Learning Prime Implicant Conditions From Interpretation Transition

Tony Ribeiro, Katsumi Inoue

Learning Prime Implicant Conditions From Interpretation Transition

Tony Ribeiro[1] and Katsumi Inoue[1,2]

[1] The Graduate University for Advanced Studies (Sokendai),
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
tony_ribeiro@nii.ac.jp,
[2] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan,
inoue@nii.ac.jp

Abstract. In a previous work we proposed a framework for learning normal logic programs from transitions of interpretations. Given a set of pairs of interpretations (I, J) such that $J = T_P(I)$, where T_P is the immediate consequence operator, we infer the program P. Here we propose a new learning approach that is more efficient in terms of output quality. This new approach relies on specialization in place of generalization. It generates hypotheses by *specialization* from the most general clauses until no negative transition is covered. Contrary to previous approaches, the output of this method does not depend on variables/transitions ordering. The new method guarantees that the learned rules are minimal, that is, the body of each rule constitutes a prime implicant to infer the head.

Keywords: dynamical systems, Boolean networks, attractors, supported models, learning from interpretation, Inductive Logic Programming

1 Introduction

In recent years, there has been a notable interest in the field of Inductive Logic Programming (ILP) to learn from system state transitions as part of a wider interest in learning the dynamics of systems [14, 2, 4, 8]. Learning system dynamics has many applications in multi-agent systems, robotics and bioinformatics alike. Knowledge of system dynamics can be used by agents and robots for planning and scheduling. In bioinformatics, learning the dynamics of biological systems can correspond to the identification of the influence of genes and can help to design more efficient drugs. In some previous works, state transition systems are represented with logic programs [6, 9], in which the state of the world is represented by an Herbrand interpretation and the dynamics that rule the environment changes are represented by a logic program P. The rules in P specify the next state of the world as an Herbrand interpretation through the *immediate consequence operator* (also called the T_P *operator*) [18, 1]. With such a background, Inoue *et al.* [8] have recently proposed a framework to learn logic

programs from traces of interpretation transitions (LFIT). The learning setting of this framework is as follows. We are given a set of pairs of Herbrand interpretations (I, J) as positive examples such that $J = T_P(I)$, and the goal is to induce a *normal logic program* (NLP) P that realizes the given transition relations. In [8], the authors showed one of the possible usages of LFIT: **LF1T**, *learning from 1-step transitions*. In that paper, an algorithm is proposed to iteratively learn an NLP that realizes the dynamics of the system by considering step transitions one by one. The iterative character of **LF1T** has applications in bioinformatics, cellular automata, multi-agent systems and robotics.

In this paper, our main concern is the minimality of the rules and the NLPs learned by LF1T. Our goal is to learn all minimal conditions that imply a variable to be true in the next state, e.g. all prime implicant conditions. In bioinformatics, for a gene regulatory network, it corresponds to all minimal conditions for a gene to be activated/inhibited. It can be easier and faster to perform model checking on Boolean networks represented by a compact NLP than the set of all state transitions. Knowing the minimal conditions required to perform the desired state transitions, a robot can optimize its actions to achieve its goals with less energy consumption. From a technical point of view, for the sake of memory usage and reasoning time, a small NLP could also be preferred in multi-agent and robotics applications. For this purpose, we propose a new version of the **LF1T** algorithm based on specialization. Specialization is usually considered the dual of generalization in ILP [13, 11, 12]. Where generalization occurs when a hypothesis does not explain a positive example, specialization is used to refine a hypothesis that implies a negative example.

In [7], prime implicants are defined for DNF formula as follows: a clause C, implicant of a formula ϕ, is prime if and only if none of its proper subset $S \subset C$ is an implicant of ϕ. In this work, explanatory induction is considered, while in our approach prime implicants are defined in the LFIT framework. Knowing the Boolean functions, prime implicants could be computed by Tisons consensus method [17] and its variants [10]. The novelty of our approach, is that we compute prime implicants incrementally during the learning of the Boolean function. In [8, 16], **LF1T** uses resolution techniques to generalize rules and reduces the size of the output NLP. This technique generates hypotheses by *generalization* from the most specific clauses until every positive transitions are covered. Compared to previous **LF1T** algorithms, the novelty of our new approach is that it generates hypotheses by *specialization* from the most general clauses until no negative transition is covered. The main weak point of the previous **LF1T** algorithms is that the output NLPs depends on variable/transition ordering. Our new method guarantees that the NLPs learned contain only minimal conditions for a variable to be true in the next state. Study of the computational complexity of our new method shows that it remains equivalent to the previous version of LF1T. Using examples from the biological literature, we show through experimental results that our specialization method can compete with the previous versions of **LF1T** in practice. We provide all proofs of theorems in the appendix.

2 Background

In this section we recall some preliminaries of logic programming. We consider a first-order language and denote the Herbrand base (the set of all ground atoms) as $\mathcal{B}$. A (*normal*) *logic program* (NLP) is a set of *rules* of the form

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \tag{1}$$

where A and A_i's are atoms ($n \geq m \geq 0$). For any rule R of the form (1), the atom A is called the *head* of R and is denoted as $h(R)$, and the conjunction to the right of $\leftarrow$ is called the *body* of R. We represent the set of literals in the body of R of the form (1) as $b(R) = \{A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n\}$, and the atoms appearing in the body of R positively and negatively as $b^+(R) = \{A_1, \ldots, A_m\}$ and $b^-(R) = \{A_{m+1}, \ldots, A_n\}$, respectively. The set of ground instances of all rules in a logic program P is denoted as $ground(P)$.

The Herbrand Base of a program P, denoted by $\mathcal{B}$, is the set of all atoms in the language of P. An *interpretation* is a subset of $\mathcal{B}$. If an interpretation is the empty set, it is denoted by ϵ. An interpretation I is a model of a program P if $b^+(R) \subseteq I$ and $b^-(R) \cap I = \emptyset$ imply $h(R) \in I$ for every rule R in P. For a logic program P and an Herbrand interpretation I, the *immediate consequence operator* (or T_P *operator*) [1] is the mapping $T_P : 2^{\mathcal{B}} \rightarrow 2^{\mathcal{B}}$:

$$T_P(I) = \{\, h(R) \mid R \in ground(P),\, b^+(R) \subseteq I,\, b^-(R) \cap I = \emptyset\}. \tag{2}$$

In the rest of this paper, we only consider rules of the form (1). To simplify the discussion we will just use the term rule.

Definition 1 (Subsumption). *Let R_1 and R_2 be two rules. If $h(R_1) = h(R_2)$ and $b(R_1) \subseteq b(R_2)$ then R_1* **subsumes** *R_2. If $b(R_1) \subset b(R_2)$ then R_1 is* **more general** *than R_2 and R_2 is* **more specific** *than R_1. Let S be a set of rules and R be a rule. If there exists a rule $R' \in S$ that subsumes R then S* **subsumes** *R. This also holds for normal logic program since a NLP is a set of rules.*

In ILP, search mainly relies on generalization and specialization that are dual notions. Generalization is usually considered an induction operation, and specialization a deduction operation. In [13], the author define the minimality and maximality of the generalization and specialization operations as follows.

Definition 2 (Generalization operator [13]). *A generalization operator maps a conjunction of clauses S onto a set of minimal generalizations of S. A* **minimal generalization** *G of S is a generalization of S such that S is not a generalization of G, and there is no generalization G' of S such that G is a generalization of G'.*

Definition 3 (Specialization operator [13]). *A specialization operator maps a conjunction of clauses G onto a set of maximal specializations of G. A* **maximal specialization** *S of G is a specialization of G such that G is not a specialization of S, and there is no specialization S' of G such that S is a specialization of S'.*

The body of a rule of the form (1) can be considered as a clause, so that, the Definition 2 and 3 can also be used to compare the body of two rules.

3 Learning from 1-Step Transitions

LF1T is an *any time algorithm* that takes a set of one-step state transitions E as input. These one-step state transitions can be regarded as positive examples. From these transitions the algorithm learns a logic program P that represents the dynamics of E. To perform this learning process we can iteratively consider one-step transitions. Figure 1 represents a Boolean nework with its corresponding state transition diagram. Given this state transition diagram as input, **LF1T** can learn the Boolean network N_1.

In *LF1T*, the Herbrand base $\mathcal{B}$ is assumed to be finite. In the input E, a state transition is represented by a pair of Herbrand interpretations. The output of *LF1T* is an NLP that realizes all state transitions of E.

Learning from 1-Step Transitions (LF1T)

Input: $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$: (positive) examples/observations
Output: An NLP P such that $J = T_P(I)$ holds for any $(I, J) \in E$.

To construct an NLP with **LF1T** we use a bottom-up method that generates hypotheses by *generalization* from the most specific clauses or examples until every positive example is covered. **LF1T** first constructs the most specific rule R^I_A for each positive literal A appearing in $J = T_P(I)$ for each $(I, J) \in E$:

$$R^I_A := (A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in \mathcal{B} \setminus I} \neg C_j)$$

It is important here that *we do not construct any rule to make a literal false.* For instance, from the state transition (qr, pr) **LF1T** will learn only two rules: $p \leftarrow \neg p \wedge q \wedge r$ and $r \leftarrow \neg p \wedge q \wedge r$. The rule R^I_A is then possibly generalized when another transition from E makes A true, which is computed by several generalization methods.

The two generalization methods considered in [8] are based on *resolution.* In [8], naïve and ground resolutions are defined between two ground rules as follows. Let R_1, R_2 be two ground rules and l be a literal such that $h(R_1) = h(R_2)$,

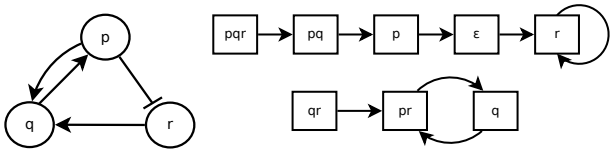

Fig. 1. A Boolean Network N_1(left) and its state transition diagram (right)

$l \in b(R_1)$ and $\bar{l} \in b(R_2)$. If $(b(R_2) \setminus \{\bar{l}\}) \subseteq (b(R_1) \setminus \{l\})$ then the *ground resolution of R_1 and R_2 (upon l)* is defined as

$$res(R_1, R_2) = \left(h(R_1) \leftarrow \bigwedge_{L_i \in b(R_1) \setminus \{l\}} L_i \right). \tag{3}$$

In particular, if $(b(R_2) \setminus \{\bar{l}\}) = (b(R_1) \setminus \{l\})$ then the ground resolution is called the *naïve resolution of R_1 and R_2 (upon l)*. In this particular case, the rules R_1 and R_2 are said to be *complementary* to each other *with respect to l*.

Both naïve resolution and ground resolution can be used as generalization methods of ground rules. For two ground rules R_1 and R_2, the naïve resolution $res(R_1, R_2)$ subsumes both R_1 and R_2, but the non-naïve ground resolution subsumes R_1 only.

Definition 4 (Consistency). *Let R be a rule and E be a set of state transitions. R is* **consistent** *with E if $\forall (I, J) \in E$ when $b(R) \subseteq I$ then $h(R) \in J$. Let P be a NLP. P is* **consistent** *with E if $\forall (I, J) \in E$, $\forall R' \in P$, when $b(R') \subseteq I$ then $h(R') \in J$.*

Ground and naïve resolutions can be used to learn a ground NLP. Both methods keep the consistency of the learned rules. For example, let us consider the three rules: $R_1 = (p \leftarrow q \wedge r)$, $R_2 = (p \leftarrow \neg q \wedge r)$, $R_3 = (p \leftarrow \neg q)$, and their resolvent: $res(R_1, R_2) = res(R_1, R_3) = (p \leftarrow r)$. R_1 and R_2 are complementary with respect to q. Both R_1 and R_2 can be generalized by the naïve resolution of them because $res(R_1, R_2)$ subsumes both R_1 and R_2. On the other hand, the ground resolution $res(R_1, R_3)$ of R_1 and R_3 is equivalent to $res(R_1, R_2)$. However, $res(R_1, R_3)$ subsumes R_1 but does not subsume R_3.

LF1T with Naïve Resolution: In the first implementation of **LF1T** of [8], naïve resolution is used as a least generalization method. This method is particularly intuitive from the ILP viewpoint, since each generalization is performed based on a least generalization operator. In [8], it is shown that for two complementary ground rules R_1 and R_2, the naïve resolution of R_1 and R_2 is the least generalization [15] of them, that is, $lg(R_1, R_2) = res(R_1, R_2)$. When naïve resolution is used, **LF1T** needs an auxiliary set P_{old} of rules to globally store subsumed rules, which increases monotonically. P_{old} is set to be $\emptyset$ at first. When a generated rule R is newly added **LF1T** searches a rule $R' \in P \cup P_{old}$ such that $h(R') = h(R)$ and $b(R)$ and $b(R')$ differ in the sign of only one literal l. If there is no such a rule R', then R is just added to P; otherwise, R and R' are added to P_{old} and then $res(R, R')$ is added to P.

LF1T with Ground Resolution: Using naïve resolution, $P \cup P_{old}$ possibly contains all patterns of rules constructed from the Herbrand base $\mathcal{B}$ in their bodies. In the second implementation of **LF1T** of [8], ground resolution is used as an alternative generalization method. This replacement of resolution leads to a lot of computational gains since the use of P_{old} is not necessary any more: all generalized rules obtained from $P \cup P_{old}$ by naïve resolution can be obtained using ground resolution on P. By Theorem 3 of [8], using the naïve version,

the memory use of the **LF1T** algorithm is bounded by $O(n \cdot 3^n)$, and the time complexity of learning is bounded by $O(n^2 \cdot 9^n)$, where $n = |\mathcal{B}|$. On the other hand, with ground resolution, the memory use is bounded by $O(2^n)$, which is the maximum size of P, and the time complexity is bounded by $O(4^n)$. Given the set E of complete state transitions, which has the size $O(2^n)$, the complexity of $\mathbf{LF1T}(E, \emptyset)$ with ground resolution is bounded by $O(|E|^2)$. On the other hand, the worst-case complexity of learning with naïve resolution is $O(n^2 \cdot |E|^{4.5})$.

4 Learning Prime Implicant Conditions

In this section, we use the notion of prime implicant to define minimality of NLP. We consider that the NLP learn by **LF1T** is minimal if the body of each rule constitutes a prime implicant to infer the head.

Definition 5 (Prime Implicant Condition). *Let R be a rule and E a set of state transitions such that R is consistent with E. $b(R)$ is a* **prime implicant condition** *of $h(R)$ for E if there does not exist another rule R' consistent with E such that R' subsumes R. Let P be a NLP such that $P \cup \{R\} \equiv P$: all models of $P \cup \{R\}$ are models of P and vice versa. $b(R)$ is a* **prime implicant condition** *of $h(R)$ for P if there does not exist another rule R' such that $P \cup \{R'\} \equiv P$ and R' subsumes R.*

Definition 6 (Prime Rule). *Let R be a rule and E a set of state transitions such that R is consistent with E. Let P be a NLP such that $P \cup \{R\} \equiv P$. R is a* **prime rule** *of E (resp. P) if $b(R)$ is a prime implicant condition of $h(R)$ for E (resp. P). For any atom p the* **most general prime rule** *for p is the rule with an empty body ($p \leftarrow$) that states that p is always true in the next state.*

Example 1. Let R_1, R_2 and R_3 be three rules and E be the set of state transitions of Figure 1 as follows: $R_1 = p \leftarrow p \wedge q \wedge r$, $R_2 = p \leftarrow p \wedge q$, $R_3 = p \leftarrow q$ The only rule more general than R_3 is $R' = p.$, but R' is not consistent with $(p, \epsilon) \in E$ so that R_3 is a prime rule for E. Since R_3 subsumes both R_1 and R_2, they are not prime rules of E. Let P be the NLP $\{p \leftarrow p, q \leftarrow p \wedge r, r \leftarrow \neg p\}$, R_3 is a prime rule of P because P realizes E and R_3 is minimal for E.

Definition 7 (Prime NLP). *Let P be an NLP and E be the state transitions of P, P is a* **prime NLP** *for E if P realizes E and all rules of P are prime rule for E. We call the set of all prime rules of E the* **complete prime NLP** *of E.*

Example 2. Let R_1, R_2 and R_3 be three rules, E be the set of state transitions of Figure 1 and P an NLP as follows: $R_1 = p \leftarrow p \wedge q$, $R_2 = q \leftarrow p \wedge r$, $R_3 = r \leftarrow \neg p$ and $P = \{R_1\} \cup \{R_2\} \cup \{R_3\}$. Since R_1, R_2 and R_3 are prime rule for E, P the NLP formed of these three rules is a prime NLP of E. There does not exist any other prime rules for E, therefore P is also the complete prime NLP of E.

The complete prime NLP of a given set of state transitions E can naïvely be obtained by brute force search. Starting from the most general rules, that is,

fact rules, it suffices to generate all maximal specific specialization step by step and keep the first ones that are consistent with E. This method implies to check all state transitions for all possible rules that correspond to $O(n \times 3^n \times 2^n) = O(6^n)$ checking operations in the worst case for a Herbrand base of n variables. But it is also possible to do it by extending previous **LF1T** algorithm for the sake of complexity. Here we propose a simple extension of naïve (resp. ground) resolution. In previous algorithms, for each rule learned, only the first least generalization found is kept. Now we consider all possible least generalizations and define full naïve (resp. ground) resolution. **LF1T** with full naïve (resp. ground) resolution learn the complete prime NLP that realize the input state transitions.

Definition 8 (full naive resolution and full ground resolution). *Let R be a rule and P be a NLP. Let P_R be a set of rule of P such that, for all $R' \in P_R$, $h(R) = h(R')$ and for each R' there exists $l \in b(R)$, $(b(R') \setminus \{\bar{l}\}) = (b(R) \setminus \{l\})$ (resp. $(b(R') \setminus \{\bar{l}\}) \subseteq (b(R) \setminus \{l\})$). The* **full naïve (resp. ground) resolution** *of R by P is the set of all possible naïve (resp. ground) resolutions of R with the rules of P: $res_f(R, P) = \{res(R, R') | R' \in P_R\}$.*

Theorem 1 (Completeness and Soundness of full resolution). *Given a set E of pairs of interpretations,* **LF1T** *with full naïve (resp. ground) resolution is complete and sound for E.*

Theorem 2 (LF1T with full resolution learns complete prime NLP). *Given a set E of pairs of interpretations,* **LF1T** *with full naïve (resp. ground) resolution will learn the complete prime NLP that realizes E.*

4.1 Least Specialization for LF1T

Until now, to construct an NLP, **LF1T** relied on a bottom-up method that generates hypotheses by *generalization* from the most specific clauses or examples until every positive example is covered. This time we propose a new learning method that generate hypotheses by *specialization* from the most general rules until no negative example is covered. Learning by specialization ensures to output the most general valid hypothesis. that is similar to the notion of specialization we use here amoung body of rules with the same head. In ILP, refinement operators usually apply a substitution θ and add a set of literals to a clause [13]. Similarly, in our new algorithm, we refine rules by adding the negation of negative transitions into their body.

Definition 9 (Least specialization). *We call a* **maximal specialization** *of a rule R, a rule R_S if $h(R_S) = h(R)$ and $b(R_S)$ is a maximal specialization of $b(R)$. Let R_1 and R_2 be two rules such that $h(R_1) = h(R_2)$ and R_1 subsumes R_2, e.g. $b(R_1) \subseteq b(R_2)$. Let l_i be the i^{th} literal of $b(R_2)$, then the* **least specialization** *of R_1 over R_2 is as follows:*

$$ls(R_1, R_2) = \{h(R_1) \leftarrow (b(R_1) \wedge \neg b(R_2))\} = \{(h(R_1) \leftarrow (b(R_1) \wedge \overline{l_i}) | l_i \in b(R_2) \setminus b(R_1)\}$$

Let P be an NLP, R be a rule and S be the set of all rules of P that subsumes R. The **least specialization** *$ls(P, R)$ of P by R is as follow*

$$ls(P, R) = (P \setminus S) \cup (\bigcup_{R_P \in S} ls(R_P, R))$$

The least specialization of a rule R can be used to avoid the subsumption of another rule with a minimal reduction of the generality of R. The least specialization of an NLP P can be used to avoid the coverage of a negative transition with a minimal reduction of the generality of the rules of P.

Theorem 3 (Soundness of least specialization). *Let R_1, R_2 be two rules such that R_1 subsumes R_2. Let S_1 be the set of rules subsumed by R_1 and S_2 be the rules of S_1 that subsume R_2. The least specialization of R_1 by R_2 only subsumes the set of rules $S_1 \setminus S_2$. Let P be a NLP and R be a rule such that P subsumes R. Let S_P be the set of rules subsumed by P and S_R be the rules of S_P that subsume R. The least specialization of P by R only subsumes the set of rules $S_P \setminus S_R$.*

5 Algorithm

Now we present a new **LF1T** algorithm based on least specialization. The novelty of this approach is double: first it relies on specialization in place of generalization and most importantly, it guarantees that the output is the complete prime NLP that realize the input transitions, as shown by Theorem 5. Algorithm 1 shows the pseudo-code of **LF1T** with least specialization. Like in previous versions,

Algorithm 1 LF1T(E) : Learn the complete prime NLP P of E

INPUT: $\mathcal{B}$ a set of atoms and $E \subseteq 2^{\mathcal{B}} \times 2^{\mathcal{B}}$
OUTPUT: An NLP P such that $J = T_P(I)$ holds for any $(I, J) \in E$.

P a NLP
$P := \emptyset$
// Initialize P with the most general rules
for each $A \in \mathcal{B}$ **do**
    $P := P \cup \{A.\}$
// Specify P by interpretation of transitions
while $E \neq \emptyset$ **do**
    Pick $(I, J) \in E$; $E := E \setminus \{(I, J)\}$
    for each $A \in \mathcal{B}$ **do**
        if $A \notin J$ **then**
            $R_A^I := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg C_j$
            $P :=$ **Specialize**(P, R_A^I)
end while
return P

LF1T takes a set of state transitions E as input and outputs an NLP P that realizes E. To guarantee the minimality of the learned NLP, **LF1T** starts with an initial NLP $P_0^{\mathcal{B}}$ that is the most general complete prime NLP of the Herbrand base $\mathcal{B}$ of E, i.e. the NLP that contains only facts (lines 3-7): $P_0^{\mathcal{B}} = \{p. | p \in \mathcal{B}\}$. Then **LF1T** iteratively analyzes each transition $(I, J) \in E$ (lines 8-13).

For each variable A that **does not appear** in J, **LF1T** infers an **anti-rule** R_A^I (lines 11-12):

$$R_A^I := A \leftarrow \bigwedge_{B_i \in I} B_i \wedge \bigwedge_{C_j \in (\mathcal{B} \setminus I)} \neg C_j$$

A is in the head as it denotes a negative example. Then, **LF1T** uses least specialization to make P consistent with all R_A^I (line 12). Algorithm 2 shows in detail the pseudo code of this operation. **LF1T** first extracts all rules $R_P \in P$

Algorithm 2 **specialize**(P,R) : specify the NLP P to not subsume the rule R

```
1: INPUT: an NLP P and a rule R
2: OUTPUT: the maximal specific specialization of P that does not subsumes R.

3: conflicts : a set of rules
4: conflicts := ∅
   // Search rules that need to be specialized
5: for each rule R_P ∈ P do
6:     if b(R_P) ⊆ b(R) then
7:         conflicts := conflicts ∪ R_P
8:         P := P \ R_P
   // Revise the rules by least specialization
9: for each rule R_c ∈ conflicts do
10:    for each literal l ∈ b(R) do
11:        if l ∉ b(R_c) and l̄ ∉ b(R_c) then
12:            R'_c := (h(R_c) ← (b(R_c) ∪ l̄))
13:            if P does not subsumes R'_c then
14:                P := P \ all rules subsumed by R'_c
15:                P := P ∪ R'_c
16: return P
```

that subsume R_A^I (lines 3-8). It generates the least specialization of each R_P by generating a rule for each literal in R_A^I (lines 9-12). Each rule contains all literals of R_P plus the opposite of a literal in R_A^I so that R_A^I is not subsumed by that rule. Then **LF1T** adds in P all the generated rules that are not subsumed by P (line 13-15), so that P becomes consistent with the transition (I, J) and remains a complete prime NLP. When all transitions have been analyzed, **LF1T** outputs P that has become the complete prime NLP of E.

Table 1 shows the execution of **LF1T** with least specialization on step transitions of figure 1 where $pqr \rightarrow pq$ represents the state transition $(\{p, q, r\}, \{p, q\})$. Introduction of literal by least specialization is represented in bold and rules

Table 1. Execution of **LF1T** with least specialization on state transitions of figure 1

Initialization	$pqr \to pq$	$pq \to p$	$p \to \epsilon$	$\epsilon \to r$	$r \to r$
$p.$	$p.$	$p.$	$p \leftarrow \neg \boldsymbol{p}.$	$p \leftarrow q.$	$p \leftarrow q.$
$q.$	$q.$	$q \leftarrow \neg \boldsymbol{p}.$	$p \leftarrow \boldsymbol{q}.$	$p \leftarrow r.$	$p \leftarrow \boldsymbol{p} \wedge r.$
$r.$	$r \leftarrow \neg \boldsymbol{p}.$	$q \leftarrow \neg \boldsymbol{q}.$	$p \leftarrow \boldsymbol{r}.$	~~$p \leftarrow \neg p \wedge \boldsymbol{q}.$~~	~~$p \leftarrow \boldsymbol{q} \wedge r.$~~
	$r \leftarrow \neg \boldsymbol{q}.$	$q \leftarrow \boldsymbol{r}.$	$q \leftarrow \neg p.$	~~$p \leftarrow \neg p \wedge \boldsymbol{r}.$~~	$q \leftarrow \neg p \wedge q.$
	$r \leftarrow \neg \boldsymbol{r}.$	$r \leftarrow \neg p.$	$q \leftarrow r.$	$q \leftarrow r.$	$q \leftarrow \boldsymbol{p} \wedge r.$
		$r \leftarrow \neg q.$	~~$q \leftarrow \neg \boldsymbol{p} \wedge \neg q.$~~	$q \leftarrow \neg p \wedge \boldsymbol{q}.$	$q \leftarrow \boldsymbol{q} \wedge r.$
		~~$r \leftarrow \neg \boldsymbol{p} \wedge \neg r.$~~	~~$q \leftarrow \neg q \wedge \boldsymbol{r}.$~~	~~$q \leftarrow \neg p \wedge \boldsymbol{r}.$~~	$r \leftarrow \neg p.$
		~~$r \leftarrow \neg \boldsymbol{q} \wedge \neg r.$~~	$r \leftarrow \neg p.$	$r \leftarrow \neg p.$	$r \leftarrow \neg q \wedge r.$
			~~$r \leftarrow \neg \boldsymbol{p} \wedge \neg q.$~~	$r \leftarrow \neg q \wedge r.$	
			$r \leftarrow \neg q \wedge \boldsymbol{r}$.		

$qr \to pr$	$pr \to q$	$q \to pr$
$p \leftarrow q.$	$p \leftarrow q.$	$p \leftarrow q.$
$p \leftarrow p \wedge r.$	~~$p \leftarrow p \wedge \boldsymbol{q} \wedge r.$~~	$q \leftarrow p \wedge r.$
$q \leftarrow p \wedge r.$	$q \leftarrow p \wedge r.$	$r \leftarrow \neg p.$
$q \leftarrow \neg p \wedge q \wedge \neg \boldsymbol{r}.$	$q \leftarrow \neg p \wedge q \wedge \neg r.$	
~~$q \leftarrow \boldsymbol{p} \wedge q \wedge r.$~~	$r \leftarrow \neg p.$	$q \leftarrow \neg p \wedge q \wedge \neg r.$
$r \leftarrow \neg p.$	~~$r \leftarrow \neg \boldsymbol{p} \wedge \neg q \wedge r.$~~	is removed because
$r \leftarrow \neg q \wedge r.$		it cannot be specialized

that are subsumed after specialization are stroked. **LF1T** starts with the most general set of prime rules that can realize E, that is $P = \{p., q., r.\}$. From the transition (pqr, pq) **LF1T** infers the rule $r \leftarrow p \wedge q \wedge r$ that is subsumed by $r. \in P$. **LF1T** then replaces that rule by its least specialization: $ls(r., r \leftarrow p \wedge q \wedge r) = \{r \leftarrow \neg p, r \leftarrow \neg q, r \leftarrow \neg r\}$ Furthermore, P becomes consistent with (pqr, pq). From (pq, p) **LF1T** infers two rules: $q \leftarrow p \wedge q \wedge \neg r$ and $r \leftarrow p \wedge q \wedge \neg r$, that are respectively subsumed by $q.$ and $r \leftarrow \neg r$. The first rule, $q.$, is replaced by its least specialization: $\{q \leftarrow \neg p, q \leftarrow \neg q, q \leftarrow r\}$. For the second rule, $r.$, its least specialization by $r \leftarrow p \wedge q \wedge \neg r$ generates two rules, $r \leftarrow \neg p \wedge \neg r$ and $r \leftarrow \neg q \wedge \neg r$. But these rules are respectively subsumed by $r \leftarrow \neg p$ and $r \leftarrow \neg q$ that are already in P. The subsumed rules are not added to P, so that the analysis of (pq, p) results in the specialization of $q.$ and the deletion of $r \leftarrow \neg r$.

Learning continues with similar cases until the last transition (q, pr) where we have a special case. From this transition, **LF1T** infers the rule $q \leftarrow \neg p \wedge q \wedge \neg r$ that is subsumed by P on $R := q \leftarrow \neg p \wedge q \wedge \neg r$. Because $|b(R)| = |\mathcal{B}|$ it cannot be specialized so that P becomes consistent with (q, pr), **LF1T** just removes R from P.

Theorem 4 (Completeness of LF1T with least specialization). *Let $P_0^{\mathcal{B}}$ be the most general complete prime NLP of a given Herbrand base B. Initializing* **LF1T** *with $P_0^{\mathcal{B}}$, by using least specialization iteratively on a set of state transitions E,* **LF1T** *learns an NLP that realizes E.*

Theorem 5 (LF1T with least specialization outputs a complete prime NLP). *Let $P_0^{\mathcal{B}}$ be the most general complete prime NLP of a given Herbrand base B. Initializing* **LF1T** *with $P_0^{\mathcal{B}}$, by using least specialization iteratively on a set of state transitions E,* **LF1T** *learns the complete prime NLP of E.*

Theorem 6 (Complexity). *Let n be the size of the Herbrand base $|\mathcal{B}|$. Using least specialization, the memory complexity of* **LF1T** *remains in the same order as the previous algorithms based on ground resolution, i.e., $O(2^n)$. But the computational complexity of* **LF1T** *with least specialization is higher than the previous algorithms based on ground resolution, i.e $O(n \cdot 4^n)$ and $O(4^n)$, respectively. Same complexity results for full naïve (resp. ground) resolution.*

6 Evaluation

In this section, we evaluate our new learning methods through experiments. We apply our new **LF1T** algorithms to learn Boolean networks. Here we run our learning program on the same benchmarks used in [8] and [16]. These benchmarks are Boolean networks taken from Dubrova and Teslenko [3], which include those networks about control of flower morphogenesis in *Arabidopsis thaliana*, budding yeast cell cycle regulation, fission yeast cell cycle regulation, mammalian cell cycle regulation and T helper cell cycle regulation. Like in [8, 16], we first construct an NLP $\tau(N)$ from the Boolean function of a Boolean network N where each Boolean function is transformed into a DNF formula. Then, we get all possible 1-step state transitions of N from all $2^{|\mathcal{B}|}$ possible initial states I^0's by computing all stable models of $\tau(N) \cup I^0$ using the answer set solver `clasp` [5]. Finally, we use this set of state transitions to learn an NLP using our **LF1T** algorithm. Because a run of **LF1T** returns an NLP which can contain redundant rules, the original NLP P_{org} and the output NLP P_{LFIT} of **LF1T** can be different, but remain equivalent with respect to state transition, that is, $T_{P_{org}}$ and $T_{P_{LFIT}}$ are identical functions. Regarding the new algorithms, it can also be the case if the original NLP is not a complete prime NLP. For the new versions of **LF1T**, if P_{org} is not a prime complete NLP we will learn a simplification of P_{org}. Table 2 shows the memory space and time of a single **LF1T** run in learning a Boolean network for each benchmark on a processor Intel Core I7 (3610QM, 2.3GHz) with 4GB of RAM. It compares memory and run time of the three previous algorithm (naïve, ground and the BDD optimization of the ground version) with their extension to learn complete prime NLP and the new algorithm based on least specialization. For each version of **LF1T** the variable ordering is alphabetical and transition ordering is the one that clasp outputs. The time limit is set to two hours for each experiment. Memory is represented in (maximal) number of literal in the NLP learned. Except for LF1T-BDD, all implemented algorithms uses the same data structures. That is why even **LF1T** with least specialization cannot compete with the ground-BDD version regarding memory and run time. It is more relevant to compare it to the original implementation of **LF1T** with ground resolution and the new one with full ground resolution.

On Table 2 we can observe that, as the number of variable increases, the memory efficiency of least specialization regarding ground version becomes more interesting. Regarding run time, both algorithms have globally equivalent performances. But least specialization ensure that the output is unique in the fact that it is the complete prime NLP of the given input transitions. **LF1T** with

Algorithm	Mammalian (10)	Fission (10)	Budding (12)	Arabidopsis (16)	T helper (23)
Naïve	142 118/4.62s	126 237/3.65s	1 147 124/523s	T.O.	T.O.
Ground	1036/**0.04s**	1218/**0.05s**	21 470/0.26s	271 288/4.25s	T.O.
Ground-BDD	**180**/0.24s	**147**/0.24s	**541/0.19s**	**779/2.8s**	**611/3360s**
Full Naïve	377 539/29.25s	345587/24.03s	T.O.	T.O.	T.O.
Full Ground	1066/0.24s	1178/0.23s	23 738/4.04s	399 469/111s	T.O.
Least Specialization	**375/0.06s**	**377/0.08s**	**641/0.35s**	**2270/5.28s**	**3134/5263s**

Table 2. Memory use and learning time of **LF1T** for Boolean networks benchmarks up to 23 nodes in the same condition as in [8]

full ground resolution also ensure this property, but is much less efficient than least specialization regarding both memory use and run time. On the benchmark, least specialization is respectively 75%, 65%, 91% and 95% faster. Least specialization version also succeed to learn the t-helper benchmark (23 variables) in 1 hour and 21 minutes. The main interest of using least specialization is that it guarantees to obtain a unique NLP that contains all minimal conditions to make a variable true. Previous versions of **LF1T** do not have this property and experimental results showed that their output is sensitive to variable ordering and especially transition ordering. For a given set of state transitions E, the output of **LF1T** with least specialization is always the same whatever the variable ordering or transition ordering. It is easy to see that variable ordering has no impact on both learning time and memory use of the new versions of **LF1T** since they consider all generalizations/specializations. But transition ordering has a significant impact on the learning time of the new version of **LF1T** compared to previous ones. On all experiments we run, the ordering of the output of clingo gives the best results. More investigation are required to determine if we can design a heuristic to make a good ordering of the input transition to speed up the run time of the new algorithms.

7 Conclusion and Future Work

We proposed a new algorithm for learning from interpretation transitions based on least specialization. Given any state transition diagram we can learn an NLP that exactly captures the system dynamics. Learning is performed only from positive examples, and produces NLPs that consist only of rules to make literals true. Consistency of state transition rules is achieved by least specialization, in which minimality of rules is guaranteed. As a result, given any state transition diagram E, **LF1T** with least specialization always learns a unique NLP that contains all prime rules that realize E. It implies that the output of **LF1T** is no more sensitive to variable ordering or transition ordering. But, experimental results showed that the new algorithm is sensitive to input transitions ordering regarding run time. Design of an heuristic to make a good ordering of the input is one possible future work. We are now considering to extend our framework to learn non-deterministic dynamic systems. One of our expectation is to be able to learn probabilistic logic program from interpretation of transitions. Assuming that probability of transition are given as input it should be possible to infer probabilistic rules using adapted LFIT techniques. But how to combine probabilities when generalization/specialization occurs is an interesting problem that we plan to tackle in our future works.

References

1. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. Foundations of deductive databases and logic programming p. 89 (1988)
2. Avila Garcez, A., Zaverucha, G.: The connectionist inductive learning and logic programming system. Applied Intelligence 11(1), 59–77 (1999)
3. Dubrova, E., Teslenko, M.: A sat-based algorithm for finding attractors in synchronous boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 8(5), 1393–1399 (2011)
4. d'Avila Garcez, A., Broda, K., Gabbay, D.: Symbolic knowledge extraction from trained neural networks: A sound approach. Artificial Intelligence 125(12), 155 – 207 (2001), http://www.sciencedirect.com/science/article/pii/S0004370200000771
5. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers (2012)
6. Inoue, K.: Logic programming for boolean networks. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two. pp. 924–930. AAAI Press (2011)
7. Inoue, K.: Dnf hypotheses in explanatory induction. In: Inductive Logic Programming, pp. 173–188. Springer (2012)
8. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning pp. 1–29 (2012)
9. Inoue, K., Sakama, C.: Oscillating behavior of logic programs. In: Correct Reasoning, pp. 345–362. Springer (2012)
10. Kean, A., Tsiknis, G.: An incremental method for generating prime implicants/implicates. Journal of Symbolic Computation 9(2), 185–206 (1990)
11. Michalski, R.S.: A theory and methodology of inductive learning. Artificial intelligence 20(2), 111–161 (1983)
12. Mitchell, T.M.: Generalization as search. Artificial intelligence 18(2), 203–226 (1982)
13. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming 19, 629–679 (1994)
14. Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., Srinivasan, A.: Ilp turns 20. Machine learning 86(1), 3–23 (2012)
15. Plotkin, G.D.: A note on inductive generalization. Machine intelligence 5(1), 153–163 (1970)
16. Ribeiro, T., Inoue, K., Sakama, C.: A bdd-based algorithm for learning from interpretation transition. to appear in LNAI (2013), presented at the 23rd International Conference on Inductive Logic Programming (ILP 2013)
17. Tison, P.: Generalization of consensus theory and application to the minimization of boolean functions. Electronic Computers, IEEE Transactions on (4), 446–456 (1967)
18. Van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM (JACM) 23(4), 733–742 (1976)

A Appendix

A.1 Proof of Theorem 1 (completeness)

Given a set E of pairs of interpretations, **LF1T** with full naïve (resp. ground) resolution is complete for E.

Proof. According to Theorem 1 (resp. 2) of [8], **LF1T** with naïve (resp. ground) resolution is complete for E. It is trivial that any rules produced by naïve (resp. ground) resolution can be obtained by full naïve (resp. ground) resolution. Then, if P and P' are respectively obtained by naïve (resp. ground) resolution and full naïve (resp. ground) resolution, P' theory-subsumes P. If a program P is complete for E , a program P' that theory-subsumes P is also complete for E. Since P is complete for E by Theorem 1 of [8], P' is complete for E. □

A.2 Proof of Theorem 1: (soundness)

Given a set E of pairs of interpretations, **LF1T** with full naïve (resp. ground) resolution is sound for E.

Proof. All rules that can be produced by naïve (resp. ground) resolution can be obtained by full naïve (resp. ground) resolution. Since all rules produced by naïve (resp. ground) resolution are sound for E (Corrollary 1 (resp. 2) of [8]), full naïve (resp. ground) resolution is sound for E. □

A.3 Proof of Theorem 2

Given a set E of pairs of interpretations, **LF1T** with full naïve (resp. ground) resolution learn the complete prime NLP that realize E.

Proof. Let us assume that **LF1T** with full naïve resolution does not learn a prime NLP of E. If our assumption is correct it implies that there exists R a prime rule for E that cannot be learned by **LF1T** with full naïve resolution. Let $\mathcal{B}$ be the herbrand base of E.
Case 1: $|b(R)| = |\mathcal{B}|$, R will be directly infer from a transition $(I, J) \in E$. This is a contradiction with our assumption.
Case 2: $|b(R)| < |\mathcal{B}|$. let l be a literal such that $l \notin b(R)$, according to our assumption, their is a rule R' that is one of the rule $R_1 := h(R) \leftarrow b(R) \cup l$ or $R_2 := h(R) \leftarrow b(R) \cup \bar{l}$ and R' cannot be learned because $res(R_1, R_2) = R$. Recursively, what applies to R applies to R' until we reach a rule R'' such that $|b(R'')| = |\mathcal{B}|$. Our assumption implies that this rule R'' cannot be learned, but R'' will be directly infer from a transition $(I, J) \in E$, this is a contradiction. Since ground resolution can learn all rules learn by naïve resolution, the proof also applies to **LF1T** with full ground resolution. □

A.4 Proof of Theorem 3

Let R_1, R_2 be two rules such that $b(R_1) \subseteq b(R_2)$. Let S_1 be the set of rules subsumed by R_1 and S_2 be the rules of S_1 that subsume R_2. The least specialization of R_1 by R_2 only subsumes the set of rules $S_1 \setminus S_2$.

Proof. :
According to Definition 9, the least specialization of R_1 by R_2 is as follows:

$$ls(R_1, R_2) = \{h(R_1) \leftarrow (b(R_1) \wedge \neg b(R_2))\}$$

All rule R of S_2 subsumes R_2, then according to Definition 1 $b(R) \subseteq b(R_2)$. If $ls(R_1, R_2)$ subsumes an R then there exists $R' \in ls(R_1, R_2)$ and $b(R') \subseteq b(R)$. Since $R' \in ls(R_1, R_2)$, there is a $l \in b(R_2)$ such that $\bar{l} \in b(R')$, so that $b(R') \not\subseteq b(R_2)$. Since all $R \in S_2$ subsume R_2, R' cannot subsume any R since R' does not subsume R_2.
Conclusion 1: the least specialization of R_1 by R_2 cannot subsume any $R \in S_2$.

Let us suppose there is a rule $R' \in S_1$ that does not subsumes R_2 and is not subsumed by $ls(R_1, R_2)$. Let l_i be the i^{th} literal of $b(R_2)$, then:

$$ls(R_1, R_2) = \{(h(R_1) \leftarrow (b(R_1) \wedge \overline{l_i}) | l_i \in b(R_2) \setminus b(R_1)\} (1)$$

R' is subsumed by R_1, so that $R' = h(R_1) \leftarrow b(R_1) \cup S$, with S a set of literal. R' does not subsume R_2, so that there exists a $l \in b(R_2) \setminus b(R_1)$ such that $\bar{l} \in S$. According to (1), the rule $R'' = h(R_1) \leftarrow b(R_1) \wedge \bar{l}$ is in $ls(R_1, R_2)$. Since R'' subsumes R' and $R'' \in ls(R_1, R_2)$, $ls(R_1, R_2)$ subsumes R'.
Conclusion 2: the least specialization of R_1 by R_2 subsumes all rule of S_1 that does not subsume R_2.
Final conclusion: the least specialization of R_1 by R_2 only subsumes $S_1 \setminus S_2$. □

Now, let P be an NLP and R be a rule such that P subsumes R. Let S_P be the set of rules subsumed by P and S_R be the rules of S_P that subsume R. The least specialization of P by R only subsumes the set of rules $S_P \setminus S_R$.

Proof. :
According to Definition 5, the least specialization $ls(P, R)$ of P by R is as follows:

$$ls(P, R) = (P \setminus S_P) \cup (\bigcup_{R_P \in S_P} ls(R_P, R))$$

For any rule R_P let S_{R_P} be the set of rules subsumed by R_P and $S_{R_P 2} \in S_R$ be the rule of S_{R_P} that subsume R.

According to Theorem 3 the least specialization of R_P by R only subsumes $S_{R_P} \setminus S_{R_P 2}$. So that $\bigcup_{R_P \in S_P} ls(R_P, R)$ only subsumes $(\bigcup_{R_P \in S_P} S_{R_P} \setminus S_{R_P 2}) = (\bigcup_{R_P \in S_P} S_{R_P}) \setminus S_R$. Then $ls(P, R)$

only subsumes the rules subsumed by $(P \setminus S_P) \cup (\bigcup_{R_P \in S_P} S_{R_P}) \setminus S_R$, that is $S_P \setminus S_R$.

Conclusion: The least specialization of P by R only subsumes $S_P \setminus S_R$. □

A.5 Proof of Theorem 4

Let $P_0^{\mathcal{B}}$ be the most general complete prime NLP of a given Herbrand base B, i.e. the NLP that contains only facts

$$P_0^{\mathcal{B}} = \{p.|p \in \mathcal{B}\}$$

Initializing **LF1T** with $P_0^{\mathcal{B}}$, by using least specialization iteratively on the transitions of a set of state transitions E, **LF1T** learns an NLP P that realizes E.

Proof. :
Let P be an NLP consistent with a set of transitions E', S_P be the set of rules subsumed by P and a state transition (I, J) such that $E' \subset E$ and $(I, J) \in E$ but $(I, J) \notin E'$. According to Theorem 3, for any rule R_A^I that can be inferred by **LF1T** from (I, J) that is subsumed by P, the least specialization $ls(P, R_A^I)$ of P by R_A^I exactly subsumes the rules subsumed by P except the ones subsumed by R_A^I. Since $|R_A^I|$ is $|\mathcal{B}|$, R_A^I only subsumes itself so that $ls(P, R)$ exactly subsumes $S_P \setminus R_A^I$. Let P' be the NLP obtained by least specialization of P with all R_A^I that can be inferred from (I, J), then P' is consistent with $E' \cup \{(I, J)\}$.
Conclusion 1: LF1T keep the consistency of the NLP learned.

LF1T start with $P_0^{\mathcal{B}}$ as initial NLP. $P_0^{\mathcal{B}}$ is at least consistent with $\emptyset \subseteq E$. According to conclusion 1, initializing **LF1T** with $P_0^{\mathcal{B}}$ and by using least specialization iteratively on the element of E when its needed, **LF1T** learns an NLP that realizes E. □

A.6 Proof of Theorem 5

Let $P_0^{\mathcal{B}}$ be the most general complete prime NLP of a given Herbrand base B, i.e. the NLP that contains only facts

$$P_0^{\mathcal{B}} = \{p.|p \in \mathcal{B}\}$$

Initializing **LF1T** with $P_0^{\mathcal{B}}$, by using least specialization iteratively on a set of state transitions E, **LF1T** learns the complete prime NLP of E.

Proof. :
Let us assume that **LF1T** with least specialization does not learn a prime NLP of E. If our assumption is correct, according to Theorem 4, **LF1T** learns a NLP P, that is consistent with E and P is not the complete prime NLP of E. **LF1T** start with $P_0^{\mathcal{B}}$ as initial NLP, $P_0^{\mathcal{B}}$ is the most general complete prime NLP that can cover E.
Consequence 1: LF1T with least specialization can transform a complete prime NLP into an NLP that is not a complete prime NLP.

Let P be the complete prime NLP of a set of state transition $E' \subset E$ and $(I, J) \notin E'$, such that P is not consistent with (I, J). Our assumption implies that the least specialization P' of P by the rules inferred from (I, J) is not the complete prime NLP of $E' \cup (I, J)$. According to Definition 7, there is two possibilities:

- case 1: $\exists R \in P'$ such that R is not a prime rule of $E' \cup (I, J)$.
- case 2: $\exists R' \notin P'$ such that R' is a prime rule of $E' \cup (I, J)$.

Case 1.1: If $R \in P$, it implies that R is a prime rule of E' and that R is consistent with (I, J), otherwise R should have been specialized. Because R is not a prime rule of $E' \cup (I, J)$ it implies that there exists a rule R_m consistent with $E' \cup (I, J)$ that is more general than R, i.e. $b(R_m) \subset b(R)$. Then R_m is also consistent with E', but since R is a prime rule of E' there does not exist any rule consistent with E' that is more general than R. This is a contradiction.

Case 1.2: Now let us suppose that $R \notin P$; then R has been obtained by least specialization of a rule $R_P \in P$ by a rule inferred from (I, J). It implies that $\exists l \in b(R)$ and $\bar{l} \in I$. If R is not a prime rule of $E' \cup (I, J)$, there exists R_m a prime rule of $E' \cup (I, J)$ and R_m is more general than R. It implies that $l \in R_m$ otherwise R_m is not consistents with (I, J) because it will also subsumes R_P that is not consistents with (I, J). Since R_m is consistent with $E' \cup (I, J)$ it is also consistent with E'. This implies that $\exists R'_m$ a prime rule of E' that subsumes R_m (it can be R_m itself), R'_m also subsumes R. Since P is the complete prime NLP of E', $R'_m \in P$.

Case 1.2.1: Let suppose that $l \notin b(R'_m)$, since $l \in b(R)$ and R'_m subsumes R then R'_m subsumes R_P because $R = h(R_P) \leftarrow b(R_P) \cup l$. But since R_P is a prime rule of E' it implies that $R'_m = R_P$. In that case it means that R_P subsumes R_m and since $l \in R_m$, $h(R_P) \leftarrow b(R_P) \cup l$ also subsumes R_m. Since $h(R_P) \leftarrow b(R_P) \cup l$ is R, R subsumes R_m and R_m can neither be more general than R nor a prime rule of $E' \cup (I, J)$. This is a contradiction with case 2.

Case 1.2.2: Finally let us suppose that $l \in b(R'_m)$, since R'_m is consistent with E and $l \in I$, R'_m is consistent with $E' \cup (I, J)$. But R'_m subsumes R_m and since R_m is a prime rule of $E' \cup (I, J)$ it implies that $R'_m = R_m$. In that case $R_m \in P$ and because R_m is consistent with (I, J) and R_m subsumes R, **LF1T** will not add R into P'. This is a contradiction with case 1.

Case 2: Let consider that there exists a $R' \notin P'$ such that R' is a prime rule of $E' \cup (I, J)$. Since $R' \notin P'$, $R' \notin P$ and R' is not a prime rule of E' since P is the complete prime NLP of E'. Then, there exists $R_m \in P$ a prime rule of E' such that R_m subsumes R' and $R_m \notin P'$ since R' is a prime rule of $E' \cup (I, J)$. Then, $b(R') = b(R'_m) \cup S$ with S a non-empty set of literals such that for all $l \in S$, $l \notin b(R_m)$. Since $R_m \notin P'$, there is a rule $R^I_{h(R_m)}$ that can be inferred from (I, J) and subsumed by R_m. And there is no rule $R'_m \in ls(R_m, R^I_{h(R_m)})$ that subsumes R' since R' is a prime rule of $E' \cup (I, J)$. Then, for all $l' \in b(R^I_{h(R_m)})$, $\overline{l'} \notin b(R')$ otherwise there is a R'_m that subsumes R'. Since $|b(R^I_{h(R_m)})| = \mathcal{B}$, $b(R')$ cannot contains a literal that is not in $b(R^I_{h(R_m)})$ so that R' subsumes $R^I_{h(R_m)}$. R' cannot be a prime rule of $E' \cup (I, J)$ since R' is not consistent with (I, J), this is a contradiction.

Conclusion: If P is a complete prime NLP of $E' \subset E$, for any $(I, J) \in E$ **LF1T** with least specialization will learn the complete prime NLP P' of $E' \cup (I, J)$. Since **LF1T** starts with a complete prime NLP that is $P^{\mathcal{B}}_0$, according to Theorem 4, **LF1T** will learn a NLP consistent with E, our last statement implies that this NLP is the complete prime NLP of E since **LF1T** cannot specify a complete prime NLP into an NLP that is not a complete prime NLP. □

A.7 Proof of Theorem 6

Let n be the size of the Herbrand base $|\mathcal{B}|$. Using least specialization, the memory complexity of **LF1T** remains in the same order as the previous algorithms based on ground resolution, i.e., $O(2^n)$. But the computational complexity of **LF1T** with least specialization is higher than the previous algorithms based on ground resolution, i.e $O(n \cdot 4^n)$ and $O(4^n)$, respectively. Same complexity results for full naïve (resp. ground) resolution.

Proof. Let n be the size of the Herbrand base $|\mathcal{B}|$ of a set of state transitions E. This n is also the number of possible heads of rules. Furthermore, n is also the maximum size of a rule, i.e. the number of literals in the body; a literal can appear at most one time in the body of a rule. For each head there are 3^n possible bodies: each literal can either be positive, negative or absent from the body. From these preliminaries we conclude that the size of a NLP $|P|$ learned by **LF1T** from E is at most $n \cdot 3^n$. But since a NLP P learned by **LF1T** only contains prime rules of E, $|P|$ cannot exceed $n \cdot 2^n$; in the worst case, P contains only rules of size n where all literals appear and there is only $n \cdot 2^n$ such rules. If P contains a rule with m literals $(m < n)$, this rule subsumes 2^{n-m} rules which cannot appear in P. Finally, least specialization also ensures that P does not contain any pair of complementary rules, so that the complexity is further divided by n; that is, $|P|$ is bounded by $O(\frac{n \cdot 2^n}{n}) = O(2^n)$.

When **LF1T** infers a rule R^I_A from a transition $(I, J) \in E$ where $A \notin J$, it has to compare it with all rules in P to extract the ones that need to be specialized. This operation has a complexity of $O(|P|) = O(2^n)$. Since $|b(R^I_A)| = n$, according to Definition 5 the least specialization of a rule $R \in P$ can at most generate n different rules. In the worst case all rules of P with $\overline{h(R^I_A)}$ as head subsume R^I_A. There are possibly $2^n/n$ such rules in P, so that **LF1T** generates at most 2^n rules for each R^I_A. For each $(I, J) \in E$, **LF1T** can infer at most n rules R^I_A. In the worst case, **LF1T** can generates $n \cdot 2^n$ rules that are compared with the 2^n rules of P. Thus, construction of an NLP which realizes E implies $n \cdot 2^n.2^n = n \cdot 4^n$ operations. The same proof applies to **LF1T** naïve (resp. ground) resolution, when **LF1T** infers a rule R^I_A from a transition $(I, J) \in E$ where $A \in J$. The complexity of learning an NLP from a complete set of state transitions with an Herbrand base of size n is $O(n \cdot 4^n)$. □