

# Hardware Implementation of Temporal Nonmonotonic Logics

Insu Song and Guido Governatori

School of Information Technology & Electrical Engineering  
The University of Queensland, Brisbane, QLD, 4072, Australia  
{insu, guido}@itee.uq.edu.au

**Abstract.** In order to apply nonmonotonic logics for specifying industrial automation controllers, we define (1) a method to extend atemporal nonmonotonic logics with temporal operators and (2) a mapping of these new temporal nonmonotonic logics into a Metric Temporal Logic. This mapping provides a formal specification method for real-time temporal reasoning digital circuits for the temporal nonmonotonic logics. We present our method in the context of synthesizing custom digital hardware (called *agent chip*) automatically from high level agent specifications.

**Keywords:** agent, chip design, nonmonotonic logic, knowledge representation, temporal logic.

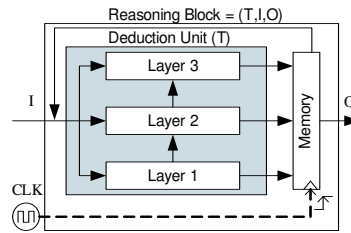
## 1 Introduction

Previously, Song and Governatori [12] described a method for synthesizing *agent chips* from high-level agent specifications. An agent chip is a custom hardware implementation of an agent specification, which performs several million times faster than conventional CPU-based systems. In the previous work, however, temporal expressions are not allowed in the specification language and a formal specification method for implementing temporal aspects of agent chips is missing. Time is a central factor in any automation controllers. Any industrial controllers must reason with timing issues of input and output control signals.

In this paper, we present a method for producing real-time temporal reasoning digital circuits for temporal nonmonotonic logics. This method extends the previous method for synthesizing *agent chips* [12]. Particularly, we provide nonmonotonic temporal logics for specifying real-time reasoning processors, which are suitable for reactive agent systems that are targeted for industrial automation controllers. The need for such a method is becoming acute since custom chips (e.g., FPGAs<sup>1</sup> and ASICs<sup>2</sup>) are now more and more affordable and vital applications such as sensor networks, smart sensors, autonomous mobile robots, and even small consumer electronic devices, require (a) low profile features, such as smaller size and power efficiency, and (b) high performance real-time features.

<sup>1</sup> FPGAs (Field Programmable Gate Arrays) are configurable digital circuits.

<sup>2</sup> ASICs (Application Specific Integrated Circuits).



**Fig. 1.** A reasoning block with three layers. A reasoning block is specified by a knowledge base  $T$ , an input specification  $I$ , and an output specification  $O$ .

Extending any logical languages, however, particularly nonmonotonic logics, with temporal operators maintaining real-time reasoning property is not easy, because reasoning under temporal logic is generally undecidable or EXPSPACE-complete [1]. In this paper, however, we propose a simple solution for extending propositional languages with temporal operators for automation controllers while maintaining real-time reasoning capability of the agent chip described in [12].

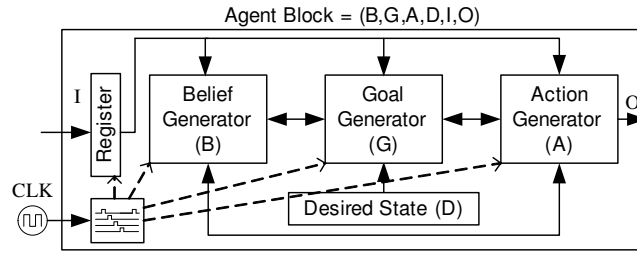
The basic idea is that we separate the evaluation of temporal operators from the main reasoning process in order to convert temporal languages into atemporal languages. We, then, build a separate reasoning circuit especially for the evaluation of temporal formulas. We can, then, combine this temporal reasoning circuit and the agent chip circuit produced by the method presented in [12] to produce a complete agent chip from a high-level agent specification that allows temporal expressions.

Particularly, we extend propositional languages with bounded temporal operators of Metric-Temporal Logic [6,7] which is an extension of Linear Temporal Logic. Since many nonmonotonic logics can be mapped into classical logic (e.g., Default Logic [2], Defeasible Logic [12], Layered Argumentation System [12,11]), we can formulate various temporal nonmonotonic logics and temporal argumentation systems maintaining real-time reasoning property based on this idea.

This paper is organized as follows. In Section 2, we overview an agent-based chip specification method described in [12]. In Section 3, we then show how to separate temporal reasoning from the main reasoning process. In Section 4, we define a Metric temporal logic (MTL), a method to extend propositional languages, and a mapping of these languages into the MTL. In Section 5, we conclude this paper with some remarks and related works.

## 2 Agent Based Chip Specification

In order to provide the context of this paper, we briefly describe the agent chip architecture [12]. An agent chip is a hardware implementation of a high-level agent specification. It is basically an electronic circuit that produces outputs in reaction to its inputs, which are connected to the agent's environment. The basic building blocks of an agent chip are the *reasoning blocks* shown in Figure 1. Each reasoning block is specified by a



**Fig. 2.** Agent Block architecture: an example agent architecture consisting of three control blocks: belief generator (B), goal generator (G), and action generator (A).

knowledge base  $T$ , an input specification  $I$ , and an output specification  $O$ . The deduction unit of a reasoning block is implemented on FPGAs or ASICs as *combinational logic circuits*<sup>3</sup>, which is organized in layers corresponding to the layers of the knowledge base  $T$ . The reasoning block also contains a *memory unit*, which provides temporal reasoning capabilities and other utility functions (e.g., counters, timers, arithmetic) that cannot be expressed in propositional languages. The circuit generation method of the deduction unit is described in [12] and the details of the language used in this method is described in [11]. The memory unit provides evaluation of temporal formulas separating temporal evaluation from the main reasoning process of the reasoning block. Section 4 defines a formal specification method for the implementation of the memory unit for supporting evaluation of temporal formulas based on a Metric-Temporal Logic (MTL). MTL[6] is an extension of Linear Temporal Logic to provide more convenient temporal operators for expressing bounded responses in formal specifications of real-time systems.

Figure 2 shows an example agent architecture called *Agent Block* which consists of three reasoning blocks: a belief generator (B), a goal generator (G), and an action generator (A). Each agent block is an autonomous system that performs actions in order to achieve a certain desired state  $D$ . An agent block operates in the following four-stage cycles:

1. At the beginning of the first stage of a cycle, the input signals in  $I$  are captured in the register. Then, the agent reasons about the current state of the world in  $B$  with its background knowledge, the memory contents of  $B$ ,  $G$ , and  $A$ .
2. At the second stage of the cycle, the (belief) conclusions in  $B$  are captured in the memory unit of  $B$ . Then, the goal generator ( $G$ ) decides what goals are needed to achieve the desired state with its goal description knowledge, the memory contents of  $B$ ,  $G$ , and  $A$ .
3. At the third stage of the cycle, the (goal) conclusions in  $G$  are captured in the memory unit of  $G$ . Then, the set of goals instantiates a particular behavioral specification from a set of plans in the action generator ( $A$ ) and produces appropriate actions for

<sup>3</sup> A digital circuit that does not have internal states, i.e., the output signals are solely defined by the input signals of the circuit.

the output ( $O$ ) with its action description knowledge base, the memory contents of  $B$ ,  $G$ , and  $A$ .

4. At the last stage of the cycle, the (action) conclusions in  $A$  are captured in the memory unit of  $A$  which is connected to the output port of the agent block.

The four stage cycle is driven by the clock signal (CLK) shown in the figure. The register and the memory units of the three reasoning blocks keep the interconnecting signals steady while the input signals or the output signals of a reasoning block are used by other reasoning blocks. Consequently, the frequency of the clock is determined by the time required to draw all of the conclusions in each deduction unit. That is, the computation bottle-neck is in the combinational logic layers of reasoning blocks because the layers in a reasoning block are responsible for drawing all conclusions.

However, in agent chips, the combinational logic layers are implemented as pure combinational logic circuits, and thus each reasoning block can compute all of the conclusions almost instantaneously. For instance, our implementation of an agent chip consisting of 16,000 rules on a *Xilinx<sup>TM</sup>* Spartan-3 FPGA chip (currently each chip costs less than US\$9) can operate in 140 Mhz clock frequency (i.e., less than  $8 \times 10^{-9}$  seconds for each cycle) whereas conventional approaches based on more expensive CPUs (e.g., 3Ghz Pentium 4 CPU) require more than a minute for each cycle.

### 3 Separation of Temporal Evaluation

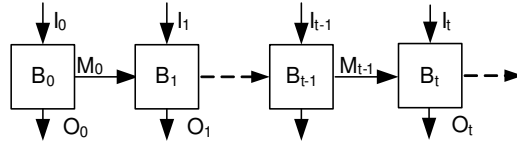
One important feature of the reasoning block is that its memory unit provides very efficient temporal reasoning capability by forming a closed-loop controller similarly to industrial automation controllers such as PLCs (Programmable Logic Controllers).

*Example 1.* As an example, suppose that when a cleaning robot has detected a signal on sensor  $sD$ , it needs to memorize the state that the area is dirty (denoted as  $d$ ) even if  $sD$  is not true anymore until it detects another signal on sensor  $sC$ , which tells that the area is clean. If the agent concludes that the area is dirty, it turns on the vacuuming unit ( $vc$ ). In most logical formalisms, this cannot be modelled without introducing the notion of time in the language of the knowledge base resulting in high computational complexity as shown below:

$$\{sD \rightarrow d, (P_1d) \rightarrow d, sC \rightarrow \neg d, d \rightarrow vc\}$$

where all of the rules are defeasible rules (e.g., defeasible rules in Defeasible Logic [8]) and  $P_n$  is a bounded temporal operator denoting ‘‘Previous  $n$  time moments’’ for  $n \geq 0$ . Thus,  $P_1d$  denotes that  $d$  was true just one time moment ago. Since the rules are defeasible rules, when both  $sD$  and  $sC$  are true, it does not introduce inconsistency, but rather it just means that neither  $d$  nor  $\neg d$  can be proved. We should note that, in general, reasoning under temporal logic is EXPSpace-complete [1]. In a reasoning block, however, we can remove the notion of time in the object language by converting the formulas containing temporal operators into simple propositions referring to the evaluation results of the formulas in the memory unit of the reasoning block as shown below:

$$\{sD \rightarrow d, d(P', 1) \rightarrow d, sC \rightarrow \neg d, d \rightarrow vc\}$$



**Fig. 3.** Memory captures the previous state with which an agent can reason about time in a very simple way.

where  $d('P', 1)$  is just a *proposition* denoting the previous conclusion of  $d$ : it denotes that the agent knows that  $d$  was true in the previous round of reasoning. Here,  $d('P', 1)$  refers to a state in the memory of a reasoning block: it is an input to the knowledge base of the deduction unit. That is, when the agent concludes that the area is dirty, the truth value of  $d$  is captured in the memory as  $d('P', 1)$ .

Figure 3 illustrates how an agent's belief state (the state of the reasoning block representing the belief generator of an agent block) changes over time based on the current input state and the immediate previous state captured in memory  $M_{t-1}$ . This example demonstrates that the temporal operator ( $P_n$ ) can be efficiently implemented in reasoning blocks. Similarly to PLCs, other bounded temporal operators of Metric-Temporal Logic [6] can also be efficiently implemented with the memory unit.

Therefore, symbolically, the object language for the reasoning blocks can be considered purely propositional and atemporal because the evaluation of the formulas with temporal operators is separated from the main reasoning process. Consequently, we can now study the object language of agents' knowledge bases and evaluation of temporal formulas separately.

## 4 Formal Temporal Specification Method of Agent Chips

In this section, we discuss a formal specification method of the memory units of reasoning blocks defined in Section 2. We also briefly discuss formal verification issues of agent chips. First, we give a brief introduction to Metric-Temporal Logic (MTL) which provides convenient operators for specifying quantitative requirements, such as bounded response. We refer the reader to [6,7] for more thorough treatments of MTL. Linear Temporal Logic (LTL) in comparison provides only the temporal operators for specifying qualitative requirements on execution sequences.

### 4.1 Introduction of Metric-Temporal Logic (MTL)

MTL extends Linear Temporal Logic (LTL) with the following bounded temporal operators  $\square_{\#d}q$ ,  $\diamond_{\#d}q$ , and  $pU_{\#d}q$  where  $\# \in \{<, \leq\}$  and  $d \in Nat$ . For example,  $\diamond_{\leq d}$  expresses the notion "eventually within time  $d$ ." Given a finite set  $P$  of propositions, the

set of *MTL formulas* is inductively defined as follows:

$$\phi := true | false | p | \neg\phi | \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2 | \Box\phi | \Diamond\phi | X\phi | P\phi | \\ X_d\phi | P_d\phi | \Box_{\#d}\phi | \Diamond_{\#d}\phi | \phi_1 U_{\#d}\phi_2$$

where  $p \in P$ ,  $\Box$  denotes ‘always’,  $\Diamond$  denotes ‘possibly’,  $X$  denotes ‘next’,  $P$  denotes ‘previous’, and  $U$  denotes ‘until’. The time domain  $T$  of MTL is the set of natural numbers  $Nat$ . An interpretation of an MTL formula is a *trace*  $h : Nat \rightarrow 2^P$  that maps to each time position  $i \in T$  the set of propositions that hold at that position. We define a distance function to denote the time elapsed between time positions  $i$  and  $j$  in a trace:

$$dist(i, j) = |i - j| \times \delta$$

where  $\delta$  is a time unit. In the case of the reasoning block with clock frequency  $f$ ,  $\delta$  is  $1/f$  seconds.

The semantics of the temporal operators is defined over system traces and the distance function as follows:

1.  $(h, i) \models p$  iff  $p \in h(i)$ .
2.  $(h, i) \models Xp$  iff  $(h, i + 1) \models p$ .
3.  $(h, i) \models X_dp$  iff  $(h, i + d) \models p$ .
4.  $(h, i) \models Pp$  iff  $(h, i - 1) \models p$ .
5.  $(h, i) \models P_dp$  iff  $(h, i - d) \models p$ .
6.  $(h, i) \models \Box p$  iff  $\forall j \geq i. (h, j) \models p$ .
7.  $(h, i) \models \Diamond p$  iff  $\exists j \geq i. (h, j) \models p$ .
8.  $(h, i) \models \Box_{\#d} p$  iff  $\forall j \geq i, (h, j) \models p$  and  $dist(i, j) \# d$ .
9.  $(h, i) \models \Diamond_{\#d} p$  iff  $\exists j \geq i, (h, j) \models p$  and  $dist(i, j) \# d$ .
10.  $(h, i) \models pU_{\#d}q$  iff  $\exists j \geq i, (h, j) \models q$  and  $dist(i, j) \# d$  and  $\forall i \leq k < j, (h, k) \models p$ .

An *MTL theory* is a set of MTL formulas.

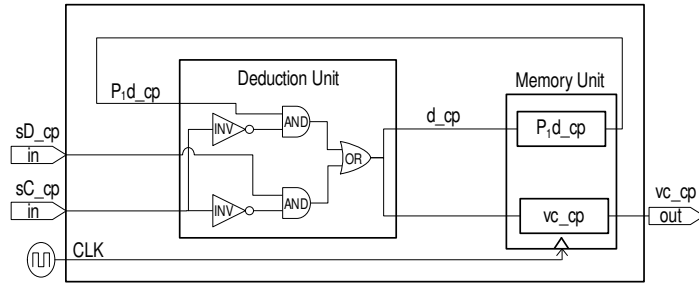
## 4.2 Mapping Knowledge Bases into MTL

Let  $L$  denote an atemporal logical language for a reasoning block. We now define special propositions for  $L$  to allow for the specification of temporal operations. We consider the following temporal operators for propositions  $q$  and  $\phi$  in  $L$  in the antecedents of rules:

1.  $q('P', d)$  meaning  $P_dq$ ;
2.  $\phi('X', d)$  meaning  $X_d\phi$ ;
3.  $\phi('G', \#, d)$  meaning  $\Box_{\#d}\phi$ ;
4.  $\phi('F', \#, d)$  meaning  $\Diamond_{\#d}\phi$ ;
5.  $\phi('U', p', \#, d)$  meaning  $pU_{\#d}\phi$ .

For example,  $x('P', 1) \rightarrow y$  means that if  $x$  was true just before,  $y$  is usually true.

We can also consider the following temporal operators in the consequents of rules for future-time events, but they will be interpreted as actions setting future-time events:



**Fig. 4.** An implementation of the temporal operator  $P_1$  for the cleaning robot in Example 1. The signal labels  $sD\_cp$ ,  $sC\_cp$ ,  $vc\_cp$ ,  $d\_cp$ , and  $P_1d\_cp$  represent the positive literals  $sD$ ,  $sC$ ,  $vc$ ,  $d$ , and  $P_1d$ , respectively. The value 1 of these signals means that the corresponding positive literals are provable. The value 0 of the signals means that the provability of these positive literals is unknown.

1.  $q(Set, 'X', d)$  meaning that the system must reset a timer so that  $X_dq$  is true, i.e.,  $q$  is true in the next  $d$ -th time units;
2.  $q(Set, 'G', '#', d)$  meaning that the system must reset a timer so that  $\square_{\#d}q$  is true;
3.  $q(Set, 'F', '#', d)$  meaning that the system must reset the a timer so that  $\diamond_{\#d}q$  is true;
4.  $q(Set, 'U', 'p', '#', d)$  meaning that the system must reset a timer so that  $pU_{\#d}q$  is true.

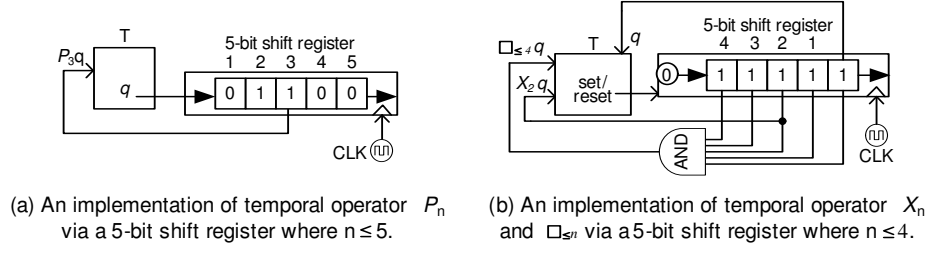
For example,  $x \rightarrow y(Set, 'X', 1)$  means that if  $x$  is true now, we usually set a timer so that  $y$  is (usually) true in the next time moment.

Let us now suppose  $L$  is a logical language that can be mapped into an equivalent classical propositional language: there exists a mapping  $\Pi$  of  $L$  into a propositional language. Defeasible Logic [12], Layered Argumentation System ([12] and [11]), and Default Logic ([2]) are such languages.

Then, mapping a theory  $T$  of  $L$  into an MTL theory is straightforward. We obtain the corresponding MTL theory  $MT(\Pi(T))$  of  $\Pi(T)$  by replacing the temporal information of each proposition  $p$  in  $\Pi(T)$  with the temporal operators as follows :

1. Replace all propositions  $a('X', d)$  in  $\Pi(T)$  with  $X_d a$ ;
2. Replace all propositions  $a('P', d)$  in  $\Pi(T)$  with  $P_d a$ ;
3. Replace all propositions  $a('G', '#', d)$  in  $\Pi(T)$  with  $\square_{\#d} a$ ;
4. Replace all propositions  $a('F', '#', d)$  in  $\Pi(T)$  with  $\diamond_{\#d} a$ ;
5. Replace all propositions  $a('U', 'b', '#', d)$  in  $\Pi(T)$  with  $bU_{\#d} a$ .

The resulting MTL theory formally specifies how the memory unit should be implemented in hardware. As shown in Section 3 and Figure 4, the temporal operator  $P_1$  can be directly implemented in reasoning blocks by just memorizing immediate previous conclusions. Figure 4 shows an implementation of operator  $P_1$  for  $d('P', 1)$  (denoted as ' $P_1d$ ' in the figure) as a single-bit register in the memory unit of the cleaning robot. The digital circuit description in the deduction unit is automatically generated from the



**Fig. 5.** Implementations of temporal operators.

defeasible rules in Example 1 using a compiler that we have developed for Layered Argumentation System [11]. It is easy to verify that the combinational logic circuit performs the required nonmonotonic reasoning as described in the example.

Similarly, for  $0 < n \leq N$ ,  $P_n$  can be implemented using an  $N$ -bit shift register:  $(h, i - n) \vdash P_n q$  iff the  $n$ -th bit of the shift register is value 1. A shift register shifts its bits in one direction (e.g., from the least significant bit (LSB) to the most significant bit (MSB)) at every clock cycle. Figure 5 illustrates an implementation of operator  $P_n$ . It is straightforward to show that this implementation is sound and complete for  $P_n$ .

However, implementation of temporal formulas referring to the future time events is more complex. In this paper, however, we restrict our language to provide simple, yet practical, implementations of some future-time temporal operators as follows: we restrict our language so that the propositions  $\phi$  in the future-time temporal formulas ( $X_d \phi$ ,  $\square_{\#d} \phi$ ,  $\diamond_{\#d} \phi$ , and  $pU_{\#d} \phi$ ) in the antecedents of rules, are not allowed in the consequents of rules. That is, let us consider the special case that future events are determined by the input to the system or timers in the memory unit.

Then, similarly to  $P_n$ , for  $0 < n < N$ ,  $X_n q$  can be implemented using an  $(N + 1)$ -bit shift register which always receive value 0 in its most significant bit. This can be used to test an event  $q$  that will occur within  $n$  time units. In other words, the event can be considered as an internal alarm clock that was set by the agent through the agent's action. Setting the  $n$ -th bit of the shift register with  $q(Set, X', n)$  will make  $X_n q$  true.  $\square_{\leq n} q$  can be represented just as a conjunction of all of the bits in the shift register. Consequently, setting all of the bits in the shift register with  $q(G', \leq', n)$  will make  $\square_{\leq n} q$  true. Figure 5 illustrates implementations of these temporal operators.

Other useful temporal operators can also be similarly implemented. This approach (using internal timers) is very similar to the implementations used in PLCs (Programmable Logic Controllers), the popular industrial controllers. Alternatively, an MTL theory or a subset of it can be directly used to synthesize real-time systems [9]. In this case, then, the temporal operators in the consequents of the rules should be interpreted as normal temporal formulas and mapped into proper MTL temporal formulas exactly like the temporal operators in the antecedents of rules. For example, instead of using  $q(Set, X', d)$ , we should use  $q(X', d)$  and map it to  $X_d q'$ .

### 4.3 Formal Verification Issues of Agent Chip Properties

Once we have the corresponding MTL theory,  $MT(\Pi(T))$ , of a knowledge base,  $T$ , we can formally test some system properties over  $MT(\Pi(T))$  with existing proof theories of MTL. One interesting system property is testing for conditionals. For example, given an agent specification  $A$ , the following system properties might be interesting:

1.  $A \vdash \Box(a \Rightarrow b)$  which says that, for all of the entire traces of agent  $A$ ,  $b$  follows from  $a$ .
2.  $A \vdash a \Rightarrow \Box(b \Rightarrow c)$  which says that, for all of the entire traces of agent  $A$  in which  $a$  is true, the agent has the property of  $A \vdash \Box(b \Rightarrow c)$ .

We have used a different arrow ' $\Rightarrow$ ' to denote that the formula is not an MTL formula, but a conditional in some conditional logic. For instance,  $A \vdash \Box(a \Rightarrow b)$  is not true for an agent that concludes  $b$  always even if  $a$  is false. The property test is asking whether there is an inference relation between  $a$  and  $b$  such that  $b$  follows from  $a$  in every possible traces of the agent.

## 5 Conclusion

The main result of this paper has been the solution to the problem of a real-time temporal reasoning that can be realized in small electronic chips. The key idea has been the separation of temporal evaluation from the main reasoning process and the mapping for generating formal specifications for temporal reasoning circuits. This mapping is a simple direct mapping and yet produces compact and scalable hardware specification. It is scalable, since it is mapped into MTL which is widely used in the formal specification of digital circuits (e.g., [9]).

In this paper, we have not prescribed any particular logical language for our method. The reason is because our method can be applied to most of nonmonotonic logical languages extended with *bounded temporal operators* provided that they can be mapped into a classical propositional logic.

Some important issues, which we have not discussed in detail in this paper, because of the limited space, are the evaluation of future temporal formulas ( $X_dq$ ,  $\Box_{\#d}q$ ,  $\Diamond_{\#d}a$ , and  $pU_{\#d}a$ ) and the interpretation of temporal operators in the consequents of defeasible rules. For the former we have shown that a subset of language can be implemented as hardware straightforwardly. For the latter, we considered the temporal formulas in the consequents as actions for setting or resetting timers in the memory unit. PLCs (Programmable Logic Controllers), which are popular industrial automation controllers, have adopted this interpretation. In this case, they cannot refer to the past state, i.e., we cannot change what has happened and the facts of what we have concluded in the past. Of course conflicting actions must be resolved. Unlike PLCs, however, many nonmonotonic logics (e.g., Defeasible Logic [8], Defeasible Argumentation System [4], Layered Argumentation System [11]) can resolve conflicts between competing rules.

The most closely related work to this paper is "situated automata" [10], in which Kaelbling and Rosenchein [10] have developed a compiler that directly synthesizes digital circuits from a knowledge based model [3] of an agent's environment. Their

language is based on a weak temporal Horn-clause language with the addition of *init* and *next* operators. While this work clearly highlighted the realization of epistemic theories of agents, they provide no formal semantics of the language. Their approach is also a model based top-down approach that the compiler generates a target system from a model of the world. Thus, the synthesis is not always guaranteed and resulting systems can be sub-optimal because in most cases the required solution is usually a small subset of the solutions of the model.

The results are important, since we can combine our method and atemporal logical languages to obtain temporal languages, which can be realized in hardware for real-time reasoning, such as temporal argumentation systems from [11] and [4], temporal Default Logics from [2], and temporal Defeasible Logics from [8]. Of course the reasoning blocks for these temporal languages can also be implemented in software. In fact, our method can be used to provide a very efficient implementation of the temporal Defeasible Logic detailed in [5].

## References

1. Rajeev Alur and Thomas A. Henzinger. Real-time logics: complexity and expressiveness. Technical report, Stanford, CA, USA, 1990.
2. Rachel Ben-Eliyahu and Rina Dechter. Default reasoning using classical logic. *Artif. Intell.*, 84(1-2):113–150, 1996.
3. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.
4. Guido Governatori, Michael J. Maher, Grigoris Antoniou, and David Billington. Argumentation semantics for defeasible logics. *Journal of Logic and Computation*, 14(5):675–702, 2004.
5. Guido Governatori, Antonino Rotolo, and Giovanni Sartor. Temporalised normative positions in defeasible logic. In *Proceedings of the 10th International Conference on Artificial Intelligence and Law*, pages 25–34. ACM Press, 2005.
6. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.
7. Ron Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.
8. M. J. Maher and G. Governatori. A semantic decomposition of defeasible logics. In *AAAI '99*, pages 299–305, 1999.
9. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. POPL '89*, pages 179–190, New York, NY, USA, 1989. ACM Press.
10. Stanley J. Rosenschein and Leslie Pack Kaelbling. A situated view of representation and control. *Artif. Intell.*, 73(1-2):149–173, 1995.
11. Insu Song and Guido Governatori. A compact argumentation system for agent system specification. In *Proc. STAIRS'06*. In print: available at <http://eprint.uq.edu.au/archive/00004138/01/InsuGuidoSTAIRS.pdf>. IOS Press, 2006.
12. Insu Song and Guido Governatori. Designing agent chips. In Peter Stone and Gerhard Weiss, editors, *5th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1311–1313. ACM Press, 10–12 May 2006.