Limitations of learning in automata-based systems.

OLIVEIRA, F.S.

2010

European Journal of Operational Research, 203 (3): 684-691, 2010.

Limitations of Learning in Automata-Based Systems

Fernando Oliveira, ESSEC Business School, oliveira@essec.edu

In this article, we aim to analyze the limitations of learning in automata-based systems by introducing the L^+ algorithm to replicate quasi-perfect learning, i.e., a situation in which the learner can get the correct answer to any of his queries. This extreme assumption allows the generalization of any limitations of the learning algorithm to less sophisticated learning systems. We analyze the conditions under which the L^+ infers the correct automaton and when it fails to do so. In the context of the repeated prisoners' dilemma, we exemplify how the L^+ may fail to learn the correct automaton. We prove that a sufficient condition for the L^+ algorithm to learn the correct automaton is to use a large number of look-ahead steps. Finally, we test empirically, in the product differentiation problem, that the computational time of the L^+ algorithm is polynomial on the number of states but exponential on the number of agents.

Key words: artificial intelligence; knowledge-based systems; learning; multi-agent systems.

1. Introduction

If we abandon the perfect rationality paradigm in favor of models of bounded rationality and learning, one first question we need to answer is: how do people and organizations learn? How complex are their learning processes? Several articles have addressed this issue (e.g., Weiss, 1995; Carmel and Markovitch, 1998, 1999; Donkers et al., 2005; Wainer et al., 2007; Pendharkar, 2007), however, none of these seems to capture the way sophisticated organizations (which are complex structures composed by intelligent agents) learn. In this article we modify an algorithm developed by Angluin (1987), the L^*, which enables perfect learning in the context of finite automata, to create a learning algorithm, the L^+, which enables quasi-perfect learning. The L^+ captures how a sophisticated organization learns, as it is always able to infer the correct solution to any query.

Automata-based systems have been used in several areas of operational research, for example: to study how commuters choose alternative roads, van Ackere and Larsen (2004); to solve the shortest path problem with forbidden paths, Villeneuve and Desaulniers (2005); to develop classification systems, Gérard et al. (2005); to model asset trading in an electricity market, Bunn and Oliveira (2007, 2008); to model real-options, Oliveira (2009); to represent complex Markov systems, e.g., Stewart et al. (1995), Uysal and Dayar (1998), Gusak et al. (2003), Sbeity et al. (2008); and to model human-computer interaction, Gmytrasiewicz and Lisetti (2002), and bilateral negotiation, Vassileva and Mudgal (2002).

In this article, we study models of learning that infer automata, i.e., the behavior rules of a given subject (e.g., Shen, 1994; Mor et al., 1996; Carmel and Markovitch, 1996, 1998, 1999; Donkers et al., 2005), for two reasons: first, we can adapt the L^* to model quasi-perfect learning; second, it has been shown that models of learning, when having discrete states and actions, can be represented as automata (e.g., Gérard et al., 2005), therefore, a good understanding of the latter is important to model learning behavior. In this context, the introduction of the L^+ has the same procedural complexity as the modification of the Angluin's (1987) L^* algorithm made by Carmel and Markovitch (1996) to develop the US-L^* (unsupervised L^*), which differed from the L^* by considering that there is no teacher to guide and answer the queries of the learner (as assumed by Angluin). In our case, the L^+ algorithm assumes that there is no teacher but it allows the agent to always get the correct answer to any of his queries. Therefore, in terms of its learning abilities, the L^+ is placed between the US-L^* and the L^* (which represents perfect learning). Moreover, in Proposition 5.6 we show that, under certain conditions, the L^+ behaves as the L^*.

Why is our modification to the L^* important? First, the L^+ algorithm allows the learner to get the best possible answer to his queries. This is arguably the case of the learning abilities of the complex and sophisticated organizations which we aim to model. The limitations of information imposed by the US-L^* are realistic to model how a learner behaves in a poker game, for example, but not to model how a firm learns in the real world. The importance of the L^+ is not the algorithm itself, as it is very similar to the US-L^* and to the L^*, but the idea that by a simple modification of the L^* we are able to model quasi-perfect learning. Second, the L^+ recognizes the importance of the query, i.e., it is important to *know what to ask*. This is the main difference when compared to the L^*, in which the answers and *the questions* are given to the learner by a benevo-

lent teacher. Third, the use of the L^+ allows the modeling of learning in situations in which the agent has a very limited number of opportunities to interact with the system. (For example, a government cannot interact with the system often when deciding to build a nuclear power plant, to invest in a new defense system, or to nationalize banks.) In this case we cannot use reinforcement learning (e.g., Bertsekas and Tstsiklis, 1996) as it assumes that the learner can interact with the system a very large number of times.

In this article, we analyze the conditions under which the L^+ infers a correct model and when it fails to do so, and we study its main properties. As in all the instances in which the L^+ algorithm fails to infer the correct model the US-L^* also fails, by analyzing the limitations of the L^+ we also expose the boundaries of the US-L^*. We show that the automata generated by L^+ are always closed (the L^* also generates closed automata but not the US-L^*), which we prove to be a basic condition for optimal behavior. We analyze how the L^+ may fail to learn the correct automaton in the repeated prisoners' dilemma; this result suggests that this simple game requires perfect learning for the Nash equilibrium to hold. We prove that a sufficient condition for the L^+ to approximate perfect learning is to include a large enough number of look-ahead steps. Finally, we test empirically that the computational time of the L^+ algorithm is polynomial on the number of states, but exponential on the number of agents (this applies also to the US-L^* and L^*).

We proceed in section 2 by introducing the basic concepts in automata systems. In section 3 we review the literature in automata inference. In section 4 we describe the L^+ algorithm and in section 5 we analyze its properties. Section 6 presents an application of the L^+ algorithm to the product differentiation problem. Section 7 concludes the paper.

2. Basic Concepts in Automata Systems

This section formalizes the basic concepts in the automata literature. A very good introduction to the topic is Hopcroft and Ullman (1979). We start by defining deterministic finite automaton (Definition 2.1). Throughout the paper we use automata as a short form for "Deterministic Finite Automata".

Definition 2.1: *A deterministic finite automaton* $A^i = \left(Q^i, q_0^i, \Sigma^i, Z^i, \delta^i, \lambda^i\right)$ *is a 6-tuple in which:* a) Q^i *stands for a finite non-empty set of states of the automaton; b)* q_0^i *is the initial internal state; c)* Σ^i *is a non-empty set of possible actions of agent i; d)* Z^i *represents a finite non-empty set of possible outcomes of the system, in which each* $z^i \in Z^i$ *is dependent on the actions of each agent,* $z^i = Z^i\left(a^1,...,a^i,...,a^N\right)$*; e)* δ^i *is a transition function* $\left(\delta^i : Q^i \times Z^i \to Q^i\right)$*; f)* λ^i *is a behavioral function* $\left(\lambda^i : Q^i \to \Sigma^i\right)$ *associating an action to each possible internal state.*

Definition 2.1 is both correct and complete as it is necessary and sufficient to describe any deterministic finite automaton. However, in section 5 we show that a finite automaton can be computed as a result of an explicit optimization process which requires the definition of a utility measure u^i over the outcomes, in which $u^i = u\left(z^i\right)$ represents the utility function of agent *i*, i.e., it is the utility an agent *i* receives from the outcome z^i. The branch of research concerned with the computation of the optimal automaton, which is not the topic of this paper, and it is only briefly used in Proposition 5.1, includes the computation of best response automata (e.g., Banks and Sundaram, 1990; Piccione, 1992; Gilboa, 1988) and the analysis of algorithmic complexity (e.g., Papadimitriou, 1992; Ben-Porath, 1990; Freund et al., 1995).

The behavior of an automata-based system is defined by its individual components, i.e., by the automaton used by each agent. A system of automata can be described by a *product automaton* (Definition 2.2) in which the states are the combination of the states of the individual automata.

Definition 2.2: *The product automaton of N automata is a 6-tuple* $W = \left(Q, q_0, \Sigma, Z, \delta, \lambda\right)$ *that defines how the environment behaves, where* $Q = Q^1 \times Q^2 \times ... \times Q^N$, $q_0 = q_0^1 \times q_0^2 \times ... \times q_0^N$, $\Sigma = \Sigma^1 \times \Sigma^2 \times ... \times \Sigma^N$, $Z = Z^1 \times .Z^2 \times ... \times Z^N$, $\delta : Q \times \Sigma \to Q'$, $\lambda : Q \to \Sigma$.

In this article we introduce the concept of *residual product automaton* (Definition 2.3). In order for an agent to model the problem he only needs to infer the residual product automaton which describes his opponents, as the structure of an automata-based system is an automaton itself.

Definition 2.3: *The residual product automaton of the system* $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ *represents a model of all N-i agents, in which* $Q^{mi} = Q^1 \times ... \times Q^{i-1} \times Q^{i+1} \times ... \times Q^N$, $q_0^{mi} = q_0^1 \times ... \times q_0^{i-1} \times q_0^{i+1} ... \times q_0^N$, $\delta^{mi} : Q^{mi} \times \Sigma^i \to Q^{mi}$ and $\lambda^{mi} : Q^{mi} \to Z^i$.

The residual product automaton M^i is derived from the product automaton W by removing all the states associated to agent i and replacing the transition function depending on all the actions from all the agents by a transition function that depends only on agent i's actions, i.e., $\delta^{mi} : Q^{mi} \times \Sigma^i \to Q^{mi}$ (this function changed when compared to Definition 2.1 as the state transition is a function of i's actions). The behavioral function $\lambda^{mi} : Q^{mi} \to Z^i$ has also been modified as the outcomes (and not the actions) are a function of the state.

A second concept we introduce in this article is the perceived residual product automaton, P^i (Definition 2.4) which describes the residual product automaton learned by an agent i. Its structure is similar to the M^i, as it is assumed that an agent correctly observes the outcomes z^i and he knows his own actions Σ^i.

Definition 2.4: *The perceived residual product automaton* $P^i = \left(Q^{pi}, q_0^{pi}, \Sigma^i, Z^i, \delta^{pi}, \lambda^{pi}\right)$ *represents the model an agent infers from his opponents, in which* $\delta^{pi} : Q^{pi} \times \Sigma^i \to Q^{pi}$ and $\lambda^{pi} : Q^{pi} \to Z^i$.

How can an agent infer a model of the residual product automaton? To answer this question, before we proceed, in section 3 we revise the literature on automata inference.

3. Inferring Models of Deterministic Finite Automata

This section reviews the inference problem, i.e., how to learn the automaton generating a given stream of data. We concentrate our description on active learning (e.g., Angluin, 1987) as passive learning (e.g., Gold, 1978) is not used in the L^+ algorithm. In active learning an agent has the ability to influence the input generation process, i.e., he is able to select the inputs supplied to the automaton generating the data. Angluin (1987) showed that a learning algorithm provided with counterexamples, and with the possibility of controlling the inputs to the target automaton

on any specific type of input, can learn the target automaton in polynomial time (this algorithm was named L^*). We present the basic concepts of the L^* adapted to the residual product automaton.

An agent *i* attempts to infer a residual product automaton M^i. This learner holds a nonempty set of actions Σ^i, which are the inputs for the residual product automaton. The learner *i*, in order to infer M^i, maintains an *observation table* (*S, E, T*), see Definition 3.1, in which S is a non-empty finite pre-fixed closed set of strings (a set of strings in which all the initial elements of the strings are in the set), and E is a nonempty suffix-closed set of strings called tests (a set of strings in which all the final elements of the strings are in the set). A string, in this case, represents a succession of actions.

Definition 3.1: *T is a finite two-dimensional table with one row for each element of the set* $S \cup S\Sigma^i$ *and one column for each element of E, in which* $S\Sigma^i = \{s\sigma : s \in S, \sigma \in \Sigma^i\}$.

In the observation table the agent records the outcomes received from interacting with M^i for every possible sequence (string) of actions chosen in the previous history. Additionally, the table also records all the predictions about the future behavior of M^i, by describing the outcomes from choosing the actions in *E*.

Before proceeding, we define the *row operator,* which represents the content of a line in table *T*. For example, *row(s)* represents the ordered set of outcomes for string *s* and each one of the tests in *E,* in which for each $e \in E$ we compute $T(s,e)$, i.e., the outcome of a string of actions *se*. The *row operator* represents a sequence of these outcomes for the tests: $\varepsilon, e_1\varepsilon, e_2e_1\varepsilon, ..., e_E...e_2e_1\varepsilon$, which define a suffix-closed set of strings (in which ε represents the empty string). I.e., the *row operator* summarizes the content of a row in the table (each row is an ordered sequence of outcomes which are a function of the past actions for that specific row and tests used in the table). The examples in section 4 help to clarify these concepts. The task of the learning algorithm is to infer an observation table *T* which is consistent and closed: see Definitions 3.2 and 3.3.

Definition 3.2: *An observation table is said to be consistent if for any two elements* $s_1, s_2 \in S$, *such that,* $row(s_1) = row(s_2)$, *and for all actions* $\sigma \in \Sigma^i$, $row(s_1\sigma) = row(s_2\sigma)$.

This procedure checks that, for any two strings s_1 and s_2 in S and for every test and action, any extension σ of s_1 and s_2 has the same output. This consistency requirement implies that the agent holds a model that, for every state, does not forecast different transitions for the same action. When this model is not consistent an additional test σe is added to E. We repeat this process until the table is such that for any two strings with the same row operator outcome (s_1 and s_2 in S) any extension in the set $S\Sigma^i$ still has the same outcome.

Definition 3.3: *An observation table is said closed if, for each element* $\phi^i \in S\Sigma^i$, *there is an observation* $s \in S$ *such that* $row(\phi^i) = row(s)$.

In order to verify that a table is closed, we check that for any string $\phi^i \in S\Sigma^i$ (the extensions of S) there is an element in S such that it contains exactly the same outcomes as ϕ^i. In other words, if a model is closed every state of the automaton in $S\Sigma^i$ is already represented in S. The closure requirement implies that an agent computes a forecast for every action in every state of the automaton.

In summary, the L^* can be described as follows. Given a stream of data received during the interactions with the environment, a learner builds a table T that represents the perceived residual product automaton, P^i. This table contains the whole set of data, both the observed stream of actions and their outcomes, and the forecasted actions with the expected outcomes. Thus, in table T, S represents the stream of actions played by agent i (and the table contains the received outcomes) whilst $S\Sigma^i$ represents the forecasted stream of actions (and the table T itself contains the expected outcomes).

4. The L^+ Algorithm

The L^+ algorithm, Table 1, modifies Angluin's (1987) L^* and Carmel and Markovitch's (1996) US-L^* in order to model problems in which, even though there is no teacher, the agent is able to always get the correct answer to any query. (However, since there is no benevolent teacher the agent does not know which questions to ask.) The L^+ is applied to model systems with N automata by using the concept of residual product-automaton (introduced in Section 2). Note that, by

using this same formalization, both the L^* and the US-L^* can be adapted to model *N*-automata systems.

Table 1: The L^+ Algorithm

Algorithm $L^+(D^i)$

D^i: Data collected by agent *i*

P^i: The perceived residual product automaton model consistent with D^i

Σ^i: Set of possible actions of agent *i*

$\phi^i = (s, \sigma)$: New example of the product automaton's behavior

ε: Empty string

S: Non-empty finite pre-fixed closed set of strings

E: Non-empty finite suffix closed set of strings, called tests

T: Finite two dimensional table

Step 1. Initialize (*S, E, T*)

$S \equiv$ All prefixes of D^i

$\forall s \in S$ and $\sigma \in \Sigma^i$ if $s\sigma \notin S$ then $S\Sigma^i \leftarrow S\Sigma^i \cup \{s\sigma\}$

$E = \{\varepsilon\}$

$\forall s \in S \cup S\Sigma^i$ the table value is set using forecasting queries: $T(s, \varepsilon) = \Phi(s, \varepsilon)$.

Step 2. Check consistency

While (*S, E, T*) is not consistent,

Find $s_1, s_2 \in S$, $\sigma \in \Sigma^i$ and $e \in E$

Such that row(s_1) = row(s_2) and $T(s_1\sigma, e) \neq T(s_2\sigma, e)$, $E \leftarrow E \cup \{\sigma e\}$

$\forall s \in S \cup S\Sigma^i$ the table value is set using forecasting queries:

$T(s, \sigma e) = \Phi(s, \sigma e)$.

Step 3. Check the automaton is closed

While (*S, E, T*) is not closed

Find $s \in S\Sigma^i$ such that for all $s' \in S$ $row(s') \neq row(s)$: $S \leftarrow S \cup \{s\}$

$\forall \sigma \in \Sigma^i$, $S\Sigma^i \leftarrow \{S\Sigma^i \setminus \{s\}\} \cup \{s\sigma\}$

The table value is set using forecasting queries

$\forall e \in E$, $T(s\sigma, e) = \Phi(s\sigma, e)$.

Return P^i(*S*, *E*, *T*)

Φ (*s*, *e*):- returns the correct outcome of the example ϕ^i (*s*, *e*)

In step 1 we initialize a table T containing all the prefixes of D^i and the empty test set $E = \{\varepsilon\}$, and we build the set $S\Sigma^i$ containing all the possible extensions of the strings in S by using the actions $\sigma \in \Sigma^i$. Then table $S \cup S\Sigma^i$ (see Definition 3.1) is completed using the query function $T(s,e) = \Phi(s,e)$, which returns the outcome of example $\phi^i(s, e)$.

In step 2 we check that the model inferred in step 1 is consistent (see Definition 3.2). Having checked that table T is consistent we can then proceed to confirm that it is also closed (see Definition 3.3), step 3. As in the L^*, the model inferred by the L^+ is always closed, this is not the case of the model inferred by the US-L^*.

This section proceeds by presenting three different examples illustrating how the L^+ algorithm works under different settings.

4.1. The L^+ Fails to Learn the Correct Model

Assume that the learner chooses between two possible actions $\Sigma^i = \{a,b\}$ with two possible outcomes observed by i, $Z^i = \{0,1\}$. In all the examples we assume that the learning agent observes the aggregated behavior of his opponents but not the individual actions. For example, if we have two opponents, then at each state of the residual product automaton we have one of four possible states arrangements of opponents' actions *aa*, *ab*, *ba*, *bb*. The agent observes the aggregated behavior of λ^{mi}, for example $\lambda^{mi}(aa) = \lambda^{mi}(bb) = 1$ and $\lambda^{mi}(ab) = \lambda^{mi}(ba) = 0$.

Let Table 2.a) describe the residual product automaton $M^i = (Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi})$ and Table 2.b) represent agent i's initial model of the residual product automaton $P_1^i = (Q^{pi}, q_0^{pi}, \Sigma^i, Z^i, \delta^{pi}, \lambda^{pi})$.

Table 2: a) The Residual Automaton M^i b) Perceived Residual Automaton P_1^i

		δ^{mi}	
Q^{mi}	λ^{mi}	a	b
q_1^{mi}	0	q_1^{mi}	q_2^{mi}
q_2^{mi}	0	q_3^{mi}	q_4^{mi}
q_3^{mi}	0	q_5^{mi}	q_4^{mi}
q_4^{mi}	1	q_2^{mi}	q_3^{mi}
q_5^{mi}	1	q_1^{mi}	q_5^{mi}

		δ^{pi}	
Q^{pi}	λ^{pi}	a	b
q_1^{pi}	0	q_1^{pi}	q_1^{pi}

In Table 2.a), δ^{mi} represents the transition function, showing how the internal state changes, due to the actions of the agent, from the one in column Q^{mi} to a new state. λ^{mi} stands for the behavioral function, specifying the outcomes of the residual product automaton, in each one of the states in Q^{mi}. In Table 2.b) P_1^i is compatible with the available data $D^i = \{(\varepsilon,0),(a,0),(ab,0)\}$, i.e., P_1^i could have been the automaton generating D^i. In his next move, the agent chooses *b* specifying a string *abb* and receiving a counterexample $\phi^i = (abb,1)$, which makes him revise the inferred model.

The inference process starts with the construction of the table (*S, E, T*) as shown in Table 3 (it includes an extra column Q^{pi} which classifies each row into the corresponding state of the model). The algorithm proceeds as described in Table 1. As a first step agent *i* checks the consistency and closure of the new model. The model in Table 3 is not consistent as *row*(ε) = *row*(*ab*) but $row(b) \neq row(abb)$, i.e., the same action *b*, applied in the same state, q_1^{pi}, leads to a different outcome.

In order to solve the inconsistency, agent *i* adds another test, action *b*, to the tests set *E*. Table 4.a) represents the new model (*S, E, T*) closed and consistent with the new data. Table 4.b) represents the new perceived residual product automaton P_2^i.

This example shows that the L^+ can lead to inference errors, even though the agent was able to answer his own queries correctly. This shows that the ability to ask important questions is a component of perfect learning (as in the L^*).

Table 3: (*S, E, T*) Revised Table

		E	Q^{pi}
		ε	
S	ε	0	q_1^{pi}
	a	0	q_1^{pi}
	ab	0	q_1^{pi}
	abb	1	q_2^{pi}
$S\Sigma^i$	b	0	q_1^{pi}
	aa	0	q_1^{pi}
	aba	0	q_1^{pi}
	abba	0	q_1^{pi}
	abbb	0	q_1^{pi}

Table 4: a) (*S, E, T*) Table

		E		Q^{pi}
		ε	$b\varepsilon$	
S	ε	0	0	q_1^{pi}
	a	0	0	q_1^{pi}
	ab	0	1	q_2^{pi}
	abb	1	0	q_3^{pi}
$S\Sigma^i$	b	0	1	q_2^{pi}
	aa	0	0	q_1^{pi}
	aba	0	1	q_2^{pi}
	abba	0	1	q_2^{pi}
	abbb	0	1	q_2^{pi}

b) Perceived Residual Automaton P_2^i

		δ^{pi}	
Q^{pi}	λ^{pi}	a	b
q_1^{pi}	0	q_1^{pi}	q_2^{pi}
q_2^{pi}	0	q_2^{pi}	q_3^{pi}
q_3^{pi}	1	q_2^{pi}	q_2^{pi}

4.2. L^+ Learns the Correct Model of a Finite Automaton

Let us look at a modified version of the previous example. Let Table 2.b) represent agent *i*'s initial perceived residual product automaton P_1^i. Once again, this model is compatible with the available data: $D^i = \{(\varepsilon,0),(a,0)\}$. In this example, Table 5.a) describes the residual product automaton M^i. The main feature of this automaton is that it chooses a different action in each state.

Agent *i* plays *b*, specifies a string *ab* and receives a counterexample $\phi^i = (ab,1)$, which makes him revise the inferred model, constructing the table (*S, E, T*) as shown in Table 5.b).

Table 5: a) The Residual Automaton M^i

		δ^{mi}	
Q^{mi}	λ^{mi}	*a*	*b*
q_1^{mi}	0	q_1^{mi}	q_2^{mi}
q_2^{mi}	1	q_3^{mi}	q_4^{mi}
q_3^{mi}	2	q_5^{mi}	q_4^{mi}
q_4^{mi}	3	q_2^{mi}	q_3^{mi}
q_5^{mi}	4	q_1^{mi}	q_5^{mi}

b) (*S, E, T*) Revised Table

		E	Q^{pi}
		ε	
S	ε	0	q_1^{pi}
	a	0	q_1^{pi}
	ab	1	q_2^{pi}
$S\Sigma^i$	b	1	q_2^{pi}
	aa	0	q_1^{pi}
	aba	0	q_1^{pi}
	abb	3	q_4^{pi}

The model in Table 5.b) is not closed as *abb* leads to a state in which actions are unknown. The agent needs to analyze more actions to close the table, including *abbb* and *abbba* in *S*, the result is in Table 6, which is closed, consistent and correctly represents the automaton in Table 5.a).

Table 6: (*S, E, T*) Revised Table

		E	Q^{mi}
		ε	
S	ε	0	q_1^{mi}
	a	0	q_1^{mi}
	ab	1	q_2^{m}
	abb	3	q_4^{m}
	abbb	2	q_3^{m}
	abbba	4	q_5^{m}
$S\Sigma^i$	b	1	q_2^{m}
	aa	0	q_1^{mi}
	aba	0	q_1^{mi}
	abba	1	q_2^{m}
	abbbb	3	q_4^{m}
	abbbaa	0	q_1^{mi}
	abbbab	4	q_5^{m}

4.3. The L^+ Fails to Learn an Automaton for the Repeated Prisoner's Dilemma

In this section we analyze the prisoner's dilemma. In this problem two prisoners (*A* and *B*) were accomplices in a crime. Each one of them can confess the crime or not. If both agents confess, i.e. play Defect (*D*) then they go to prison for four years. If only one of them confesses (chooses *D*) and the other cooperates (chooses *C*), the agent that confesses serves no time, while the agent that cooperates gets five years in prison. If none of them confesses (both play *C*) then each one gets two years in prison. Table 7 represents the payoff matrix.

Table 7: Payoff Matrix

		B	
		C	D
A	C	(2, 2)	(5, 0)
	D	(0, 5)	(4, 4)

We analyze the repeated prisoner's dilemma, as presented in Carmel and Markovitch (1996), as they have used this game as a test bed for the US-L* algorithm. In this version of the game, two players interact repeatedly playing the same game. This may seem surreal in the case of a "true" prisoner's dilemma but it may resemble real-world situations such as, for example, e-collaboration (Cai and Kock, 2009) and investment games (Oliveira, 2009).

As shown in Rubinstein (1986) a pure strategy for the repeated prisoners' dilemma can be represented by a deterministic finite automaton. Let Table 8 stand for the automaton used by agent *A* to play a pure strategy in the repeated prisoner's dilemma (representing agent B's residual product automaton, M^B). The question is: can agent *B* infer the correct residual automaton representing *A*'s behavior? This is important as it restricts *B*'s choices.

Table 8: The Residual Product Automaton M^B

		δ^{mB}	
Q^{mB}	λ^{mB}	*C*	*D*
q_1^{mB}	*C*	q_1^{mB}	q_2^{mB}
q_2^{mB}	*D*	q_3^{mB}	q_2^{mB}
q_3^{mB}	*D*	q_1^{mB}	q_3^{mB}

In games with very high stakes, such as the prisoner's dilemma, an agent cannot afford to repeat the game a very high number of times in order to learn the optimal action at each state using reinforcement learning (e.g., Bertsekas and Tstsiklis, 1996). When using reinforcement-learning an agent attempts non-optimal actions in order to explore the strategy space even though these may, for example, lead the opponent to forever play D (if there was such a sink state in the automaton): in such situation not only the learning algorithm fails to learn the optimal strategy but it leads to the worst possible outcome.

In reality, in such situations the agents may learn not by acting but by studying, analyzing, enquiring, i.e., by asking the right questions. Such a process of inquiry can be emulated by the L^+. In the case of the prisoner's dilemma this means that, before committing the first crime, the criminal enquires about the character (i.e., automaton) of his accomplice and learns as much as he can about his behavior. This learning by query provides an explanation for cooperation in the prisoner's dilemma, even when the Nash equilibrium of the game is defection: by using learning by query the potential criminal associates only with accomplices which he can trust to cooperate in case they are caught. Next, we illustrate how the L^+ can be used to imitate this learning process.

Let Table 9 represent agent *B*'s initial perceived residual product automaton P_1^B. This model is compatible with the available data $D^B = \{(\varepsilon, C), (C, C)\}$.

Table 9: The Perceived Residual Product Automaton P_1^B

		δ^{pB}	
Q^{pB}	λ^{pB}	C	D
q_1^{pB}	C	q_1^{pB}	q_1^{pB}

Agent *B* plays *D*, as he thinks that agent *A* will always cooperate, and therefore generates a string *CD* and gets a counterexample $\phi^B = (CD, D)$, which makes him revise the inferred model, constructing the table (*S, E, T*), Table 10.a). The model represented in Table 10.a) is closed and consistent but represents the automaton in Table 10.b), which is not correct. This shows that the prisoners' dilemma requires the perfect learning of the L^* for a prisoner to learn the correct automaton.

Table 10: a) (*S, E, T*) Revised Table

		E	Q^{pB}
		ε	
S	ε	C	q_1^{pB}
	C	C	q_1^{pB}
	CD	D	q_2^{pB}
$S\Sigma^i$	D	D	q_2^{pB}
	CC	C	q_1^{pB}
	CDC	D	q_2^{pB}
	CDD	D	q_2^{pB}

b) The Perceived Residual Automaton P_2^B

		δ^{pB}	
Q^{pB}	λ^{pB}	C	D
q_1^{pB}	C	q_1^{pB}	q_2^{pB}
q_2^{pB}	D	q_2^{pB}	q_2^{pB}

Next, in section 5 we generalize these findings, providing an analysis of the L^+ algorithm.

5. Analysis of the L^+ Algorithm

We now analyze the properties of the L^+ algorithm. First, we justify why an automaton needs to be closed and consistent. If an agent wants to optimize his behavior he needs to maximize the present value of his utility, i.e., $V_{i,t}^*\left(q_t^i\right) = u_t^i\left(z_t^i\right) + \rho_i V_{i,t+1}^*\left(\delta^i\left(q_t^i, a_t^i\right)\right)$, where $V_{i,t}^*$ represents the value an agent receives by playing against M^i, and ρ_i stands for the discount factor. In Proposition 5.1 we prove that the ability of an agent to infer a closed and consistent model (P^i) is a necessary condition to compute $V_{i,t}^*$. The proof is based on the observation that if the inferred automaton is not closed and consistent the agent cannot compute the best response due to an ill defined transition function.

Proposition 5.1: *Let equation* $V_{i,t}^*\left(q_t^i\right) = u_t^i\left(z_t^i\right) + \rho_i V_{i,t+1}^*\left(\delta^i\left(q_t^i, a_t^i\right)\right)$ *represent the present value of agent i's utility. A necessary condition for the computation of the optimal strategy is the capability of an agent to infer a closed and consistent model* (P^i) *of the residual product automaton.* [Proof is in the online appendix.]

Therefore, the ability of an agent to describe the problem as an automaton is a necessary condition for optimal behavior. Consequently, as the US-L^* does not always generate closed automata it precludes optimal behavior by the learning agent (meaning that in these type of games optimal behavior may not be possible), this necessary condition is always met when we use the L^+ (or the L^*) as the inferred automata are always closed and consistent, as shown in Proposition 5.2.

Proposition 5.2. *Under L^+, for any set of data D^i there is at least one closed and consistent automaton capable of generating D^i.* [Proof is in the online appendix.]

However, even though the L^+ always infers a closed and consistent model, it may not describe the complete residual product automaton, as shown in Proposition 5.3. We prove that an automaton closed, consistent and compatible with a set of data D^i can be extended into an automaton with a larger number of states and compatible with D^i.

Proposition 5.3. *Let $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ represent a closed and consistent automaton compatible with the data D^i. It is always possible to find an automaton $H^i = \left(Q^{hi}, q_0^{hi}, \Sigma^i, Z^i, \delta^{hi}, \lambda^{hi}\right)$, also closed, consistent and compatible with the data D^i, such that $Q^{mi} \subset Q^{hi}$, $q_0^{mi} \equiv q_0^{hi}, \Sigma^{mi} \subset \Sigma^{hi}$ and for which*: a) *for every state it follows that $\lambda^{mi}(q) = \lambda^{hi}(q)$. b) For every string $s \in D^i$ and every for every state $q \in Q^{mi} \subset Q^{hi}$* it follows that $\delta^{mi}(q,a) = \delta^{hi}(q,a)$. [Proof is in the online appendix.]

In the rest of this section we analyze the conditions under which the L^+ algorithm infers a correct model and when it fails to do so. Proposition 5.4 shows that when no state has the same output the L^+ algorithm is able to infer the correct automaton. In Proposition 5.4, for simplification, we adopt a representation of the transition function on a string s=$a_1a_2\ldots a_n$ $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$ in which q_1 is the state reached from q_0^{mi} after playing, sequentially $a_1a_2\ldots a_n$.

Proposition 5.4: *Let $D^i = \{\ \}$, $\phi^i = (s, \sigma)$ represent an example of the residual product automaton's behavior, and let s represent a string of actions such that $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$. Let the residual product automaton $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ be such that for every state $q_1, q_2 \in Q^{mi}$,*

$\lambda^{mi}(q_2) \neq \lambda^{mi}(q_1)$. *Under* L^+*, the inferred automaton,* P^i*, is equal to* M^i. [Proof is in the online appendix.]

As suggested by examples 4.1 and 4.3, for automata in which the same action is played in different states it may be impossible to infer the correct automaton using L^+. In Proposition 5.5 we generalize this finding.

Proposition 5.5: *Let the automaton* $M^i = (Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi})$ *be such that for some states* $q_1, q_2 \in Q^{mi}$, $\lambda^{mi}(q_1) = \lambda^{mi}(q_2)$. *Further let* s *represent a string of actions such that* $q_1 = \delta^{mi}(q_0^{mi}, s)$ *and* $s \in D^i$. *Under* L^+*, if there is at least one action* a *such that* $q_2 = \delta^{mi}(q_1, a)$ *and* $sa \notin D^i$ *then* $P^i \neq M^i$. [Proof is in the online appendix.]

We next give a sufficient condition for correct automata learning using the L^+. It follows from Proposition 5.5 that a simple way of disguising a strategy is to introduce a fake action after each correct state. In this case, the forward looking abilities of the L^+ are not sufficient to ensure that the correct automaton is revealed. Therefore, only by looking beyond the states required to build the *closed* and *consistent* model will an agent learn his opponents' strategies. The number of look-ahead steps necessary to infer the correct automaton is described in Proposition 5.6.

Proposition 5.6: *Let the automaton* $M^i = (Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi})$ *be such that R is size of the longer sequence of actions starting in* q_0^{mi} *without encountering a state with an outcome not yet observed since state* q_0^{mi}. *If the learner uses a test set composed of strings representing every possible sequence of actions with at least length R, then the residual product automaton will always be revealed, i.e.,* $P^i = M_1^i$. [Proof is in the online appendix.]

However, as shown by Proposition 5.7, the number of strings is an exponential function of the step size *R*. If the *R* is too large then it is not computationally possible to discover the correct automaton.

Proposition 5.7: *Let* $\#\Sigma$ *represent the number of actions. The number of strings in a set E with a look-ahead testing ability of R steps equals* $(\#\Sigma)^R$. *The total number of elements in table T equals* $(\#\Sigma)\frac{1-(\#\Sigma)^R}{1-(\#\Sigma)}$. [Proof is in the online appendix.]

Next, we apply the L^+ algorithm to the analysis of the product differentiation problem.

6. An Application of the L^+ Algorithm to the Product Differentiation Problem

We illustrate the L^+ algorithm in the product differentiation problem (e.g., Hsu and Wang, 2004; Mallik and Chhajed, 2006) in which a learner attempts to infer the residual product automaton of a system with three and five other agents (respectively in two different sets of experiments) when deciding how to position his product (high, low, and niche segments). At each stage, the outcome used in revising decisions is profit (classified as high, normal or low).

The behavior function of each one of these automata was generated randomly, i.e., each state was assigned a random segment at each stage. We have generated 15000 different problems for the experiments with five agents and 8800 different problems for the experiment with three agents. We have evaluated the complexity of the inferred automata and the time required to infer these automata (in computer seconds, standardize to 100 as the maximum in each set of experiments). (The model was coded in SICStus Prolog 4.0.) In Figures 1 and 2 we represent the relationship between the number of states in each residual product automaton and the computational time required to infer the model. The maximum number of states was about 5000 and 1750 for the experiments with five and three agents, respectively.

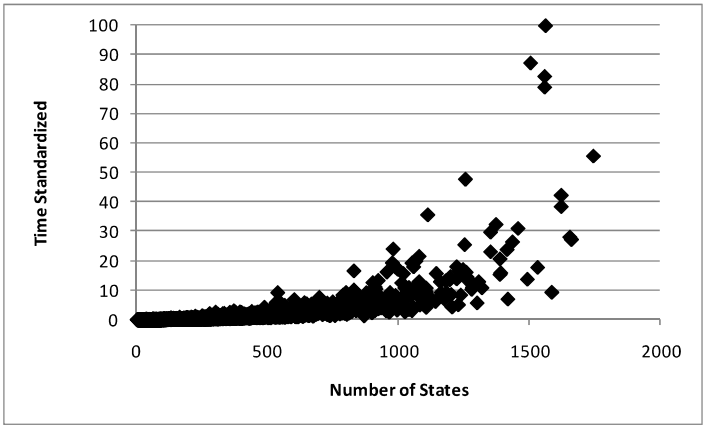

Figure 1: Computation time in the experiments with three agents (standardized to maximum 100).

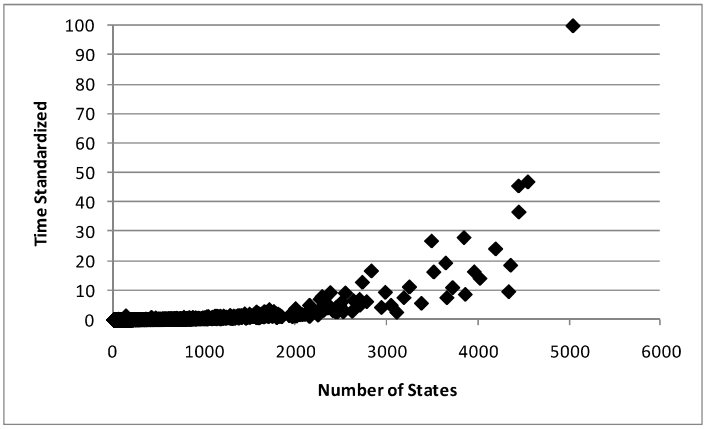

Figure 2: Computation time in the experiments with five agents (standardized to maximum 100).

The volatility of the computational time increases with the number of states (this is due to a greater uncertainty in the complexity of the automata). Moreover, there is a pattern in the worst case computational time. In Figure 3 we plot the worst case computational times for both sets of experiments.

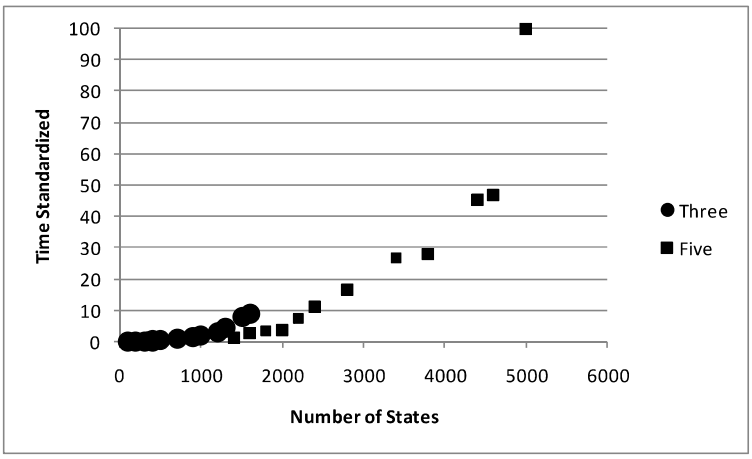

Figure 3: Maximum computation time in the experiments with three and five agents.

The worst case computational time is polynomial on the number of states: a regression of the worst time on the number of states can be fitted as a polynomial of order three and five, respectively, to the experiments with the same number of agents, both R^2-adjusted are about 99%. This means that the computational time is exponential on the number of agents. We can, therefore, conclude that the US-L^*, the L^+, and the L^* algorithms can only be applied in problems with a relatively small number of agents, or in problems with a large number of very simple agents (as the overall complexity of each model depends both on the number of agents and on their complexity).

7. Conclusions

If we abandon perfect rationality which model of learning can we use to model organizational behavior? We propose the use of a modification of Angluin's (1987) L^* algorithm, the L^+, which enables the modeling of quasi-perfect learning. The main contribution of this article is to study the properties of quasi-perfect learning, analyzing the conditions under which it fails or succeeds to infer the correct model.

Through the analysis of the properties of the L^+ we show that: there is at least one possible automaton capable of generating any data set; a close and consistent model of an automaton may not be a complete description of the correct automaton; if the same action is played in different states the L^+ may fail to learn the correct automaton; in the context of the repeated prisoners' dilemma

the L^+ may also fail to learn the correct automaton, which suggests that only perfect learners can collect the necessary information to play the Nash equilibrium. Furthermore, we show that a sufficient condition for the L^+ to approximate perfect learning (the L^*) is to consider a large enough number of look-ahead steps. Moreover, we show that closure of an automaton is a necessary condition for optimal behavior. (The L^* and the L^+ algorithms infer closed automata but this is not the case of the US-L^*.) Finally, we have tested empirically, in the product differentiation problem, that the computation time for the L^+ algorithm is polynomial on the number of states, but exponential on the number of agents (the same applies to the US-L* and L* algorithms), which suggests that these algorithms can only be used in systems with a small number of agents or with a large number of very simple agents.

References

Angluin, D. 1987. "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, pp. 87-106.

Banks, J. S., and R. K. Sundaram, 1990. "Repeated Games, Finite Automata, and Complexity," *Games and Economic Behavior*, vol. 2, pp. 97-117.

Ben-Porath, E. 1990. "The Complexity of Computing a Best Response Automaton in Repeated Games with Mixed Strategies," *Games and Economic Behavior*, vol. 2: pp. 1-12.

Bertsekas, D. P., and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Mass, 1996.

Bunn, D. W., and F. S. Oliveira. 2008. "Modeling the Impact of Market Interventions on the Strategic Evolution of Electricity Markets," *Operations Research*, 56 (5): 1116-1130.

Bunn, D. W., and F. S. Oliveira. 2007. "Agent-Based Analysis of Technological Diversification and Specialisation in Electricity Markets," *European Journal of Operational Research,* 181 (3): 1265-1278.

Cai, G, and N. Kock. 2009. "An Evolutionary Game Theoretic Perspective on e-collaboration: The collaboration effort and media relativeness**,**" *European Journal of Operational Research,* 194: 821-833.

Carmel, D., and S. Markovitch, 1999. "Exploration Strategies for Model-based Learning in Multi-agent Systems," *Autonomous Agents and Multi-agent Systems*, 2: 141 – 172.

Carmel, D., and S. Markovitch, 1998. “Pruning algorithms for multi-model adversary search,” *Artificial Intelligence,* 99 (2): 325 – 355.

Carmel, D., and S. Markovitch, “Learning models of intelligent agents”, *in Proceedings of The Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996, pp. 62-67.

Donkers, H.H.L.M., H. van den Herik, and J.W.H.M. Uiterwijk, 2005. “Selecting evaluation functions in Opponent-Model Search,” *Theoretical Computer Science* 349 (2): 245-267.

Freund, Y. , M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, and R. E. Schapire, “Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries,” *in Proceedings of the 36th IEEE Symposium on the Foundations of Computer Science*, 1995, pp. 332-341.

Gérard, P., J.-A. Meyer, and O. Sigaud. 2005. “Combining latent learning with dynamic programming in the modular anticipatory classifier system,” *European Journal of Operational Research*, 160: 614-637.

Gilboa, I. 1988. “The Complexity of Computing Best-Response Automata in Repeated Games,” *Journal of Economic Theory*, vol. 45, pp. 342-352.

Gold, E. M., 1978. “Complexity of automaton identification from given data,” *Information and Control*, vol. 37, pp. 302-320.

Gmytrasiewicz, P. J., and C. L. Lisetti, “Emotions and personality in agent design and modeling”, in *Game Theory and Decision Theory in Agent-Based Systems*, Ed. S. Parsons, P. J. Gmytrasiewicz, and M. Wooldridge, 2002, pp. 81-95.

Gusak, O., T. Dayar, J.-M. Fourneau. 2003. “Lumpable continuous-time stochastic automata networks,” *European Journal of Operational Research*, 148: 436-451.

Hopcroft, J. E., and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Mass, 1979.

Hsu, H.-M., and W.-P. Wang. 2004. “Dynamic programming for delayed product differentiation,” *European Journal of Operational Research*, 156: 183–193.

Mallik S., and D. Chhajed. 2006. “Optimal temporal product introduction strategies under valuation changes and learning,” *European Journal of Operational Research*, 172: 430–452.

Mor, Y. , C. V. Goldman, and J. S. Rosenschein, "Learn Your Opponent's Strategy (in Polynomial Time)!," *in Lecture Notes in Artificial Intelligence*, G. Weiss and S. Sen, Eds., Vol. 1042, Springer Verlag, 1996.

Oliveira, F. S. 2009. "Bottom-up Design of Strategic Options as Finite Automata," *Computational management Science*, forthcoming.

Papadimitriou, C. H. 1992. "On Agents with a Bounded Number of States," *Games and Economic Behavior*, vol. 4, pp. 122 – 131.

Pendharkar, P. C., 2007. "The Theory and experiments of designing cooperative intelligent systems," *Decision Support Systems* 43 (2007): 1014-1030.

Piccione, M., 1992. "Finite Automata Equilibria with Discounting," *Journal of Economic Theory,* vol. 56, pp. 180-193.

Rubinstein, A. 1986. "Finite automata play the repeated prisoner's dilemma," *Journal of Economic Theory*, vol. 39, pp. 83-96.

Sbeity, I., L. Brenner, B. Plateau, W. J. Stewart. 2008. "Phase-type distributions in stochastic automata networks," *European Journal of Operational Research*, 186: 1008-1028.

Shen, W. M., 1994. "Learning deterministic finite automata using local distinguishing experiments," *in Computational Learning Theory and Natural Learning Systems*, T. Petsche, S. Judd and S. Hanson, MIT Press.

Stewart, W. J., K. Atif, and B. Plateau. 1995. "The numerical solution of stochastic automata networks," *European Journal of Operational Research*, 86: 503-525.

Uysal, E., and T. Dayar. 1998. "Iterative methods based on splitting for stochastic automata networks," *European Journal of Operational Research*, 110: 166-186.

van Ackere, A., E. R. Larsen. 2004. "Self-organizing behaviour in the presence of negative externalities: A conceptual model of commuter choice," *European Journal of Operational Research,* 157: 501–513

Vassileva, J., and C. Mudgal, "Bilateral Negotiations with Incomplete and Uncertain Information", in *Game Theory and Decision Theory in Agent-Based Systems*, Ed. S. Parsons, P. J. Gmytrasiewicz, and M. Wooldridge, 2002, 337-353.

Villeneuve, D, and G. Desaulniers. 2005. "The shortest path problem with forbidden paths," *European Journal of Operational Research*, 165: 97-107.

Wainer, J., P. R. Ferreira, E. R. Constantino 2007. "Scheduling meetings through multi-agent negotiations," *Decision Support Systems* 44: 285-297.

Weiss, G., "Adaptation and Learning in Multi-Agent Systems: Some Remarks and a Bibliography," in *Adaptation and Learning in Multiagent Systems*. G. Weiss and S. Sen, Ed. Springer, 1995: 1-21.

Online Appendix

Proposition 5.1: *Let equation* $V_{i,t}^{*}\left(q_t^i\right)=u_t^i\left(z_t^i\right)+\rho_i V_{i,t+1}^{*}\left(\delta^i\left(q_t^i,a_t^i\right)\right)$ *represent the present value of agent i's utility. A necessary condition for the computation of the optimal strategy is the capability of an agent to infer a closed and consistent model (*P^i*) of the residual product automaton.*

Proof. Closure: Assume that the model inferred by agent *i* is not closed, i.e., for some states $q_o^{pi}\in Q^{pi}$ this agent cannot forecast the output of some of his actions. Assume also that a_t^i is the action that this agent *i* cannot forecast. Consequently, it is impossible for agent *i* to compute the transition function $q_{t+1}^{pi}=\delta^{pi}\left(q_{o,t}^{pi},a_t^i\right)$, and therefore it is not possible to compute $V_{i,t}^{*}$. Consistency: Assume that the model inferred by an agent *i* is not consistent, i.e., for some states $q_o^{pi}\in Q^{pi}$ the same action a_t^i implies different transitions, e.g., $q_{1,t+1}^{pi}=\delta^{pi}\left(q_{o,t}^{pi},a_t^i\right)$ and $q_{2,t+1}^{pi}=\delta^{pi}\left(q_{o,t}^{pi},a_t^i\right)$. Once again, the function $V_{i,t}^{*}$ is ill defined. Q.E.D.

Proposition 5.2. *Under* L^{+}*, for any set of data* D^i *there is at least one closed and consistent automaton capable of generating* D^i.

Proof. Under L^{+}, for a given set of prefix-closed set of strings D^i, the consistency loop classifies the different strings into classes. Within each class an action always generates the same outcome. Therefore, we can define a transition function $\delta^{pi}:Q^{pi}\times\Sigma^i\rightarrow Q^{pi}$ such that for any class $q_k^{pi}\in Q^{pi}$ and any action $a_i\in\Sigma^i$ a unique transition is defined into another class $q_{k+1}^{pi}\in Q^{pi}$. If there is at least one class in the observation table (S, E, T) such that a transition is not defined, i.e., if there is a class which belongs to $S\Sigma^i$ but not to *S*, then the closure loop computes the tran-

sitions δ^{pi} for any of such classes: the L^+ algorithm terminates when for each class there is a well defined transition. Moreover, in order for the automaton to be well defined we still need to show that the L^+ algorithm, for any set of prefix-closed set of strings D^i, correctly computes the behavioral function λ^{pi}, mapping a state into actions $\left(\lambda^{pi}: Q^{pi} \rightarrow Z^i\right)$. This function is computed by picking up the value in the first column of table *T*, for each class in *S*. Q.E.D.

Proposition 5.3. *Let* $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ *represent a closed and consistent automaton compatible with the data* D^i. *It is always possible to find an automaton* $H^i = \left(Q^{hi}, q_0^{hi}, \Sigma^i, Z^i, \delta^{hi}, \lambda^{hi}\right)$, *also closed, consistent and compatible with the data* D^i, *such that* $Q^{mi} \subset Q^{hi}$, $q_0^{mi} \equiv q_0^{hi}, \Sigma^{mi} \subset \Sigma^{hi}$ *and for which*: (a) *for every state it follows that* $\lambda^{mi}(q) = \lambda^{hi}(q)$. (*b) For every string* $s \in D^i$ *and every for every state* $q \in Q^{mi} \subset Q^{hi}$ it follows that $\delta^{mi}(q,a) = \delta^{hi}(q,a)$.

Proof. This proof proceeds by analyzing a possible way of building $H^i = \left(Q^{hi}, q_0^{hi}, \Sigma^i, Z^i, \delta^{hi}, \lambda^{hi}\right)$ out of $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$. Assume that M^i is known and make $q_0^{mi} \equiv q_0^{hi}$. As a first step, choose a new action a^{hi} not in Σ^{mi} and a new state q^{hi} not in Q^{mi} and build the state and action sets for automaton H^i as: $Q^{hi} = Q^{mi} \cup \left\{q^{hi}\right\}$ and $\Sigma^{hi} = \Sigma^{mi} \cup \left\{a^{hi}\right\}$. Consequently $Q^{mi} \subset Q^{hi}$ and $\Sigma^{mi} \subset \Sigma^{hi}$. Moreover, for each state $q \in Q^{mi}$ compute the transition for the new action a^{hi}: $q_1 = \delta^{hi}\left(q, a^{hi}\right)$ such that for some states $q_1 = q^{hi}$. Then, for the new state q^{hi} specify the behavioral function, $\lambda^{hi}\left(q^{hi}\right)$ and for every possible action *a* specify and the transition function, $q_1 = \delta^{hi}\left(q^{hi}, a\right)$, in which q_1 represents a state in Q^{hi} .In conclusion, none of the initial transitions or behaviors is altered and $\lambda^{mi}(q) = \lambda^{hi}(q)$, $\delta^{mi}(q,a) = \delta^{hi}(q,a)$ and $Q^{mi} \subset Q^{hi}$ and $\Sigma^{mi} \subset \Sigma^{hi}$. Q.E.D.

Proposition 5.4: *Let* $D^i = \{\ \}$, $\phi^i = (s, \sigma)$ *represent an example of the residual product automaton's behavior, and let s represent a string of actions such that* $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$. *Let the residual*

product automaton $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ *be such that for every state* $q_1, q_2 \in Q^{mi}$, $\lambda^{mi}(q_2) \neq \lambda^{mi}(q_1)$. *Under L*$^+$*, the inferred automaton, P*^{i}*, is equal to M*^{i}.

Proof. As $D^i = \{\ \}$ for any initial example, under L$^+$, the initial observation table (S, E, T) is consistent. Moreover, for every state $q_1, q_2 \in Q^{mi}$, there is at least an $a \in \Sigma^i$, such that another state is reached from the initial one, i.e., $q_2 = \left(\delta^{mi}(q_1, a)\right)$. Consequently, from the definition of closure, it follows that the table (S, E, T) is not closed. Furthermore, in order to close the table, the string $s \in S\Sigma^i$ such $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$ is transferred from $S\Sigma^i$ into *S*, and then expanded for closure, i.e., $\forall \sigma \in \Sigma^i$, $S\Sigma^i \leftarrow \left(S\Sigma^i \setminus \{s\}\right) \cup \{s\sigma\}$. Once again, for as long as $|S| < \left|Q^{mi}\right|$ there is at least a class $q_1 \in S$ such that for some $a \in \Sigma^i$, we have $q_2 = \delta^{mi}(q_1, a)$ and $\lambda^{mi}(q_2) \neq \lambda^{mi}(q_1)$ and $q_2 \notin S$, as for any new reached state q_2, $\lambda^{mi}(q_2)$ is not in the current set of outputs used by the automaton. In conclusion, only when for every class in *S* a closed and consistent transition is computed (and therefore every row in $S\Sigma^i$ is represented by a class in *S*) does the algorithm terminate. This implies that the residual product automaton has been inferred. Q.E.D.

Proposition 5.5: *Let the automaton* $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ *be such that for some states* $q_1, q_2 \in Q^{mi}$, $\lambda^{mi}(q_1) = \lambda^{mi}(q_2)$. *Further let s represent a string of actions such that* $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$ *and* $s \in D^i$. *Under L*$^+$*, if there is at least one action a such that* $q_2 = \delta^{mi}(q_1, a)$ *and* $sa \notin D^i$ *then P*^{i} $\neq$ *M*^{i}.

Proof. Let $s \in D^i$ and $sa \notin D^i$, under L$^+$, within the consistency loop, string *s* is extended for all possible actions (one of them being action *a*). Then, as $q_1 = \delta^{mi}\left(q_0^{mi}, s\right)$, $q_2 = \delta^{mi}(q_1, a)$ and $\lambda^{mi}(q_1) = \lambda^{mi}(q_2)$ the row for string *sa* is wrongly classified in the same class as string *s*, (even though they belong to different states) and therefore the automaton is incorrectly inferred. Q.E.D.

Proposition 5.6: *Let the automaton* $M^i = \left(Q^{mi}, q_0^{mi}, \Sigma^i, Z^i, \delta^{mi}, \lambda^{mi}\right)$ *be such that R is size of the longer sequence of actions starting in* q_0^{mi} *without encountering a state with an outcome not yet observed since state* q_0^{mi}. *If the learner uses a test set composed of strings representing every possible sequence of actions with at least length R, then the residual product automaton will always be revealed, i.e.,* $P^i = M^i$.

Proof. Let $s \in D^i$, for all *j* let $sa^j \notin D^i$, and let *E* have all the strings representing every possible sequence of actions with at least length *R*. Then, under L^+, within the consistency loop, string *s* is tested for all possible sequence of actions of length *R* (one of them being the string $a^j\, a^{j+1} \ldots a^{j+(R-1)}\, a^{j+R}$). Then, as $q^j = \delta^{mi}\left(q_0, s\right)$, and for any *k*, $q^{j+k} = \delta^{mi}\left(q^{j+k-1}, a^{j+k-1}\right)$ and as the test with string $a^j\, a^{j+1} \ldots a^{j+(R-1)}\, a^{j+R}$ will always encounter an outcome not yet in the set *S*, in order to close the inference table a new state is discovered after every move. As the algorithm is checking consistency *R* steps ahead the correct automaton is always revealed. Q.E.D.

Proposition 5.7: Let $\#\Sigma$ *represent the number of actions.* The number of strings in a set *E* with a look-ahead testing ability of *R steps equals* $\left(\#\Sigma\right)^R$. *Moreover, the total number of elements in table T equals* $\left(\#\Sigma\right)\frac{1-\left(\#\Sigma\right)^R}{1-\left(\#\Sigma\right)}$

Proof. In a string of size *R* each one of its components can assume $\#\Sigma$ possible values. Therefore, there are $\left(\#\Sigma\right)^R$ possible strings of length *R*. As the set of tests *E* is suffix closed, it also contains all the possible strings of size *R-1*, *R-2*, …, *1*. Therefore, in order to complete table *T* we need to compute the outcome of each one of these strings. The number of these strings is equal to $\left(\#\Sigma\right)^R + \left(\#\Sigma\right)^{R-1} + \left(\#\Sigma\right)^{R-2} + \ldots + \left(\#\Sigma\right)^1$ which is equivalent to $\left(\#\Sigma\right)\frac{1-\left(\#\Sigma\right)^R}{1-\left(\#\Sigma\right)}$. Q.E.D.