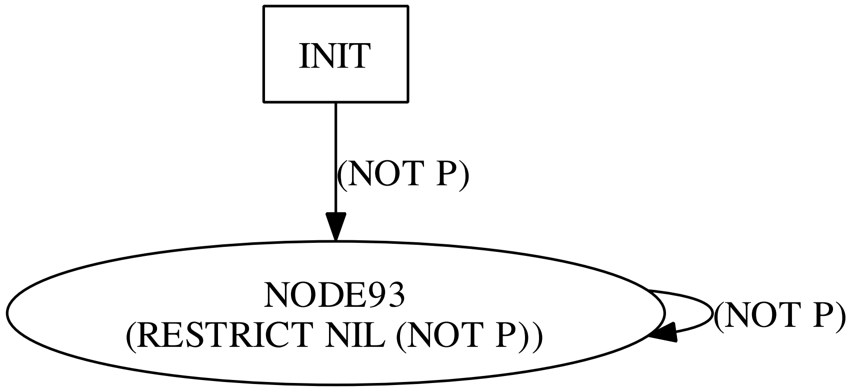


**Figure 30 Single achievement automaton.**

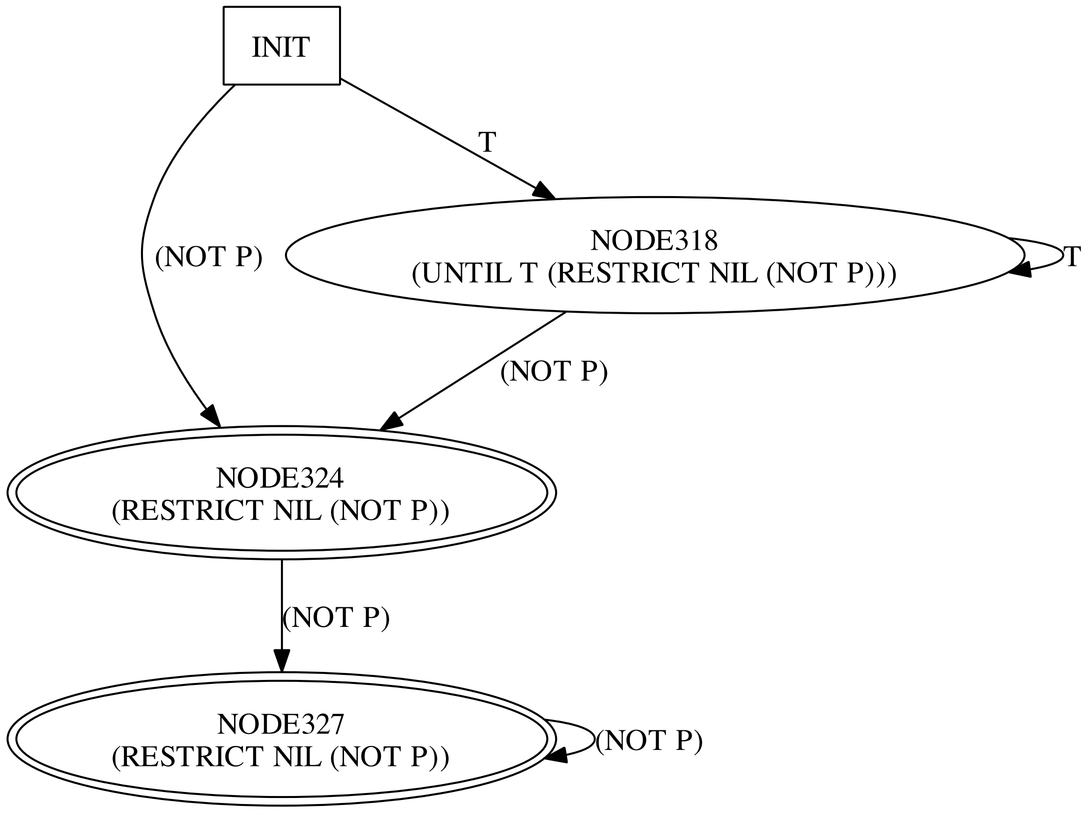


**Figure 31 Repeated achievement automaton.**

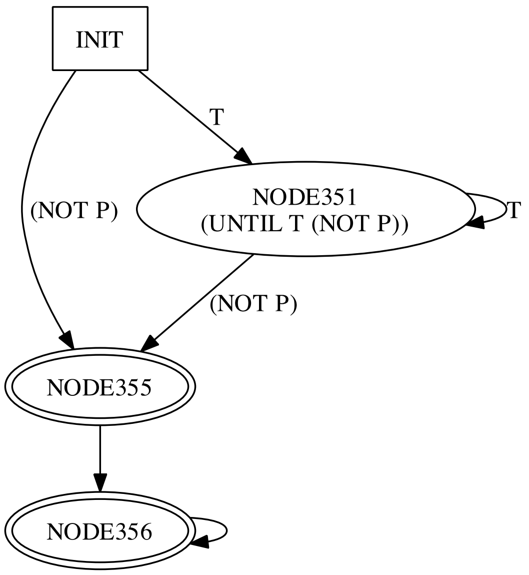


**Figure 32 Maintenance automaton.**

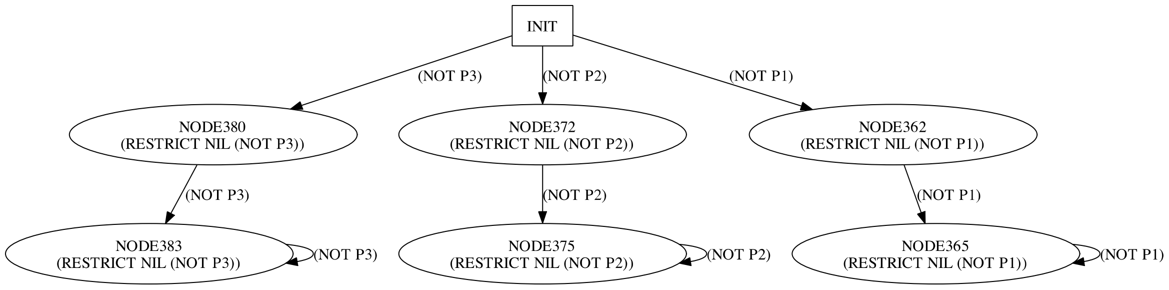

INIT
(NOT P3)
(NOT P2)
(NOT P1)
NODE380 (RESTRICT NIL (NOT P3))
NODE372 (RESTRICT NIL (NOT P2))
NODE362 (RESTRICT NIL (NOT P1))
(NOT P3)
(NOT P2)
(NOT P1)
NODE383 (RESTRICT NIL (NOT P3))
(NOT P3)
NODE375 (RESTRICT NIL (NOT P2))
(NOT P2)
NODE365 (RESTRICT NIL (NOT P1))
(NOT P1)

**Figure 33 Coverage automaton.**

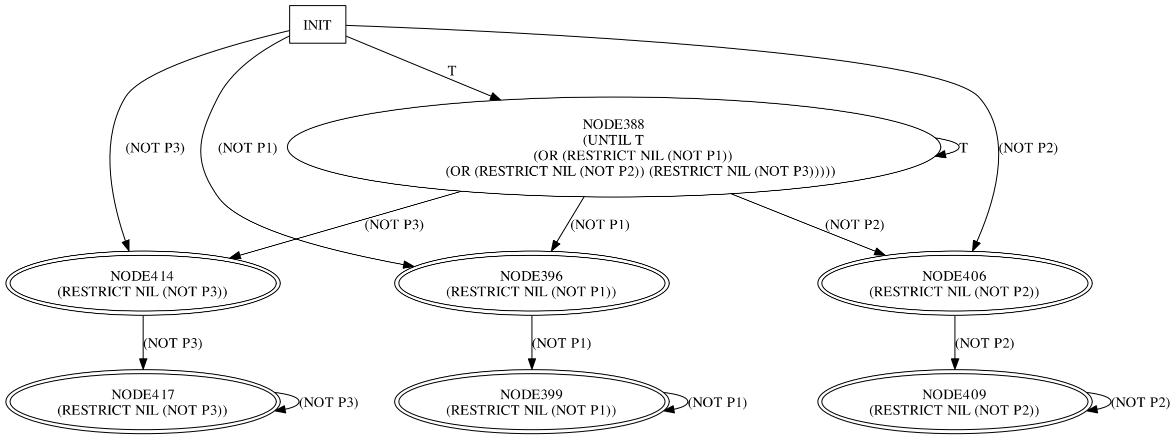

INIT
T
(NOT P3)
(NOT P1)
NODE388 (UNTIL T (OR (RESTRICT NIL (NOT P1)) (OR (RESTRICT NIL (NOT P2)) (RESTRICT NIL (NOT P3)))))
T
(NOT P2)
(NOT P3)
(NOT P1)
(NOT P2)
NODE414 (RESTRICT NIL (NOT P3))
NODE396 (RESTRICT NIL (NOT P1))
NODE406 (RESTRICT NIL (NOT P2))
(NOT P3)
(NOT P1)
(NOT P2)
NODE417 (RESTRICT NIL (NOT P3))
(NOT P3)
NODE399 (RESTRICT NIL (NOT P1))
(NOT P1)
NODE409 (RESTRICT NIL (NOT P2))
(NOT P2)

**Figure 34 Recurrent coverage automaton.**

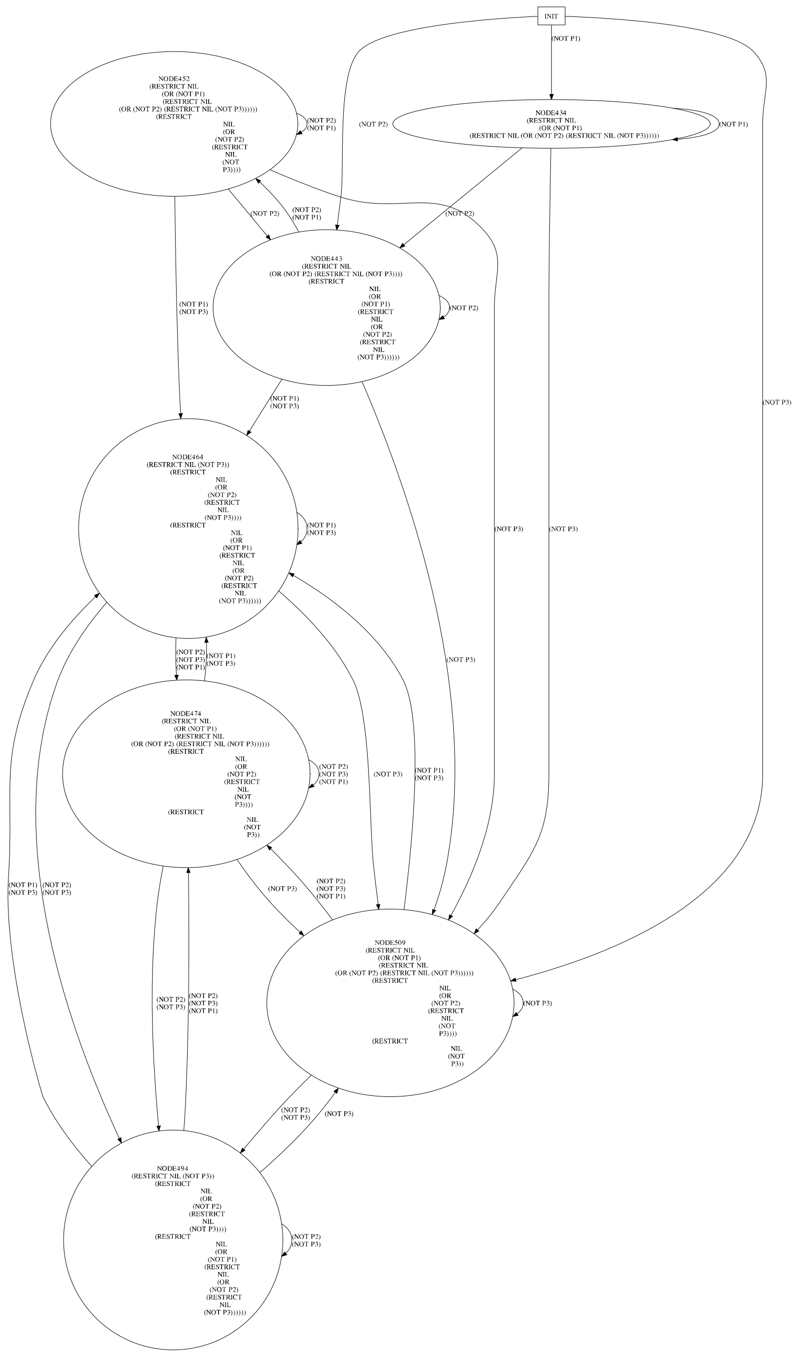


**Figure 35 Sequencing automaton.**

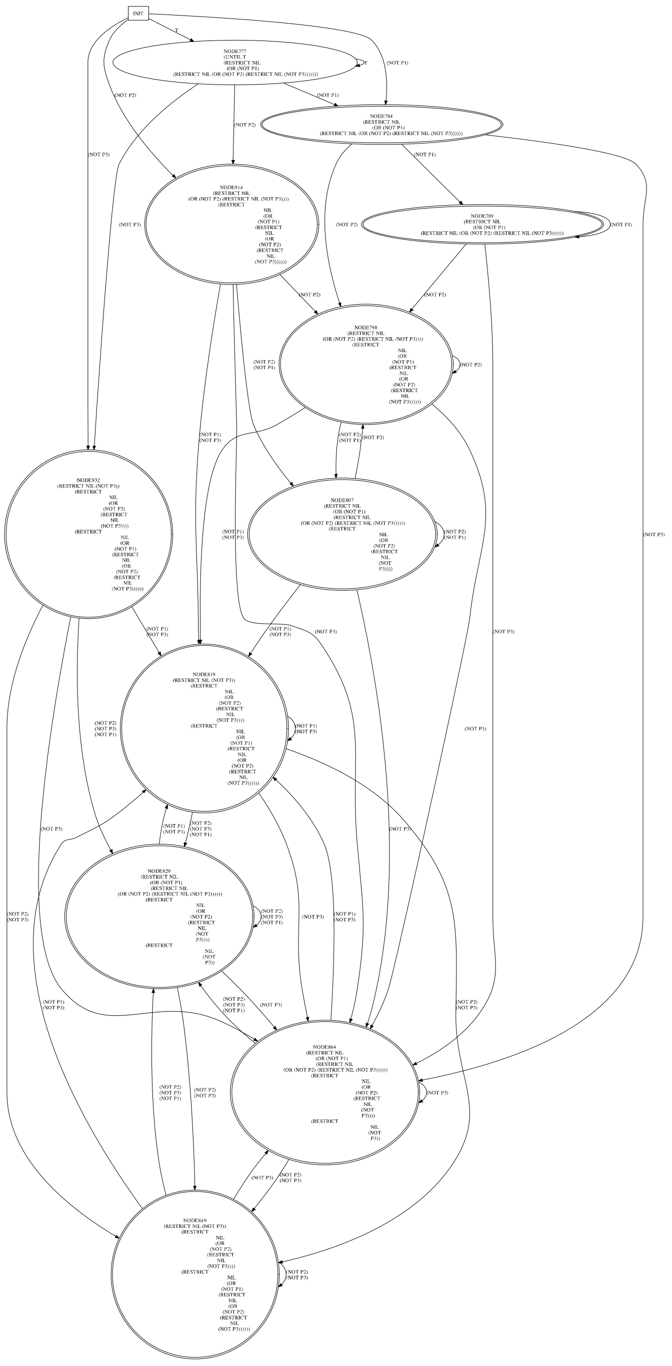


**Figure 36 Recurrent sequencing automaton.**

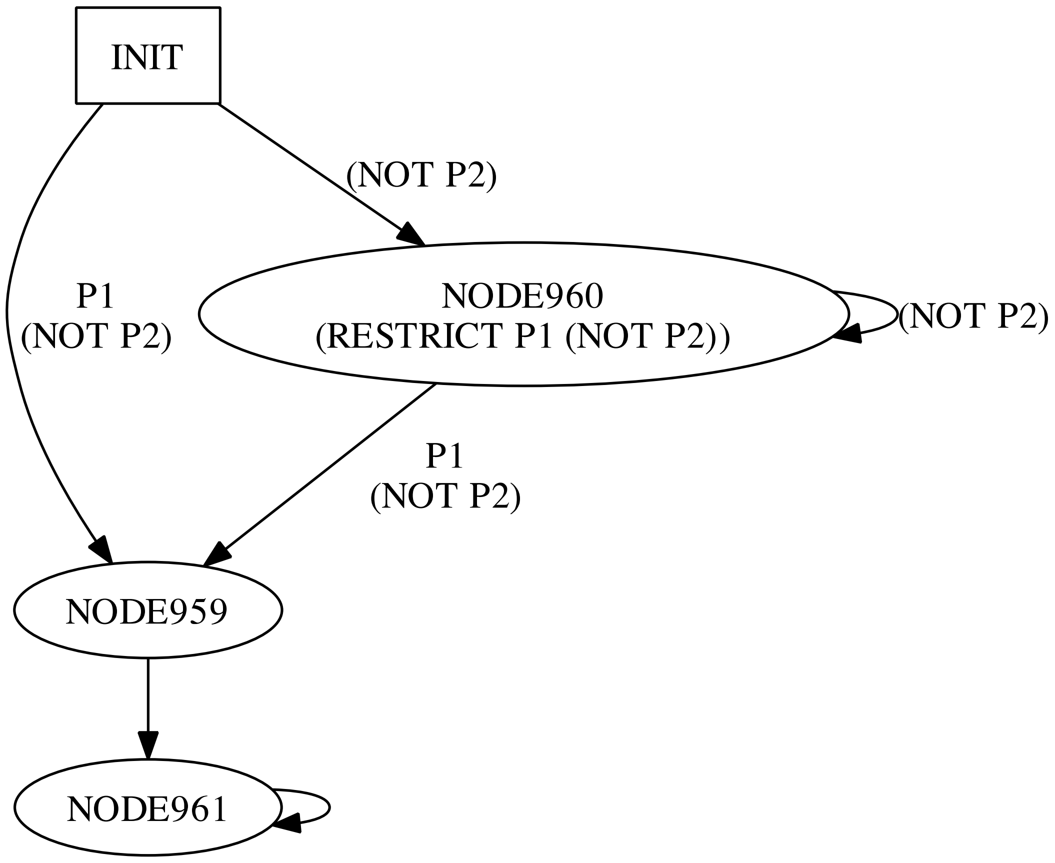


**Figure 37 Avoidance automaton.**

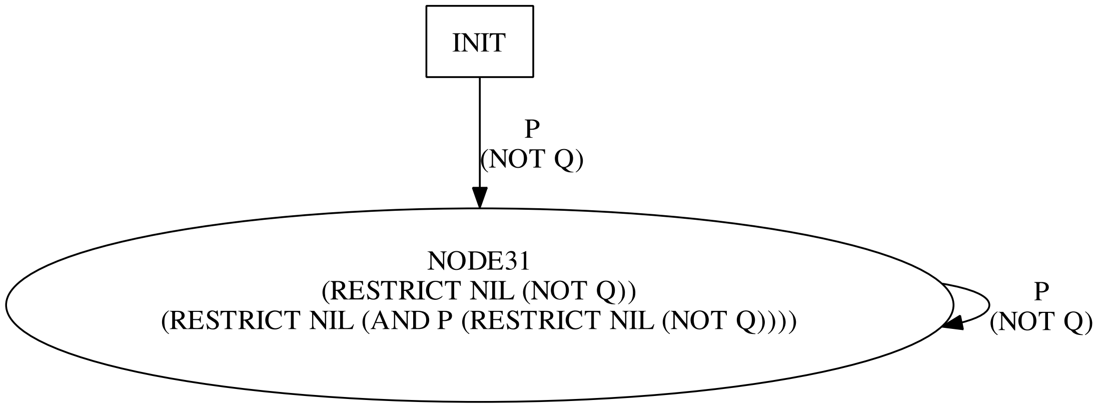


**Figure 38 Response automaton.**

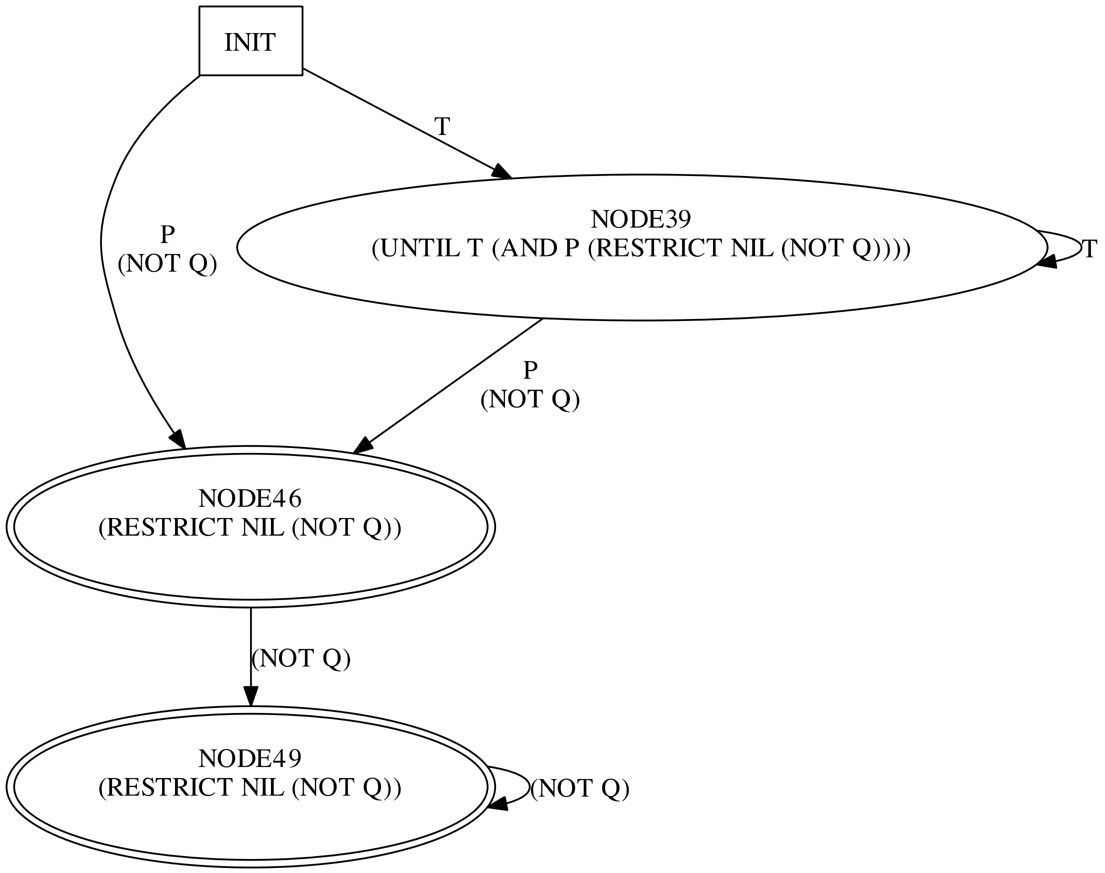


**Figure 39 Consistent response automaton.**

## Appendix B: SMT translation of CIRCA model

```
(set-logic QF_NRA_ODE)
(declare-fun fuel () Real [0.000000, 1000.000000])
(declare-fun posx () Real [-20.000000, 20.000000])
(declare-fun posy () Real [-20.000000, 20.000000])
(declare-fun time_clock1 () Real [0.000000, 1001.000000])
(declare-fun time_clock2 () Real [0.000000, 1001.000000])
(declare-fun time_clock3 () Real [0.000000, 1001.000000])
(declare-fun time_clock4 () Real [0.000000, 1001.000000])
(declare-fun velx () Real [-100.000000, 100.000000])
(declare-fun vely () Real [-100.000000, 100.000000])
(declare-fun fuel_0_0 () Real [0.000000, 1000.000000])
(declare-fun fuel_0_t () Real [0.000000, 1000.000000])
(declare-fun posx_0_0 () Real [-20.000000, 20.000000])
(declare-fun posx_0_t () Real [-20.000000, 20.000000])
(declare-fun posy_0_0 () Real [-20.000000, 20.000000])
(declare-fun posy_0_t () Real [-20.000000, 20.000000])
(declare-fun time_clock1_0_0 () Real [0.000000, 1001.000000])
(declare-fun time_clock1_0_t () Real [0.000000, 1001.000000])
(declare-fun time_clock2_0_0 () Real [0.000000, 1001.000000])
(declare-fun time_clock2_0_t () Real [0.000000, 1001.000000])
(declare-fun time_clock3_0_0 () Real [0.000000, 1001.000000])
(declare-fun time_clock3_0_t () Real [0.000000, 1001.000000])
(declare-fun time_clock4_0_0 () Real [0.000000, 1001.000000])
(declare-fun time_clock4_0_t () Real [0.000000, 1001.000000])
(declare-fun velx_0_0 () Real [-100.000000, 100.000000])
(declare-fun velx_0_t () Real [-100.000000, 100.000000])
(declare-fun vely_0_0 () Real [-100.000000, 100.000000])
(declare-fun vely_0_t () Real [-100.000000, 100.000000])
(declare-fun time_0 () Real [0.000000, 1001.000000])
(declare-fun mode_1_0 () Real [1.000000, 21.000000])
(declare-fun mode_2_0 () Real [1.000000, 29.000000])
(declare-fun mode_3_0 () Real [1.000000, 3.000000])
(declare-fun mode_4_0 () Real [1.000000, 3.000000])
(declare-fun mode_5_0 () Real [1.000000, 3.000000])
(declare-fun mode_6_0 () Real [1.000000, 3.000000])
(declare-fun mode_7_0 () Real [1.000000, 3.000000])
(declare-fun mode_8_0 () Real [1.000000, 3.000000])
(declare-fun mode_9_0 () Real [1.000000, 3.000000])
(declare-fun mode_10_0 () Real [1.000000, 3.000000])
(declare-fun mode_11_0 () Real [1.000000, 1.000000])
(declare-fun mode_12_0 () Real [1.000000, 1.000000])
(declare-fun mode_13_0 () Real [1.000000, 1.000000])
(declare-fun gamma_rts_model0_1 () Real [0.000000, 1.000000])
(declare-fun gamma_clock4_1 () Real [0.000000, 1.000000])
(declare-fun gamma_clock3_1 () Real [0.000000, 1.000000])
(declare-fun gamma_clock2_1 () Real [0.000000, 1.000000])
(declare-fun gamma_rts_model0_1_0_0 () Real [0.000000, 1.000000])
(declare-fun gamma_clock4_1_0_0 () Real [0.000000, 1.000000])
(declare-fun gamma_clock3_1_0_0 () Real [0.000000, 1.000000])
(declare-fun gamma_clock2_1_0_0 () Real [0.000000, 1.000000])
```

```
(define-ode flow_0
  ((= d/dt[time_clock1] (* 1 gamma_rts_model0_1))
   (= d/dt[time_clock2] (* 1 gamma_clock2_1))
   (= d/dt[time_clock3] (* 1 gamma_clock3_1))
   (= d/dt[time_clock4] (* 1 gamma_clock4_1))
   (= d/dt[gamma_rts_model0_1] 0)
   (= d/dt[gamma_clock4_1] 0)
   (= d/dt[gamma_clock3_1] 0)
   (= d/dt[gamma_clock2_1] 0) (= d/dt[fuel] 0) (= d/dt[posx] 0)
   (= d/dt[posy] 0) (= d/dt[velx] 0) (= d/dt[vely] 0)))
(assert (and (= mode_10_0 1) (= mode_9_0 1) (= mode_8_0 1) (= mode_7_0 1)
             (= mode_6_0 1) (= mode_5_0 1) (= mode_4_0 1) (= mode_3_0 1)
             (= mode_1_0 1) (= time_clock1_0_0 0) (= mode_2_0 1)
             (= fuel_0_0 1000) (= posy_0_0 0) (= posx_0_0 0) (= vely_0_0 0)
             (= velx_0_0 0) (= mode_11_0 1) (= time_clock4_0_0 0)
             (= mode_12_0 1) (= time_clock3_0_0 0) (= mode_13_0 1)
             (= time_clock2_0_0 0)))
(assert true)
(assert (and (= [time_clock1_0_t time_clock2_0_t time_clock3_0_t
                time_clock4_0_t gamma_rts_model0_1_0_0 gamma_clock4_1_0_0
                gamma_clock3_1_0_0 gamma_clock2_1_0_0 fuel_0_t posx_0_t
                posy_0_t velx_0_t vely_0_t]
               (integral 0. time_0
                 [time_clock1_0_0 time_clock2_0_0 time_clock3_0_0
                  time_clock4_0_0 gamma_rts_model0_1_0_0 gamma_clock4_1_0_0
                  gamma_clock3_1_0_0 gamma_clock2_1_0_0 fuel_0_0 posx_0_0
                  posy_0_0 velx_0_0 vely_0_0] flow_0))
             (=> (= gamma_rts_model0_1_0_0 0)
                 (not (= mode_1_0 1)))
             (=> (not (= mode_1_0 1))
                 (= gamma_rts_model0_1_0_0 0))
             (=> (= gamma_rts_model0_1_0_0 1)
                 (= mode_1_0 1))
             (=> (= mode_1_0 1)
                (= gamma_rts_model0_1_0_0 1))
             (=> (= gamma_clock4_1_0_0 0) (not (= mode_11_0 1)))
             (=> (not (= mode_11_0 1)) (= gamma_clock4_1_0_0 0))
             (=> (= gamma_clock4_1_0_0 1) (= mode_11_0 1))
             (=> (= mode_11_0 1) (= gamma_clock4_1_0_0 1))
             (=> (= gamma_clock3_1_0_0 0) (not (= mode_12_0 1)))
             (=> (not (= mode_12_0 1)) (= gamma_clock3_1_0_0 0))
             (=> (= gamma_clock3_1_0_0 1) (= mode_12_0 1))
             (=> (= mode_12_0 1) (= gamma_clock3_1_0_0 1))
             (=> (= gamma_clock2_1_0_0 0) (not (= mode_13_0 1)))
             (=> (not (= mode_13_0 1)) (= gamma_clock2_1_0_0 0))
             (=> (= gamma_clock2_1_0_0 1) (= mode_13_0 1))
             (=> (= mode_13_0 1) (= gamma_clock2_1_0_0 1))))
(assert false)
(check-sat)
(exit)
```