A Logic-based Modeling and Verification of CPS

Neda Saeedloei
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75080
neda.saeedloei@student.utdallas.edu

Gopal Gupta
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75080
gupta@utdallas.edu

ABSTRACT

Cyber-physical systems (CPS) consist of perpetually and concurrently executing physical and computational components. The presence of physical components require the computational components to deal with continuous quantities. A formalism that can model discrete and continuous quantities together with concurrent, perpetual execution is lacking. In this paper we report on the development of a formalism based on *logic programming* extended with *co-induction*, *constraints over reals*, and *coroutining* that allows CPS to be elegantly modeled. This logic programming realization can be used for verifying interesting properties as well as generating implementations of CPS. We illustrate this formalism by applying it to elegant modeling of the *reactor temperature control system*. Interesting properties of the system can be verified merely by posing appropriate *queries* to this model. Precise parametric analysis can also be performed.

1. INTRODUCTION

Cyper-physical systems (CPS) combine computational and physical elements in tight coordination. In addition to discrete computation, the presence of physical elements require CPS to handle continuous quantities (e.g., time, distance, acceleration, temperature, etc.). Cyber-physical systems are becoming ubiquitous. Almost every device today has (concurrent, communicating) controllers that read inputs through sensors, do some processing and then perform actions through actuators in a coordinated manner. Examples include controller systems in cars (Anti-lock Brake System, Cruise Controllers, Collision Avoidance, etc., that communicate through a network with each other), automated manufacturing, smart homes, robots, etc. Unlike embedded systems, CPS are not stand-alone systems. Rather, they are a network of interacting and concurrently executing elements that have physical inputs and outputs. In addition, CPS are assumed to run forever [5, 8]. Due to fundamentally discrete nature of computation, researchers have had difficulty dealing with continuous quantities in computations (typical approaches discretize continuous quantities, e.g., time). Likewise, modeling of perpetual computations is not straightforward (only recently, techniques such as co-inductive logic programming [17, 4] have been introduced to operationally realize rational, infinite computations). Concurrency is reasonably well understood, but when combined with continuous quantities and with perpetual computations, CPS become extremely hard to model faithfully. In this paper we develop techniques for faithfully and elegantly modeling CPS and verifying their properties. We consider communicating hybrid automata as the underlying model, and illustrate their use in specifying and verifying highly complex CPS. Our approach is based on using *logic programming* (*LP*) for modeling computations, *constraint logic programming* (*CLP(R)*) for modeling continuous physical quantities, *co-induction* for modeling perpetual execution and *coroutining* for modeling concurrency in CPS. CPS are thus represented as coroutined co-inductive constraint logic programs which are subsequently used to elegantly verify cyber-physical properties of the system relating to safety, liveness and utility. These logic programs can also be used for automatically generating implementation code for the CPS.

To illustrate our techniques, we consider the reactor temperature control system [19], and show how it can be naturally and elegantly modeled (and its properties verified) as a network of communicating hybrid automata implemented as coroutined co-inductive CLP(R) programs. Note that our framework for modeling and verifying CPS, provides diagnostic information that can be used in design of the system. For instance, if our system fails to satisfy the safety requirement, it will generate a time trace of events that leads to violation of safety property. Another interesting feature of our framework is its ability to perform precise parametric analysis. Parametric analysis is used to determine necessary and sufficient constraints on parameters under which correctness requirements are met. Using our method we were able to compute exact bounds on parameters of the reactor temperature control system thereby improving on the results of Henzinger and Ho [6], who only compute them approximately.

Our contribution includes developing practical and faithful realization of CPS as co-inductive coroutined constraint logic programs over reals. What is noteworthy in our realization of CPS is that, in contrast to other approaches, we do not discretize the physical quantities involved. Rather these physical quantities are assumed to range over continuous real values, with relationship between them modeled as constraints over reals. We assume that the reader is familiar with hybrid automata [11, 1] constraint logic programming over reals [7] and co-inductive logic programming [4].

2. MODELING CPS WITH COROUTINED CO-INDUCTIVE CLP(R)

CPS have the following four characteristics [5, 8]: (i) they perform discrete computations, (ii) they deal with continuous quantities, (iii) they are concurrent, and (iv) they run forever. We assume that CPS are communicating hybrid automata that control physical systems. A hybrid automa-

ton is a finite-state automaton, extended with a set of real-valued variables. The *control locations (states)* of the automaton are labeled with evolution laws, which are specified as differential equations. The values of the variables change continuously with time according to the associated law at each location. If the value of a variable x does not change in a location, then the evolution law is represented by $\dot{x} = 0$ and can be eliminated. Each location is also labeled with an invariant condition that must hold when the control resides at the location. The transitions of the automaton are labeled with guarded sets of assignments. A transition is enabled when the associated guard is true and its execution modifies the values of the variables according to the assignments. There is a labeling function that assigns a label (or synchronization letter) to each transition. There is also an initial condition that describes the set of possible values for the system when it starts in the initial location.

The reactor temperature control system is a traditional example of a cyber-physical system. The system consists of a reactor core and two control rods that control the temperature of the reactor core. The goal is to keep the temperature between temperatures θ_m and θ_M. If the temperature reaches θ_M, then it should be decreased by introducing one of the control rods into the reactor core. Figure 1 shows the hybrid system of this example. Variable θ measures the temperature, variables r_1 and r_2 measure the time that has elapsed since rod_1 and rod_2 were removed from the reactor core, respectively. Variables c_1 and c_2 are used by the controller so that the time of transitions can be remembered. Initially θ is θ_m degrees and both control rods are outside of the reactor core. In this case, θ rises according to the differential equation $\dot{\theta} = \frac{\theta}{10} - 50$, location *no_rod*. θ decreases according to the differential equations $\dot{\theta} = \frac{\theta}{10} - 56$ (location rod_1) and $\dot{\theta} = \frac{\theta}{10} - 60$ (location rod_2) depending on the control rod used. A control rod may be used again, if $W \geq 0$ units of time have elapsed since it was last removed. If θ cannot decrease because no control rod is available, then a shutdown of the reactor is necessary. A shutdown of the system should be prevented. As we mentioned earlier, communicating hybrid automata constitute the foundations of CPS. In the past we have developed a system for converting communicating timed automata (and communicating pushdown timed automata (PTA)) to co-inductive CLP(R) programs [15]. The system models a timed automaton/pushdown timed automaton as a set of transition rules (one rule per transition in the automaton), where each rule is extended with clock constraints and stack actions (in case of PTA). This system is extended to handle hybrid automata. In our formalism, hybrid automata are modeled as *logic programs* [9, 18], physical quantities are represented as continuous quantities (i.e., not discretized) and the constraints imposed on them by transitions are faithfully modeled with *constraint logic programming over reals* [7]. By considering *co-inductive logic programming* [4], we are able to model the non-terminating aspect of hybrid automata, and finally concurrency between hybrid automata will be handled by allowing *coroutining* (realized via delay declarations of Prolog [18] in our subsequent implementation) within logic programming computations.

Constraint logic programming (CLP) is a powerful extension of LP that allows one to reason about continuous quantities (such as real time) via constraint-solving over reals. It permits an elegant blending of computations over both discrete and continuous quantities. Real-time features along with other physical dynamics of real-time systems/CPS can be expressed as constraints over reals so that the entire system can be modeled and verified in CLP(R) [15, 14].

Co-induction is a powerful technique for reasoning about unfounded sets, unbounded structures, and interactive computations. Elegant co-inductive extensions of logic programming have been proposed. Co-inductive LP (Co-LP) gives an operational semantics to greatest fix point based computations. An important application of co-inductive LP is in directly representing and verifying properties of Kripke structures and ω-automata (automata that accept infinite strings). The perpetual execution of CPS can be elegantly expressed via Co-LP.

The coroutining facility of logic programming (in particular Prolog) is realized via built-in predicates such as *freeze*, *when*, etc. Coroutining deals with having Prolog goals scheduled for execution as soon as some conditions are fulfilled. In Prolog the most commonly used condition is the instantiation (binding) of a variable. Scheduling a goal to be executed immediately after a variable is bound can be used to model the transitions taken on synchronization letters.

Our logic programming realization of CPS is illustrated by its application to the reactor temperature control system in the next section.

3. MODELING THE REACTOR TEMPERATURE CONTROL SYSTEM

For modeling the reactor temperature control system, we set $\theta_m = 510$, and $\theta_M = 550$. We solve the differential equations in each location and determine how long the system remains in this location and how the variables (temperature, and clocks) change during this time. Solving the differential equation for location *no_rod* results in: $\theta(t) = 10e^{(t/10)}+500$. We can determine at which time point the transitions add_1 or add_2 are taken place by computing the time the reactor reaches the critical temperature θ_M; this is performed by solving the equation $\theta(t) = \theta_M$. Similarly, solving the differential equations for locations rod_1 and rod_2 results in: $\theta(t) = -10e^{(t/10)} + 560$ and $\theta(t) = -50e^{(t/10)} + 600$ respectively. By solving the equation $\theta(t) = \theta_m$, we can determine at which time point the transitions $remove_1$ or $remove_2$ are taken place. The set of transition rules of the three hybrid automata of this example, extended with clock operations, set of constraints corresponding to the guards and invariant conditions and evolution laws, are expressed via predicates `c/11`, `r1/7` and `r2/7`, which are not presented here but shown in [13]. Once each hybrid automaton in the system is realized as a set of transition rules via logic programming, the coroutined, co-inductive predicate `contr/7`, realizes the controller automaton, calling the `c/11` predicate repeatedly; while sending appropriate synchronization letters to rod_1 and rod_2. Coroutined, co-inductive predicates `rod1/6` and `rod2/6` similarly call `r1/7` and `r2/7` respectively, on receiving synchronization letters from the controller. These predicates are also shown in [13]. Finally the `main/3` predicate represents the concurrent execution of all the components of the reactor temperature control system.

```
main(S, T, W) :-
  {T - Tr1 = W, T - Tr2 = W},
  freeze(S, (rod1(S, s0, s0, Tr1, Tr2, W);
             rod2(S, s0, s0, Tr1, Tr2, W))),
  contr(S, s0, T, 510, Tc1, Tc2, 1).
```

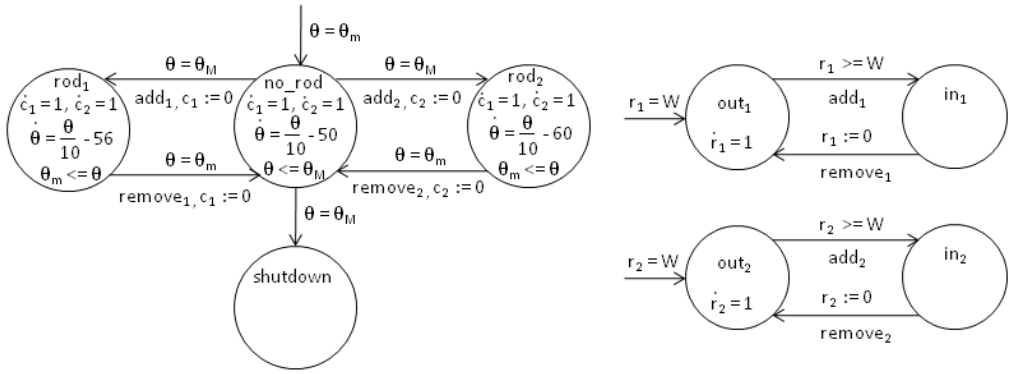

Figure 1: The reactor core, and control rod automata

The first argument of `main/3` is the list of synchronization letters along with their time-stamps, which is generated by the controller and sent to two rods (nondeterministically). The second argument is the initial wall clock time, and W is the parameter of the system explained earlier. The condition that both rods are available initially, is specified using the set of constraints $\{T-Tr1 = W, T-Tr2 = W\}$ in which $Tr1$ and $Tr2$ represent the clocks for rod1 and rod2 respectively.

Once the entire system is modeled as a co-inductive coroutined CLP(R) program, it can be used for finding the value of parameter W that guarantees the safety of the system. Calling the `main` predicate, with W as an unknown parameter, we obtain $W \leq 38.0666$, which is a necessary and sufficient condition on the parameter W that prevents the reactor from shutdown. The verification requires 0.010 seconds on an Intel dual core 3.16 GHz processor with 4.00 GB of RAM. Our logic programming realization of system can also be used to verify interesting properties of the system by posing queries. Here we exploit the natural ability of logic/constraint programming systems to explore the entire state space of a program, merely by backtracking. Given a property `Q` to be verified, we specify its negation as a logic program. Let's call this predicate `notQ`. If the property `Q` holds, the query `notQ` will fail w.r.t. the logic program that models the system. If the query `notQ` succeeds, the answer provides a counter example to why the property `Q` does not hold. To prove the *safety* property, we define the `unsafe/3` predicate which looks for an accepting timed trace that contains a *shutdown* event. Calling the predicate `unsafe/3` for any values of $T > 0$ and $W \leq 38.0666$ fails, which indicates the safety of the system.

```
unsafe(S, T, W) :-
  main(S, T, W), member((shutdown, Ts), S).
```

Henzinger and Ho have also analyzed the reactor temperature control system with their symbolic model checker, HYTECH [6]. HYTECH analyzes a class of hybrid systems expressed via linear hybrid automata in which physical quantities exhibit constant derivatives. Since the hybrid automaton for the controller involves non-constant derivatives, their work employs two methods for converting it to linear hybrid automaton. In the first method which is called the *rate translation*, the derivative of the temperature is approximated in three locations of the controller hybrid automaton by rate intervals of $[-5, -1]$, $[-9, -5]$ and $[1, 5]$ for locations rod_1, rod_2 and *no_rod*, respectively. The analysis of this converted hybrid automaton by HYTECH concludes $W < 20.44$ as a necessary and sufficient condition on the parameter W that prevents the reactor from shutdown. Using our system, we found out that this requirement on parameter W is indeed a sufficient condition, however, it is not a necessary condition, since for any value of W in which $W \leq 38.0666$ the system is still safe. Their second method for converting a nonlinear hybrid automaton to a linear hybrid automaton is called *clock translation* and it has two steps. In the first step, the nonlinear variable of controller automaton is replaced by clock; as a result θ is replaced by t_θ. In the second step, the resulting automaton is over approximated by a linear automaton with the invariants $10t_\theta \leq 161$, $10t_\theta \leq 89$, and *true* for locations rod_1, rod_2 and *no_rod*, respectively. The original differential equations are also replaced by $\dot{t_\theta} = 1$ in all the locations of the hybrid automaton. The analysis of this converted hybrid automaton by HYTECH concludes $W < 37.8$ as a weaker condition on parameter W. This result is closer to the result generated by our system. However, in our approach, these parameters are computed with exact precision, as our framework for modeling hybrid automata directly handles the physical quantities constrained by nonconstant derivatives, as long as the guards associated with transitions are of the form $x \sim a$, where $\sim \in \{=, <, >, \leq, \geq\}$ and a is a constant. In other words, we are not using any translation and approximation method for converting non-linear variables to linear variables; therefore, our system computes the parameter bounds exactly, and is free of possible imprecision that might be introduced if approximation methods are used. Note that one needs to use an appropriate solver in order to solve the set of constraints corresponding to the invariants, evolution laws, and guards on the transitions.

4. CONCLUSION AND RELATED WORK

This paper presents a general framework for modeling and verifying cyber-physical/hybrid systems. CPS are normally composed of set of processes that execute concurrently. The processes interact with each other through sending and receiving signals and run forever. Communicating hybrid automata [11, 12] constitute the foundations of CPS. In our framework, hybrid automata are modeled as *logic programs* [9], physical quantities are faithfully represented as continuous quantities (i.e., not discretized) and the constraints imposed on them by CPS physical interactions are modeled with *constraint logic programming over reals*. By considering *co-inductive logic programming* we are able to nat-

urally model the non-terminating aspect of CPS. Finally, concurrency is handled by allowing *coroutining* within logic programming computations. As a result, CPS are modeled as coroutined, co-inductive CLP(R) programs which can be used for verifying interesting properties of the system such as safety, utility, and liveness. Our approach can also be used to perform precise parametric analysis of CPS and generating their implementations.

The reactor temperature control system has been recently adopted by researchers [10] as a benchmark problem in modeling and verifying CPS. Henzinger and Ho have analyzed this system with HYTECH; a symbolic model checker for linear hybrid automata [6]. Two methods are employed in their work for converting the nonlinear hybrid automata to linear hybrid automata: rate translation and clock translation. The parametric analysis of the reactor temperature control system by HYTECH, after rate translation, results in a very conservative constraint on W. Alur et al. have presented a general framework for the formal specification and algorithmic analysis of hybrid systems [12]. This work is also restricted to linear hybrid systems where all variables follow piecewise-linear trajectories. A variant of the reactor temperature control system is analyzed using the KRONOS tool, another symbolic model checker for timed and hybrid automata. In this variant of the system, temperature rises and decreases at fixed rates, which is not faithful to the original problem. Back and Cerschi have modeled and verified a variant of the reactor temperature control system using continuous action systems [2]. In the variant of the system they have considered, temperature rises and decreases at fixed rates also. The process of verifying safety property in their model is very lengthy and complex. They start by generating the state chart of the temperature control system to get a first approximation of the invariant for proving the safety property. Then, they keep adding information to the system's states in order to figure out an invariant strong enough to ensure safety.

We showed how the co-inductive CLP(R)-based realization of hybrid automata along with coroutining can be used to elegantly and faithfully model the reactor temperature control system as well as verify its safety property. Our approach can handle non-constant derivatives directly, i.e., they do not need to be approximated (as long as a solver, such as the CLP(R) solver, is available that can handle them). Using our model, we are able to determine the exact bound on the parameter W that guarantees the safety of the system. Our modeling of the system based on hybrid automata and coroutined co-inductive CLP(R) is simpler, more elegant, and more precise than other approaches. In addition, unlike other realizations of CPS that discretize physical quantities, we treat them as continuous quantities, and do not discretize them.

Timed concurrent constraint (TCC) programming [16] comes close to our work. Timed concurrent constraints have also been considered for verification [3]. TCC does not consider perpetual computations and works only with least fixpoints of programs. Our work can be regarded as a practical realization of TCC as well as its extension with co-induction to handle perpetual computations.

To conclude, a combination of constraints over reals, co-induction, and coroutining provides an expressive, and easy-to-use formalism for modeling and analyzing complex real-time systems and CPS modeled as communicating hybrid and timed automata. In fact, our framework is a general framework that can be applied to complex systems to handle any continuous quantity such as time, temperature, distance, pressure, etc. The logic programming based approach is simpler and more elegant than other approaches that have been proposed for this purpose. It is also more precise.

5. REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[2] R.-J. B. Cristina and C. Cerschi. Modeling and verifying a temperature control system using continuous action systems. In *Proc. of the 5th Int. Workshop on FMICS*, 2000.

[3] M. Falaschi and A. Villanueva. Automatic verification of timed concurrent constraint programs. *TPLP*, 6(3):265–300, 2006.

[4] Gopal Gupta et al. Coinductive logic programming and its applications. In *ICLP*, pages 27–44, 2007.

[5] R. Gupta. Programming models and methods for spatiotemporal actions and reasoning in cyber-physical systems. In *NSF Workshop on CPS*, 2006.

[6] T. A. Henzinger and P. hsin Ho. Hytech: The cornell hybrid technology tool. In *Hybrid Systems II, LNCS 999*, pages 265–293. Springer, 1995.

[7] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.

[8] E. A. Lee. Cyber-physical systems: Design challenges. In *ISORC*, May 2008.

[9] J. W. Lloyd. *Foundations of logic programming / J.W. Lloyd*. Springer-Verlag, Berlin, New York, 2nd, extended ed. edition, 1987.

[10] R. A. Thacker et al. Automatic abstraction for verification of cyber-physical systems. In *ICCPS*, pages 12–21, 2010.

[11] R. Alur et al. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.

[12] R. Alur et al. The algorithmic analysis of hybrid systems. *TCS*, 138:3–34, 1995.

[13] N. Saeedloei and G. Gupta. A logic based model for the reactor temperature control system. http://www.utdallas.edu/ nxs048000/reactor.pdf.

[14] N. Saeedloei and G. Gupta. Timed definite clause omega-grammars. In *ICLP (Technical Communications)*, pages 212–221, 2010.

[15] N. Saeedloei and G. Gupta. Verifying complex continuous real-time systems with coinductive CLP(R). In *LATA*, pages 536–548, 2010.

[16] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *LICS*, pages 71–80, 1994.

[17] L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-logic programming: Extending logic programming with coinduction. In *ICALP*, pages 472–483, 2007.

[18] L. Sterling and E. Shapiro. *The art of Prolog (2nd ed.): advanced programming techniques*. MIT Press, Cambridge, MA, USA, 1994.

[19] X. Nicollin et al. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, pages 149–178, 1992.