# Automatic Synthesis of Reactive Agents

Insu Song
School of Business and IT
James Cook University Australia
Singapore
insu.song@jcu.edu.sg

Guido Governatori
Education in Queensland Research Laboratory
NICTA (National ICT Australia)
Australia
guido.governatori@nicta.uq.edu.au

Joachim Diederich
School of Business and IT
James Cook University Australia
Singapore
joachim.diederich@jcu.edu.sg

***Abstract*—This paper introduces a new approach to designing smart control chips that enables automatic synthesis of real-time control systems from agent specifications. An agent specification is compiled into a hardware description format, such as RTL-VHDL (Register Transfer Level–VLSI Hardware Description Language) or RTL Verilog, which is synthesized using computer-assisted tools to develop ASIC masks or FPGA configurations. A rule-based specification language called Layered Argumentation System (LAS) is defined and a sound and complete mapping to Verilog is developed. LAS combines fuzzy reasoning and non-monotonic reasoning. This enables chip designers to capture commonsense knowledge and concepts having varying degrees of confidence collaboratively and incrementally.**

***Index Terms*—control systems, reactive systems, circuit synthesis, agent programming, fuzzy controller, logic controller**

## I. Introduction

Over the past years, we have witnessed massive production of small electronic consumer devices, such as cell phones, set-top boxes, home network devices, and MP3 players, just to name a few. Device sizes get smaller, and behaviors get more and more complex. In order to survive in the current competitive market, vendors now must scramble to offer more variety of innovative products faster than ever before because most of modern consumer electronic devices have comparatively low run rates and/or short market windows [1].

One solution of reducing the development cost and time is providing a more expressive and intuitive specification language for describing the behaviors of products. One promising specification language is an agent architecture comprising of logical theories because logic is close to our natural languages and it has the ability to represent varieties of domains and problems that we are interested in; and many agent models, such as belief-desire-intention (BDI) model of rational agency [2], subsumption architecture [3], and BOID architecture [4], are suitable for specifying complex autonomous behaviors.

Classical logics and traditional specification languages for specifying electronic chips, such as C, Verilog, VHDL (VLSI Hardware Description Language), MTL (Metric-Temporal Logic), are rigid in the sense that whenever minor modifications have to be made to a system specification, the whole system specification has to be tested and recompiled to resolve any inconsistency and conflicts introduced by the modifications. This is problematic when multiple parties are involved in specifying the same system because of conflicting specifications introduced by different parties. It is also difficult to perform incremental development or partial update.

Nonmonotonic logics [5], [6], [7], [8], [9] have a unique feature that can resolve conflicts in logical theories. This feature is essential for collaborative and incremental system development. Many interpreted languages, such as Visual Basic, MATLAB, Perl, and Python, can provide a runtime environment that does not require a lengthy compilation step.

However, interpreted languages are computationally inefficient. Most logical languages (including nonmonotonic logic) are computationally too complex to be suitable for specifying real-time systems and building real-time systems using the specifications. Despite of further efforts (e.g., use of logic in robot control [10], layered approaches [10], [11], argumentation systems [11], [12], [13], [14], [15], possibilistic logics [16], [17] for expressing varying degrees of belief) to overcome limitations of logic, application of logic is still limited, in particular for building real-time systems, because of their computational inefficiency and limitations to satisfy the following essential features for specifying smart real-time control system: (a) expressing behaviors, (b) expressing varying degrees of confidence and task specificity, and (c) robustness for mission critical applications as they require sophisticated theorem provers running on a high powered CPU.

The paper is structured as follows. In the next section, we illustrate an example application of our method and a performance evaluation result. In Section III, we formally define a rule-based language called Layered Argumentation System (LAS) that is used to specify behaviors and knowledge bases of agents. After that, we develop a series of mappings of LAS to logic programs (Section IV), Boolean equations (Section V), and finally to Verilog HDL (Section VI), which can be synthesized into agent chips. Then, in Section VII we compare our approach with other executable agent-specifications and conclude with some remarks in Section VIII.

## II. Application Example and Evaluation Result

Figure 1 shows one possible practical application of our method. It shows a video-surveillance agent-chip that can be synthesized with our method. The chip has an agent block and two other specialized hardware blocks which are based on commercially available IP (Intellectual property) cores. Suppose the agent chip receives ten fames per second from the video camera observing a busy airport lounge; the segmentation unit segments each image into N segments, each of which represents a person in the image. The feature

11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010
© 2010, IEEE Press.

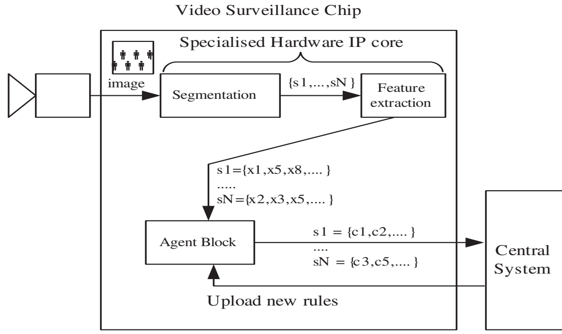


Fig. 1. Video surveillance agent chip: an example of possible application of agent chip.

extraction unit extracts a set of features from these segments. After this, a segment is represented as a set of propositions $s_i = \{x_{i1}, ..., x_{im}\}$, each of which represents a feature. Some examples of possible features are shown below:

1) Environment states: morning, afternoon, evening, cool, overcast, rain.
2) Clothing features: long coat, two pieces, short, red cloth, hat, dark hat.
3) Size: child, adult, tall, short, average.
4) Accessories: sunglasses, handbag, backpack.

The agent-chip, then, classifies the segments and assigns each segment a set of class labels based on the rules provided by the central system. Some examples of possible rules describing suspects are shown below:

Layer 1 rules: $R_1 = \{$sunglasses,evening$\rightarrow$suspicious,
child,blue2piece$\rightarrow$lostChild1,
veryTall,longCoat,redCoat,longHair$\rightarrow$criminal1$\}$

Layer 2 rules: $R_2 = \{$helmet,indoor$\rightarrow$suspicious$\}$

The arrows herein represent defeasible inferences. For instance, $c \rightarrow v$ is read as "if $c$ is true, then usually $v$ is true". The layers represent relative confidences such that layer-$n$ conclusions are more confident than layer-$(n+1)$ conclusions. A segment with one or more class labels is then sent to the central system to alert the authorities about the possible subjects they are looking for. If we have an average of 50 people in each image and we have 1000 subjects, each of which is described by 20 rules, then the total number of rules required is 20,000, and the reasoning block must classify each segment within 1/500 seconds.

In order to test whether a reasoning block implemented on a reasonably affordable FPGAs can handle this kind of workload, we randomly generated an LAS theory consisting of 16,000 rules and 4000 literals (including 1000 propositions representing all randomly generated features). This theory is compiled into RTL-Verilog and synthesized into *Xilinx*$^{TM}$ Spartan-3 FPGA configuration file using *Xilinx*$^{TM}$ ISE. The result of this evaluation is as follows: an implementation of this theory on a *Xilinx*$^{TM}$ Spartan-3 FPGA chip (currently each chip costs less than US$9 and consumes about 1W of energy) can classify each frame within $8 \times 10^{-9}$ seconds (i.e., it can classify 140 million segments every second). However, the compilation of the entire LAS theory into an FPGA configuration takes several hours on a PC with Pentium-4 3GHz CPU and 500M bytes of memory. That is, incremental updates are a very important feature for any agents that require frequent updates.

In order to compare this with some conventional reasoning algorithms, we also derived all of the conclusions of the same theory with an efficient forward chaining algorithm implemented in Python 2.4 on a PC with Pentium-4 3GHz CPU and 2G bytes of memory. This approach in comparison took more than 20mins to generate all of the conclusions. We should note that an embedded system with similarly priced ($9) CPUs might take even longer to process each frame. That is, not only agent systems can be built incrementally with our method, systems generated with our approach perform several million times faster than conventional computing methods.

## III. Definition of Layered Argumentation System (LAS)

In this section, we formally define a rule-based language called Layered Argumentation System (LAS). LAS is a logical language that can capture complex behaviors and varying degrees of agents' belief. LAS also offers improved Fuzzy-logic reasoning features [18]. A precursor of this language was used in [19] to decompose system behaviors similarly to Brook's subsumption architecture. Later in [20], it is given argumentation semantics and comparisons with other existing layered logics and hierarchical approaches.

### A. Formal Definition

As the underlying logical language, we start with essentially propositional inference rules: $r : L \rightarrow l$ where $r$ is a unique label, $L$ is a finite set of literals, and $l$ is a literal. If $l$ is a literal, $\sim l$ is its complement: if $l$ is a positive literal $p$, $\sim l$ is $\neg p$; if $l$ is a negative literal $\neg p$, $\sim l$ is $p$.

An LAS theory is a structure $T = (R, N)$ where $R = (R_1, ..., R_n, ..., R_N)$ is a sequence of finite sets of rules where each $R_n$ ($1 \leq n \leq N$) is a finite set of layer-$n$ rules and $N$ is the number of layers and $n$ is a layer index.

All rules in one and the same layer have the same degree of confidence. We stipulate that layer-$n$ rules are more confident rules than layer-$(n+1)$ rules. This is because, when we build a system, we tend to add less task specific rules (i.e., more urgent behaviors) first, such as rules for avoiding objects in a corridor, and then gradually add more task specific rules, such as rules for finding a goal location. Importantly, LAS represents only the relative degree of confidence between layers in order to avoid the need for acquiring the actual confidence-degree value of every rule.

We will sometimes use layer index $n$ (just) to index a certain imaginary confidence-degree value (*the degree of confidence*) of layer-$n$: the degree of confidence of layer index $n$ is higher than the degree of confidence of layer index $(n + 1)$. That is, layer-1 rules are the most confident rules and thus have the highest degree of confidence.

As for the semantics of the language, we modify the argumentation framework given in [21] to introduce layers into the argumentation system. Argumentation systems usually contain the following basic elements: an underlying logical language, and the definitions of: *argument*, *conflict between arguments*, and the *status of arguments*.

As usual, *arguments* are defined to be proof trees. An argument for a literal $p$ based on a set of rules $R$ is a (possibly infinite) tree with nodes labelled by literals such that the root is labelled by $p$ and for every node with label $h$:

1) If $b_1, ...., b_i$ label the children of $h$ then there is a rule in $R$ with body $b_1, ..., b_i$ and head $h$.
2) The arcs in a proof tree are labelled by the rules used to obtain them.

A literal labelling a node of an argument $A$ is called a conclusion of $A$. However, when we refer to the conclusion of an argument, we refer to the literal labelling the root of the argument. The set $Args_n$ of layer-$n$ *defeasible arguments* is the set of arguments based on a set of rules $R_1 \cup ... \cup R_n$. All of arguments in $Args_n$ have the same layer index $n$: they have the same degree of confidence as layer-$n$. We define $Args_0$ to be the empty set. $Args_n$ is analogous to the defeasible arguments in [21]. The set $SArgs_n$ of layer-$n$ *strict arguments* is a subset of arguments formed based on a set of rules $R_1 \cup ... \cup R_{n-1}$. All of arguments in $SArgs_n$ have the same layer index $n-1$: they have the same degree of confidence as layer-$(n-1)$. We define $SArgs_0$ to be the empty set. $SArgs_n$ is analogous to the strict arguments formed of the strict rules in [21]. That is, $SArgs_n$ is thought of as arguments that have already been demonstrated to be justified at layer-$(n-1)$.

We now introduce a set of usual notions for argumentation system: *attack*, *support*, and *undercut* at layer-$n$. A layer-$n$ argument $A$ *attacks* a layer-$n$ argument $B$ if the conclusion $p$ of $A$ is the complement of a conclusion $q$ of $B$, and that conclusion of $B$ is not part of a strict subargument $C$ of $B$ (i.e., $C \notin SArgs_n$). A layer-$n$ defeasible argument $A$ is *supported* by a set of layer-$n$ arguments $S \subseteq Args_n \cup SArgs_n$ if every proper sub-argument of $A$ is in $S$. A layer-$n$ defeasible argument $A$ is *undercut* by a set of layer-$n$ arguments $S \subseteq Args_n \cup SArgs_n$ if $S$ supports a layer-$n$ argument $B$ attacking a proper non-strict sub-argument $C$ of $A$ (i.e., $C \notin SArgs_n$).

*Example 1:* Consider an example LAS theory: $R_1 = \{\rightarrow d,\ r_1 : d \rightarrow c,\ r_2 : w \rightarrow \neg v, \}$ and $R_2 = \{\rightarrow w, r_3 : c \rightarrow v\}$. Now we consider the arguments below:

```
A    B   | C    D     E    F    G
• d  • c | • w  • ~v  • d  • c  • v
     |   |      |          |    |
     • d |      • w        • d  • c
         |                      |
         |                      • d
Layer 1  |        Layer 2
```

$A$ and $B$ are layer-1 defeasible arguments for $d$ and $c$, respectively, and thus they are also a layer-2 defeasible arguments ($E$,$F$) because layer-1 arguments are more confident than layer-2 arguments. $C$, $D$, and $G$ are layer-2 defeasible argument for $w$, $\sim v$, and $v$, respectively. $F$ is a sub-argument of $G$. We should note that $D$ is a layer-2 defeasible argument, because layer-1 rule $r_2$ is subsumed to layer-2. That is, layer-2 evidence for $w$ and layer-1 rule $r_2$ are combined to produce a layer-2 argument. In addition, since there is no layer-1 evidence for $w$, there is no layer-1 argument for $\sim v$. We should note that this feature of combining evidence and rules of different layers is not possible in existing layered approaches such as [22]. Unlike Fuzzy Logic approaches [16], [17], the inference process is very simple and more intuitive.

The heart of an argumentation semantics is the notion of an *acceptable argument*. Based on this concept it is possible to define *justified arguments* and *justified conclusions*, conclusions that may be drawn even taking conflicts into account. Given a layer-$n$ argument $A$ and a set $S$ of layer-$n$ arguments (to be thought of as arguments that have already been demonstrated to be justified), we assume the existence of the concept: $A$ is *acceptable w.r.t.* $S$.

In this paper, we use a modified notion of a argumentation semantics *acceptable* given in [21] that captures defeasible provability in Defeasible Logic (DL) [6] with ambiguity blocking. We refer the reader to [23], [24], [25] for a detailed discussion on ambiguity blocking and the difference between ambiguity blocking and ambiguity propagating non-monotonic logics.

*Definition 1:* A layer-$n$ argument $A$ for $p$ is *acceptable* w.r.t a set of layer-$n$ arguments $S$ if $A$ is finite, and
1. $A$ is a layer-$n$ strict argument, or
2. every layer-$n$ argument attacking $A$ is undercut by $S$.

Based on this concept we proceed to define justified arguments and justified literals.

*Definition 2:* Let $T = (R, N)$ be an LAS theory. We define $J_i^n$ as follows:
(1) $JArgs^0 = \emptyset$;
(2) $SArgs_n = JArgs^{n-1}$;
(3) $J_0^n = \emptyset$;
(4) $J_{i+1}^n = \{a \in Args_n \cup SArgs_n \mid a$ is acceptable w.r.t. $J_i^n\}$;
(5) $JArgs^n = \cup_{i=1}^{\infty} J_i^n$ is the set of justified layer-$n$ arguments of an LAS theory $T$.
We can now give the definition of a finite set of literals denoting the set of layer-$n$ conclusions $C_n$ of $T$ as the set of the conclusions of the arguments in $JArgs^n$. A literal $p$ is layer-$n$ justified if it is the conclusion of an argument in $JArgs^n$. Since arguments cannot attack stronger arguments, conclusions made in lower layers are subsumed to upper layers.

*Proposition 1:* (Conclusion Subsumption) Let $T = (R, N)$ be an LAS theory. Then, the following holds for $1 \leq n \leq N$

$$C_{n-1} \subseteq C_n$$

The process of building justified arguments is incremental, that is, first we build all layer-1 arguments, and then layer-2, so on towards layer-$N$. Figure 2 illustrates the process of generating justified arguments incrementally. Thus, when we develop an implementation of LAS, it is convenient to have a notion referring to the argumentation of each individual layer.

*Definition 3:* Let $T = (R, N)$ be an LAS theory. Let $JArgs^n$ be the set of justified layer-$n$ arguments of $T$. The

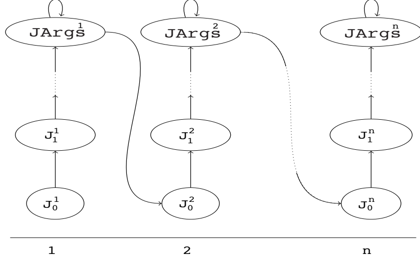


Fig. 2. Incremental generation of justified arguments starting from layer-1.

layer-$n$ argumentation of $T$ is defined inductively as follows:
1. $T_0 = (\emptyset, \emptyset)$.
2. $T_1 = (\emptyset, \mathbb{R}_1)$ is layer-1 argumentation where $\mathbb{R}_1 = R_1$;
3. $T_n = (JArgs^{n-1}, \mathbb{R}_n)$ is layer-$n$ argumentation where $\mathbb{R}_n = R_1 \cup ... \cup R_n$;

We should note that, by the definition, layer-$n$ subsumes (includes) layer-($n-1$) rules. That is, unlike other layered or hierarchical approaches (e.g., [10], [11], [22]) lower layer rules (more confident rules) are reused in higher layers. This feature is important because facts and rules with different degrees of confidence can interact similarly to Possibilistic Logic approaches. [16], [17] and Fuzzy-logic approaches.

## IV. Meta-Encoding of LAS

We now map LAS into propositional logic programs. This step is necessary to map LAS into Boolean equations in Section V.

First, we define two literal encoding functions that encode literals in an LAS theory to previously unused positive literals. Let $q$ be a literal and $n$ a positive integer. Then, these functions are defined below:

$$Supp(q,n) = \begin{cases} p_n^{+s} & \text{if } q \text{ is a positive literal } p. \\ p_n^{-s} & \text{if } q \text{ is a negative literal } \neg p. \end{cases}$$

$$Con(q,n) = \begin{cases} p_n^{+} & \text{if } q \text{ is a positive literal } p. \\ p_n^{-} & \text{if } q \text{ is a negative literal } \neg p. \end{cases}$$

$Supp(q,n)$ denotes a support of $q$ at layer-$n$ and $Con(q,n)$ denotes a conclusion $q$ at layer-$n$. $Supp(q,n)$ corresponds to supported$_n(q)$. $Con(q,n)$ corresponds to conclusion$_n(q)$. With these functions, we now define the meta-encoding schema.

Let $T = (R, N)$ be an LAS theory. Let $T_n = (JArgs^{n-1}, \mathbb{R}_n)$ be layer-$n$ argumentation of $T$. Let $SL_n = \{C(r) | r \in \mathbb{R}_n\}$ be the set of all supportive literals in layer-$n$ of $T$. The meta-encoding $G(\mathbb{R}_n)$ of $\mathbb{R}_n$ is obtained according to the following guidelines for each layer-$n$ ($1 \leq n \leq N$):

**G1**: For each $q \in SL_n$, add

$$Con(q,n) \text{ :- } Con(q, n-1)$$

**G2**: For each $q \in SL_n$, add

$$Con(q,n) \text{ :- } Supp(q,n), not Supp(\sim q, n), not Con(\sim q, n-1)$$

**G3**: For each $r \in \mathbb{R}_n$, add

$$Supp(C(r), n) \text{ :- } \bigwedge_{q \in A(r)} Con(q, n)$$

We should note that $G(\mathbb{R}_n)$ is a propositional normal logic program.

## V. Mapping LAS into Boolean-Equations

In this section we use the classical Clark *completion* [26] of a logic program to map a meta-program of an LAS theory to a set of Boolean equations. In [26], Clark gave semantics of *normal programs*, which are logic programs with negation as failure. We used symbol "not" to denote negation as failure in the meta-programs defined in the previous sections. That is, negated atoms (denoting negation as failure) are allowed in a rule's body of normal programs. Clark showed that how each normal program $\Pi$ can be associated with a first-order theory $CDB(\Pi)$, called its completion or completed database, by converting all clauses to "iff" assertions.

For the sake of completeness, we first describe how a Clark completion of a propositional normal logic program is obtained [27]. We consider only propositional normal logic programs, that is, a set of rules of the form

$$q \leftarrow a_1, ..., a_n, not\ b_1, ...not\ b_m \tag{1}$$

where $q$, $a_1$,...,$a_n$, and $b_1$,...,$b_m$ are atoms. Then, given a propositional logic program $P$, $CDB(P)$ is obtained in two steps:

**Step1**: Replace not with $\neg$: replace each rule of the form 1 with the rule

$$q \leftarrow a_1, ..., a_n, \neg\ b_1, ..., \neg\ b_m$$

**Step2**: For all rules $r : q \leftarrow Body_r$ in the program with head a symbol $q$, where $Body_r$ is the body of the rule, add

$$q \leftrightarrow \bigvee_{\forall r \in P \text{ with head } q} (Body_r)$$

If there are no rules with head $q$ in $P$, add $\neg q$.

We now define Boolean equation mapping of an LAS theory. Let $T = (R, N)$ be an LAS theory, $G(T)$ be the corresponding meta-encoding (a propositional normal logic program) of $T$, and $G(T_n)$ be meta-encoding of a layer-$n$ argumentation $T_n$ of $T$, where $G(T)$ and $G(T_n)$ are defined in Section IV. Let $SL_n = \{C(r) | r \in \mathbb{R}_n\}$ be the set of all supportive literals in layer-$n$ of $T$, where $\mathbb{R}_n$ is the set of layer-$n$ rules of $T$.

We define the Boolean-equation mapping $B(G(T))$ of $G(T)$ be the Clark completion of $G(T)$, which is obtained by adding Boolean equations according to the following guidelines for each layer-$n$ (we used the equality sign '=' of Calculus of Logic Circuit instead of 'iff' since the Calculus is isomorphic to Propositional Logic):

**B1**: For each $q \in SL_n$, add

$$Con(q,n) = Con(q, n-1) \vee (Supp(q,n) \wedge \neg Supp(\sim q, n) \wedge \neg Con(\sim q, n-1))$$

**B2**: For each set of rules $(q_1, ...q_m \rightarrow p) \in \mathbb{R}_n$ with head $p$, add

$$Supp(p,n) = \bigvee_{\forall t \in R[p]} (Con(q_{t,1}, n) \wedge ... \wedge Con(q_{t,m_t}, n))$$

**B3**: For all atoms $p$ appearing in equations of B1 or B2, but not appearing on the left-hand side of the equations of B1 and B2, add

$$p = 0$$

When it is obvious from the context, we will use $B(T)$ to denote $B(G(T))$. $B(T)$ can be considered both a set of Boolean equations and a propositional logic theory, since it is shown that a perfect analogy exists between the calculus for logic-circuit and the classical theory of calculus of propositions [28]. B1 is the Clark completion of the set of rules generated by the guidelines G1 and G2 defined in Section IV. B2 is the Clark completion of the set of rules generated by the guideline G3.

We can show that the following theorem holds: the mapping $B(T)$ is sound and complete.

*Theorem 1:* Let $T = (R, N)$ be an LAS theory, $B(T)$ the Boolean-equation mapping of $T$. If $T_n$ is decisive (i.e., for all $q$, either $T_n \vdash q$ or $T_n \not\vdash q$) for all $n$ $(1 \leq n \leq N)$, the following holds for all literals $q$ in $T$:

$$q \text{ is layer-}n \text{ justified iff } B(T) \models Con(q, n)$$

## VI. Mapping LAS into Verilog HDL

Given the Boolean equation representation of LAS theories, it is now straight forward to map LAS to Verilog. Verilog is one of widely used HDLs (Hardware Description Languages). Most of VLSI (Very-large-scale integrated) circuit design tools can import Verilog files to generate a netlist which can be used to fabricate ICs.

The mapping of an LAS theory into a Verilog module is similar to the definition of $B(T)$ which is given in the previous section. The differences are that all the used literals (referred as 'signals' in Verilog) in rules must be declared separately and signals have binary values: 1 and 0. We use value 1 to represent that a proposition (a signal) is 'true' and 0 to represent 'unknown'. All defined signals are assumed to have value 0 by default but can be changed to value 1 if it is derived by a logic gate with the output value of 1. The exact implementation of this property is device specific, and thus it will be ignored in this thesis.

Let $T = (R, N)$ be an LAS theory. Let $Right(e)$ be the right-hand side of a Boolean equation $e$ in B1, B2, and B3; $Left(e)$ be the left-hand side. The corresponding Verilog description $V(T)$ is obtained from $B(T)$ by adding the Verilog statements according to the following guidelines for each layer-$n$ $(1 \leq n \leq N)$:

**V1**: For each atom $q$ appearing in $B(T)$, add
wire $q$;

**V2**: For each Boolean equation $e \in B(T)$, add
assign $Left(e) = Right(e)$;

Figure 3 shows the corresponding Verilog code of the following rules:

$R_1 = \{r_1 : d \rightarrow c, \quad r_2 : w \rightarrow \neg v\}$
$R_2 = \{r_3 : c \rightarrow v\}$

In the figure, signal declarations and default-value assignments are omitted for clarity. Figure 4 shows the combinational

```
module CleaningL1( dsp1, wsp1, ccp1, wcp1, vcn1);
  input dsp1, wsp1;
  output wcp1, ccp1, vcn1;
  // assume all signals are declared :
  // For example, wire dsp1, wsp1, ccp1, wcp1;
  assign dcp1 = dsp1 & ~dsn1;
  assign wcp1 = wsp1 & ~wsn1;
  assign ccp1 = csp1 & ~csn1;
  assign vcn1 = vsn1 & ~vsp1;
  assign csp1 = dcp1;
  assign vsn1 = wcp1;
  // Assume 0 is assigned to all signals not assigned
  // For example, assign dsn1 = 0;
endmodule
```

```
module CleaningL2(dcp1, ccp1, wcp1, vcn1, vcp2);
  input dcp1, ccp1, wcp1, vcn1;
  output vcp2;
  // assume all signals are declared
  assign dcp2 = dcp1 | (dsp2 & ~dsn2 & ~dcn1);
  assign wcp2 = wcp1 | (wsp2 & ~wsn2 & ~wcn1);
  assign ccp2 = ccp1 | (csp2 & ~csn2 & ~ccn1);
  assign vcn2 = vcn1 | (vsn2 & ~vsp2 & ~vcp1);
  assign csp2 = dcp2;
  assign vsn2 = wcp2;
  assign vcp2 = vcp1 | (vsp2 & ~vsn2 & ~vcn1);
  assign vsp2 = ccp2;
  // Assume 0 is assigned to all signals not assigned
endmodule
```

Fig. 3. Verilog module hardware descriptions. CleaningL1 describes a module for layer-1 and ClearningL2 describes a module for layer-2.

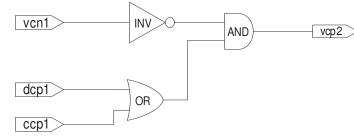


Fig. 4. Combinational logic circuit generated with Xilinx ISE for the layer-2 module "CleaningL2" shown in Figure 3.

logic circuit of layer-2 module CleaningL2 shown in Figure 3. This circuit is automatically generated by *Xilinx ISE Webpack*[1] from the Verilog description. For easy comparison with the Verilog code, the input and output signal names on the wires are labelled.

## VII. Discussion

Rosenchein [29] developed a compiler that directly synthesizes digital circuits from a knowledge based model [30] 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, their method does not provide integration with agent architectures and incremental synthesis.

In comparison, our language is more expressive since it is based on a nonmonotonic argumentation system that can express varying degrees of confidence and it can be easily extended to include the usual bounded temporal operators [31]. In addition, our approach is a bottom up characterization of a system through behavioral decomposition similar to programmable logic controllers (PLCs) and subsumption architectures [3]. However, LAS provides much more natural (nonmonotonic) language than PLCs and the original subsumption architecture. The advantage of this approach over model-based top-down approaches is demonstrated by the success of PLCs and highlighted in [32], which demonstrated that a completely autonomous mobile robot situated in a real-world environment can be built more easily through behavioral decomposition from bottom up without explicit representation of the world. Moreover, when the size of an agent's knowledge base becomes large, knowledge-based programming approaches easily become intractable [33], [34]. In comparison, LAS has a

[1]A freely available FPGA development tool from Xilinx.

polynomial space-and-time mapping into a target hardware. Incremental synthesis is also possible, this is because the guide line **B2** in Section V states that literals supported by new rules can be simply ORed with existing rules.

The notable differences with existing agent system synthesis approaches, such as [29], [35], are that (1) the Boolean equations representation has almost one to one correspondence with its original theory as shown in Section V; (2) layered architecture allows existing rules to be overridden. We should note that priority relations in existing rule-based languages (e.g. Defeasible logic [36]) require much more complex process to add new rules to override existing rules. In LAS, there is no need to create new literals unless the rules contains new literals. Thus, an existing system can be extended or updated without recompiling the whole specification again. This is particulary important if the knowledge base of an agent must be changed frequently. Many practical applications, such as the video-surveillance agent-chip example shown in Section II, demand this ability. In addition, this way, agent chips can also learn new rules and update there circuits in real-time. This is a significant advantage over other hardware synthesis methods (e.g. [29], [35]) and over other software based executable agent specification methods (e.g. [37], [30]).

## VIII. Conclusion

The main result of this paper is a solution to the problem of automatically synthesizing reactive agents. The key idea has been the mapping of a simple and expressive rule-based language into hardware structure which can be modified and extended just by adding new rules during run-time. Existing behaviors can be overridden just by adding new more-confident rules without compiling the whole system specification. This is possible because of the layered architecture. New behaviors can also be added simply by adding new rules.

## References


[1] S. Dhanani, "FPGAs enabling consumer electronics a growing trend," *FPGA and Programmable Logic Journal*, June 2005.

[2] A. S. Rao and M. P. Georgeff, "Decision procedures for BDI logics," *Journal of Logic and Computation*, vol. 8, no. 3, pp. 293–343, June 1998.

[3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[4] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre, "The BOID architecture: conflicts between beliefs, obligations, intentions and desires," in *Proc. AGENTS '01: the fifth international conference on Autonomous agents*. New York, NY, USA: ACM Press, 2001, pp. 9–16.

[5] R. Reiter, "A logic for default reasoning," in *Artificial Intelligence*, 1980, pp. 81–132.

[6] D. Nute, "Defeasible logic," in *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford: Oxford University Press, 1994, pp. 353–395.

[7] J. McCarthy, "Circumscription—a form of non-monotonic reasoning," *Artificial Intelligence*, vol. 13, pp. 27–39, 1980.

[8] D. McDermott and J. Doyle, "Nonmonotonic logic 1," *Artificial Intelligence*, vol. 13, pp. 41–72, 1980.

[9] R. Moore, "Semantics considerations on nonmonotonic logic," *Artificial Intelligence*, vol. 25, pp. 75–94, 1985.

[10] E. Amir and P. Maynard-Zhang, "Logic-based subsumption architecture," *Artif. Intell.*, vol. 153, no. 1-2, pp. 167–237, 2004.

[11] M. Wooldridge, P. McBurney, and S. Parsons, "On the meta-logic of arguments," in *Proc. AAMAS '05*, 2005, pp. 560–567.

[12] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial Intelligence*, vol. 77, no. 2, pp. 321–358, 1995.

[13] A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni, "An abstract, argumentation-theoretic approach to default reasoning," *Artificial Intelligence*, vol. 93, pp. 63–101, 1997.

[14] H. Prakken and G. Sartor, "Argument-based extended logic programming with defeasible priorities." *Journal of Applied Non-Classical Logics*, vol. 7, no. 1, 1997.

[15] P. Besnard and A. Hunter, "Practical first-order argumentation," in *AAAI'2005*. MIT Press, 2005, pp. 590–595.

[16] D. Dubois, J. Lang, and H. Prade, "Possibilistic logic," in *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, D. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Oxford: Oxford University Press, 1994, pp. 439–513.

[17] C. I. Chesnevar, G. R. Simari, T. Alsinet, and L. Godo, "A logic programming framework for possibilistic argumentation with vague knowledge," in *Proc. AUAI '04: the 20th conference on Uncertainty in artificial intelligence*, 2004, pp. 76–84.

[18] G. G. Insu Song and J. Diederich, "Layered argumentation for fuzzy automation controllers," in *4th IEEE International Conference on Cybernetics and Intelligent Systems*, 2010, p. In Print.

[19] I. Song and G. Governatori, "Designing agent chips," in *5th International Conference on Autonomous Agents and Multi-Agent Systems*, P. Stone and G. Weiss, Eds. ACM Press, 10–12 May 2006, pp. 1311–1313.

[20] I. Song and G. Goverantori, "A compact argumentation system for agent system specification," in *Proceedings of the Third Starting AI Researchers' Symposium: STAIRS'06*, vol. 142, 2006.

[21] G. Governatori, M. J. Maher, G. Antoniou, and D. Billington, "Argumentation semantics for defeasible logics," *Journal of Logic and Computation*, vol. 14, no. 5, pp. 675–702, 2004.

[22] K. Konolige, "Hierarchic autoepistemic theories for nonmonotonic reasoning," in *Non-Monotonic Reasoning: 2nd International Workshop*, 1989, vol. 346, pp. 42–59.

[23] L. A. Stein, "Resolving ambiguity in nonmonotonic inheritance hierarchies," *Artif. Intell.*, vol. 55, no. 2-3, pp. 259–310, 1992.

[24] D. Touretzky, J. Horty, and R. Thomason., "A clash of intuitions: The current state of nonmonotonic multiple inheritance systems," in *Proc. IJCAI-87*. Morgan Kaufmann, 1987, pp. 476–482.

[25] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, "A flexible framework for defeasible logics," in *AAAI 2000*, 2000, pp. 405–410.

[26] K. L. Clark, "Negation as failure," *Logic and Databases*, pp. 293–322, 1978.

[27] R. Ben-Eliyahu and R. Dechter, "Default reasoning using classical logic," *Artif. Intell.*, vol. 84, no. 1-2, pp. 113–150, 1996.

[28] C. E. Shannon, *A symbolic analysis of relay and switching circuits*, 1938.

[29] S. J. Rosenschein and L. P. Kaelbling, "A situated view of representation and control," *Artif. Intell.*, vol. 73, no. 1-2, pp. 149–173, 1995.

[30] R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*. Cambridge, MA: MIT Press, 1995.

[31] I. Song and Governatori, "Hardware implementation of temporal nonmonotonic logic," in *Advances in Artificial Intelligence*, ser. LNCS. Berlin: Springer-Verlag, 2006, p. In print.

[32] R. A. Brooks, "Intelligence without representation," *Artif. Intell.*, vol. 47, no. 1-3, pp. 139–159, 1991.

[33] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, "Knowledge-based programs," in *Symposium on Principles of Distributed Computing*, 1995, pp. 153–163.

[34] M. Y. Vardi, "Implementing knowledge-based programs," in *Proc. TARK 1996*, Y. Shoham, Ed. San Francisco: Morgan Kaufmann, 1996, pp. 15–30.

[35] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. POPL '89*. New York, NY, USA: ACM Press, 1989, pp. 179–190.

[36] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, "Representation results for defeasible logic," *ACM Transactions on Computational Logic*, vol. 2, no. 2, pp. 255–287, 2001.

[37] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Agents Breaking Away (LNAI 1038)*. Springer-Verlag: Heidelberg, Germany, 1996, pp. 42–55.