Learning rules from user behaviour

Domenico Corapi, Oliver Ray, Alessandra Russo, Arosha Bandara, and Emil Lupu

Abstract Pervasive computing requires infrastructures that adapt to changes in user behaviour while minimising user interactions. Policy-based approaches have been proposed as a means of providing adaptability but, at present, require policy goals and rules to be explicitly defined by users. This paper presents a novel, logic-based approach for automatically learning and updating models of users from their observed behaviour. We show how this task can be accomplished using a non-monotonic learning system, and we illustrate how the approach can be exploited within a pervasive computing framework.

1 Introduction

Pervasive computing is enabled by the development of increasingly complex devices and software infrastructures that accompany users in everyday life. Such systems must autonomously adapt to changes in user context and behaviour, whilst operating seamlessly with minimal user intervention. They must, therefore, be able to learn from sensory input and user actions. Yet user acceptance requires them to be predictable, capable of explaining their actions, and providing some way for users to understand and amend what has been learnt. This directs us towards techniques that use logical rules for knowledge representation and reasoning. Even though some statistical pre-processing of raw sensor data will inevitably be required, there are considerable advantages in adopting a core logical formalism, such as simple and

Domenico Corapi, Alessandra Russo, Emil Lupu
Imperial College London, Exhibition Road, London UK, email: {d.corapi, a.russo, e.c.lupu}@imperial.ac.uk
Oliver Ray
University of Bristol, Woodland Road, Bristol UK, email: oray@cs.bris.ac.uk
Arosha Bandara
The Open University, Walton Hall Campus, Milton Keynes UK, email: a.k.bandara@open.ac.uk

modular enforcement through policy frameworks [11, 18] and principled representations of space and time [23]. Logic programming is an ideal choice for knowledge representation from a computational point of view and it also benefits from Inductive Logic Programming (ILP) [17] tools that permit the learning of logic programs from examples.

Learning rules of user behaviour through inductive reasoning poses several challenges. Learning must be incremental: as examples of user behaviour are continuously added, the system must permit periodic revision of the rules and knowledge learnt. Moreover, the system must cater for temporal aspects, expressing both persistence and change through time, and exceptions to previously learnt rules. For this, the system must be capable of non-monotonic[1] reasoning [13]. The system must reason with partial information whilst providing fine grained control of the reasoning process to satisfy appropriate user-defined language and search biases (such as minimising the changes made to the initial theory).

This paper presents an algorithm for learning and revising models of user behaviour which makes use of a non-monotonic ILP system, called XHAIL (eXtended Hybrid Abductive Inductive Learning) [19], that is capable of learning normal logic programs from a given set of examples and background knowledge. The contribution of this paper is twofold. First a novel algorithm is presented that is able to perform general theory revision by supporting automatic computation of new theories $T^{'}$ that are not necessarily extensions of original theories T, to correctly account for newly acquired examples E. Second, an application of the algorithm to learning rules describing behaviour of mobile phone users is presented by means of a simplified example consisting of learning the circumstances in which users accept, reject or ignore calls. Once learnt, these rules can be periodically reviewed and amended by the user and enacted automatically on the device avoiding user intervention. This work is part of a larger project [1] that seeks to exploit the proposed approach in the context of privacy policies.

The paper is structured as follows. Section 2 summarises relevant background material on ILP. Section 3 describes the main features of the approach by introducing basic concepts, presenting a *learning-based theory revision* algorithm and illustrating its application to the example. Section 4 relates our approach with other existing techniques for theory revision. Section 5 concludes the paper with a summary and some remarks about future work.

2 Background

Inductive Logic Programming (ILP) [17] is concerned with the computation of hypotheses H that generalise a set of (positive and negative) examples E with respect to a prior background knowledge B. In this paper, we consider the case when B and

[1] A logical system is non-monotonic if given a theory (e.g. $\{bird(tweety).\ bird(X) \leftarrow penguin(X).\ fly(X) \leftarrow bird(X), \neg penguin(X).\}$), adding new information ($penguin(tweety)$) may cause some conclusions to be retracted ($fly(tweety)$).

H are normal logic programs [10], E is a set of ground literals (with positive and negative ground literals representing positive and negative examples, respectively) and H satisfies the condition $B \cup H \models E$ under the credulous stable model semantics [8]. As formalised in Definition 1 below, it is usual to further restrict the clauses in H to a set of clauses S called the *hypothesis space*.

Definition 1. Given a normal logic program B, a set of ground literals E, and a set clauses S, the task of ILP is to find a normal logic program $H \subseteq S$, consistent with B such that $B \cup H \models E$. In this case, H is called an *inductive generalisation* of E wrt. B and S.

We use the XHAIL system that, in a three-phase approach [21], constructs and generalises a set of ground hypotheses K, called a *Kernel Set* of B and E. This can be regarded as a non-monotonic multi-clause generalisation of the Bottom Set concept [16] used in several well-known monotonic ILP systems. Like most ILP systems, XHAIL heavily exploits a language bias, specified by a set M of so called *mode declarations* [16], to bound the ILP hypothesis space when constructing and generalising a Kernel Set. A mode declaration $m \in M$ is either a *head declaration* of the form $modeh(s)$ or a *body declaration* of the form $modeb(s)$ where s is a ground literal, called *scheme*, containing *placemarker* terms of the form $+t$, $-t$ and $\#t$ which must be replaced by input variables, output variables, and constants of type t respectively. For example $modeb(in\_group(+contact, \#contact\_list\_group))$ allows rules in H to contain literals with the predicate in_group; the first argument of type $contact$ must be an input variable (i.e. an input variable in the head or an output variable in some preceding body literal), the second of type $contact\_list\_group$ must be a ground term. The three phases of the XHAIL approach are implemented using a non-monotonic answer set solver. In the first phase, the head declarations are used to abduce a set of ground atoms Δ such that $T \cup \Delta \models E$. Atoms in Δ are the head atoms of the Kernel Set. In the second phase, the body atoms of the Kernel Set are computed as successful instances of queries obtained from the body declarations in M. In the third phase, the hypothesis is computed by searching for a compressive theory H that subsumes the Kernel Set, is consistent with the background knowledge, covers the examples and falls within the hypothesis space.

To represent and reason about dynamic systems, we use the Event Calculus (EC) formalism [23]. EC normal logic programs include core domain-independent rules describing general principles for inferring when properties (i.e. *fluents*) are true (resp. not true) at particular time-points, denoted as $holdsAt(F,T)$ (resp. *not* $holdsAt(F,T)$), based on which events have previously occurred (denoted $happens(E,T)$). In addition, the program includes a collection of *domain-dependent* rules, describing the effects of events (using the predicates $initiates(E,F,T)$ and $terminates(E,F,T)$, as well as the time-points at which events occur (using the predicate $happens$).

3 Learning user behaviour

Pervasive computing requires infrastructures that adapt to changes in user behaviors while minimizing user intervention. For example the user's actions on mobile devices can provide precious information that applications can exploit in order to operate autonomously and/or to improve usability and user acceptance. This can be achieved by complementing applications with policy rules describing user-behaviour. We adopt a *declarative representation* of the knowledge about user behaviour and perform *continuous revision* of this knowledge through learning from instances of user actions. The system we propose *learns* and *revises* a user theory U assumed to be accessible by applications (e.g. for mobile devices) or by policy management systems. U is a normal logic program defining conditions under which user actions are performed. The application (or policy management system) can query U to (autonomously) determine a response to events or requests. For example, on mobile devices, U can be queried to determine if the user would allow access to his/her current location in response to a request:

$$? - do(allow(current\_location, request\_id), time).$$

The answer to this query can be based on various conditions, defined by a *background theory* B about the domain knowledge of the application. These can be, for instance, (a) properties of the request, such as the ID of the requester, the time of the request, proofs of identity, etc., (b) contextual information at the particular *time*, such as the location of the user, the profile active on the mobile phone or the number of nearby devices, (c) logged actions and events and (d) other application-specific knowledge. Learning new user-behavior rules means, in this example, elaborating from instances of allowing and disallowing access to location, general rules that classify when access to location can be allowed. The user may, however, subsequently override policy-based autonomous responses on its own device, thereby generating instances of user actions that are not longer covered by the user theory U. Revising existing user-behavior rules in U means identifying situations of over-generalisation or of lack of coverage in U. This process is *iterative*. Each iteration considers all the new examples which occurred since the last computation and optionally previously computed examples based on a time based sliding window. The computation within a single iteration is captured by Algorithm 1.

3.1 The revision algorithm

The algorithm consists of three phases: the *pre-processing* phase that "transforms" the rules of the user theory U, into "defeasible" rules with exceptions, the *learning* phase that computes exception rules (if any), and the *post-processing* phase that "re-factors" the defeasible rules into revised non-defeasible rules based on the exceptions rules learnt in the second phase. Informally, exception rules learned by XHAIL are *prescriptions* for changes in the current user theory U in order to cover new examples of user actions. These changes can be addition or deletion of entire rules, and/or addition or deletion of literals in the body of existing rules.

```
Input: $B$ background theory; $U$ user agent theory; $E$ set of current examples; $M$ mode
    declarations
Output: $U'$ revised theory according to current examples
/* Pre-processing phase                                                    */
$\tilde{U} = \varnothing$;
foreach rule $\alpha_i \leftarrow {\delta_i}^1, ..., {\delta_i}^n \in U$ do
    $S = \{{\delta_i}^1, ..., {\delta_i}^n\}$;
    $\alpha{*}_i$ denotes the schema in the modeh declaration referring to $\alpha_i$;
    $\tilde{U} = \tilde{U} \cup \{\alpha_i \leftarrow try(i,1,{\delta_i}^1), ..., try(i,n,{\delta_i}^n), \neg exception(i,\alpha_i)\ \}$;
    $M = M \cup \{modeh(exception(\#int, \alpha{*}_i))\}.$ ;
    foreach ${\delta_i}^j \in S$ do
        $\tilde{U} = \tilde{U} \cup \{try(i,j,{\delta_i}^j) \leftarrow use(i,j), {\delta_i}^j\} \cup \{try(i,j,{\delta_i}^j) \leftarrow \neg use(i,j)\}$ ;
    end
end
$\tilde{U} = \tilde{U} \cup \{use(I,J) \leftarrow \neg del(I,J)\}$ ;
$M = M \cup \{modeh(del(\#int, \#int))\}$ ;
/* Learning phase                                                          */
$H = XHAIL(B \cup \tilde{U}, E, M)$ ;
/* Post-processing phase                                                   */
$U' = theory\_refactoring(U,H)$;
return $U'$;
```

Algorithm 1: Pseudo code of the algorithm.

The algorithm shows how XHAIL, which would normally be used to learn rules from scratch, can be used to discover a minimal set of revisions to an initial set of rules as well as new rules. The inputs are a set of mode declaration M, a background knowledge B, a user theory U and a set of examples E. The former defines the the atoms that are allowed to be head of new rules and part of the body of the rules. For instance, body literals can be defined as to not contain conditions about GPS location but to refer to higher-level location information (e.g. work, home) thus defining a more appropriate hypothesis space. The background knowledge B, expressed in EC, defines both static and dynamic domain-specific properties of the device and its environment, in addition to the EC domain-independent axioms. The body of these rules includes conditions expressed in terms of *happens* and *holdsAt*. The set E of current examples is a set of *do* ground literals. The output is a revised user theory U' that, together with B, covers the current examples E.

Pre-processing phase: During this phase the given user theory U is rewritten in a normal logic program, $\tilde{U}$, suitable for learning exceptions. This consists of the following two syntactic transformations. First, for every rule in U, every body literal δ^i_j is replaced by the atom $try(i, j, \delta^i_j)$, where i is the index of the rule, j is the index of the body literal in the rule and the third argument is a reified term for the literal δ^i_j. Furthermore, the literal $\neg exception(i, \alpha_i)$ is added to the body of the rule where i is the index of the rule and α is the reified term for the head of the rule.

Intuitively, this transformation lifts the standard ILP process of learning hypotheses about examples up to the (meta-)process of *learning hypothesis about the rules and their exception cases*. Second, for each $try(i, j, \delta_j^i)$ introduced in the program, the rules $try(i, j, \delta_j^i) \leftarrow use(i, j), \delta_j^i$ and $try(i, j, \delta_i{}^j) \leftarrow \neg use(i, j)$ are added to the program together with the definition $use(I,J) \leftarrow \neg del(I,J)$ of the predicate use. Head mode declaration for $exception$ and del are added to M. This sets the learning task to compute exceptions cases for rules in the current user theory U and instances of body literals that need to be deleted.

Learning phase: This phase uses the XHAIL system. In the first phase of XHAIL, a set Δ of ground atoms of the form $exception(i, \alpha_i)$ and $del(i, j)$ are computed so that $B \cup U \cup \Delta \models E$. This set indicates the definitions (i) of the predicates α_i that need exceptions, and the literals (with index j) in the rules i that need to be deleted for the given set of examples E to be entailed by the user theory U. In the second phase, XHAIL computes the body of the exception rules as instances of body declaration predicates (e.g. $holdsAt$) that are derivable from U. In the third phase, the search for maximally compressed (minimum number of literals in the hypothesis, [15]) hypothesis H such that E is true in a *partial stable model* ([9], [22]) of $B \cup \tilde{U} \cup H$, guarantees minimal induced revisions of U.

Post-processing phase: The *post-processing phase* generates a revised theory U' semantically equivalent to $\tilde{U} \cup H$ (and thus consistent with E). The algorithm is computationally simple. Informally, for each $del(i, j)$ fact in H the corresponding condition j in rule i in U is deleted. For each exception rule in H of the form $exception(i, \alpha_i) \leftarrow c_1, ..., c_n$, the corresponding rule i in U is substituted with n new rules, one for each condition c_h, $1 \leq h \leq n$. Each of these rules (h) will have in the head the predicate α_i and in the body all conditions present in the original rule i in U plus the additional condition $\neg c(h)$. An exception with empty body results in the original rule i to be deleted.

3.2 Example

This section illustrates Algorithm 1 with a simple case study where we aim to learn rules that define *the context in which a user accepts incoming calls* on a mobile phone. This is part of a larger test case in which user actions and contextual data are derived from real data on mobile phone usage collected in the Cityware project [3]. We used data collected over three days, running a revision step at the end of each day. Due to space limitations only the outcome of the third day is shown here.
A user theory U is revised up to the end of the second day:

$$\begin{aligned} U = \{ \quad & do(accept\_call(CallId, From), T) \leftarrow T \geq 07{:}30 \wedge in\_group(From, college). \\ & do(accept\_call(CallId, From), T) \leftarrow T \geq 07{:}30 \wedge \neg holdsAt(status(location(imperial)), T). \quad \} \end{aligned} \tag{1}$$

The set of examples collected in the third day is diagrammatically presented in Figure 1.

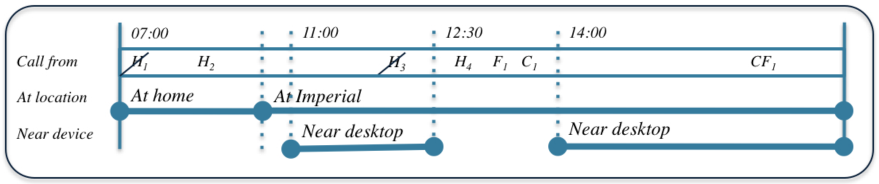

Fig. 1 Example scenario (C, H and F denote incoming calls from the user's college, home, and friends contact lists respectively. Refused calls are marked with a " ⁄ ")

Figure 2 displays a subset of the background knowledge B used in the scenario.

```
at_location(imperial, W, N) :- W > 95, W < 100, N > 75, N < 82.
...
happens(gps(57, 10), 07:30).
happens(gps(99, 78), 10:30).
...
happens(bluetooth_scan(desktop),  11:00).
happens(bluetooth_scan(daniel_laptop), 11:00).
...
in_group(bob, home).
in_group(alice, friends).
in_group(charles, college).
...
%Event Calculus
holdsAt(F,T+1)  :- holdsAt(F,T), not stopped(F,T).
holdsAt(F,T+1)  :- not holdsAt(F,T), started(F,T).
stopped(F,T)  :- happens(E,T), terminates(E,F,T).
started(F,T)  :- happens(E,T), initiates(E,F,T).
initiates(gps(W,N),status(location(P)) ,T) :- at_location(P, W, N).
initiates(bluetooth_scan(Dev),status(bluetooth_near(Dev)), T).
...
```

Fig. 2 A simplified sample of the background knowledge B used in the scenario.

U must be revised, since two calls from contacts not included in the *College* contact list are answered while at Imperial[2]), but no rule in U covers this case. The preprocessing phase transforms the user theory U into the following theory $\tilde{U}$:

[2] $do(accept\_call(bob), 13{:}00)$ and $do(accept\_call(alice), 13{:}05)$, respectively H_4 and F_1 in Figure 1. The contact lists of bob and $alice$ are defined in B by the following facts: $in\_group(bob, home)$ and $in\_group(alice, friends)$.

$$\begin{array}{ll}
\tilde{U} = \{ & do(accept\_call(CallId,From),T) \leftarrow try(1,1,T \geq 07{:}30) \wedge try(1,2,in\_group(From,college)) \wedge \\
& \quad \neg exception(2,do(accept\_call(CallId,From),T)). \\
& do(accept\_call(CallId,From),T) \leftarrow try(2,1,T \geq 07{:}30) \wedge try(2,2,\neg holdsAt(status(location(imperial)),T)) \wedge \\
& \quad \neg exception(2,do(accept\_call(CallId,From),T)). \\
& try(1,1,T \geq 07{:}30)) \leftarrow use(1,1) \wedge T \geq 07{:}30. \\
& try(1,1,T \geq 07{:}30)) \leftarrow \neg use(1,1). \\
& try(1,2,in\_group(From,college)) \leftarrow use(1,2) \wedge in\_group(From,college). \\
& try(1,2,in\_group(From,college)) \leftarrow \neg use(1,2). \\
& try(2,1,T \geq 07{:}30)) \leftarrow use(2,1) \wedge T \geq 07{:}30. \\
& try(2,1,T \geq 07{:}30)) \leftarrow \neg use(2,1). \\
& try(2,2,\neg holdsAt(status(location(imperial)),T)) \leftarrow use(1,1) \wedge T \geq \neg holdsAt(status(location(imperial)),T). \\
& try(2,2,\neg holdsAt(status(location(imperial)),T)) \leftarrow \neg use(1,1). \\
& use(I,J) \leftarrow \neg del(I,J). \quad \}
\end{array}$$

Given $\tilde{U}$, it is only possible to prove the examples set E by abducing the predicates $exception$ and del. The former is to be abduced to explain calls rejected or ignored by the user and the latter is to be abduced to explain calls the user accepts (currently not covered by U). Since XHAIL computes a minimal number of $exception$ and del clauses, U will be minimally revised. Thus the learning phase at the end of the third day gives the hypothesis:

$$\begin{array}{ll}
H = \{ & exception(2,do(accept\_call(CallId,From),T)) \leftarrow \\
& \quad holdsAt(status(bluetooth\_near(desktop\_computer)),T). \\
& del(2,2). \quad \}
\end{array}$$

The post-processing phase will then give the following revised user theory

$$\begin{array}{ll}
U' = \{ & do(accept\_call(CallId,From),T) \leftarrow \\
& \quad T \geq 07{:}30 \wedge in\_group(From,college). \\
& do(accept\_call(CallId,From),T) \leftarrow \\
& \quad T \geq 07{:}30 \wedge \neg holdsAt(status(bluetooth\_near(desktop\_computer)),T). \quad \}
\end{array} \tag{2}$$

Note that the choice between learning a new rule or revising an existing one, when both solutions are acceptable, is driven by minimality. In this way, we can preserve much of the knowledge learnt from previous examples. In the above case study, each revision computed by XHAIL took a couple of seconds on a Pentium laptop PC.

4 Discussion and related works

Although statistical techniques [5] may be necessary to process, classify and aggregate raw sensor data upstream, the core logical methodology described in this paper is well suited to learning user rules. For the application point of view it enables the computation of declarative rules that can be automatically formulated into structured English description policies which the user can validate, query and amend [2]. At the same time they can be automatically encoded into executable policies to enable dynamic, autonomous adaptation to user behaviour and context change. From a theoretical point of view, the approach proposed here shows that theory revision can be supported by integration of non-monotonic abductive and inductive reasoning. This is made possible by the use of the recently developed non-monotonic ILP system, XHAIL, that has several advantages over other state-of-the-art ILP systems. Among these, Progol5 [16] and Alecto [14], which also employ abduction to learn non-observed predicates, do not have a well-defined semantics for non-monotonic programs and their handling of negation-as-failure is limited. Compared to other first-order theory revision systems, like *INTHELEX* [7], *Audrey II* [25] and *FORTE* [20], the use of XHAIL allows for a more expressive language, more efficient inconsistency detection and exploits existing background knowledge to accurately constrain the computation of possible revisions. None of the above three revision systems can succeed in expressing contextual conditions in dynamic system behavior rules and in constructing compact rules using negated conditions. For example revised rules like (1) and (2) would not be obtained by FORTE as it can only learn Horn clauses (i.e. no negated literals in the body).

5 Conclusions and future work

We have proposed an algorithm, based on non-monotonic learning, that supports incremental learning and revision of user behavior rules. Rules are learnt based on past examples, which consist of positive and negative conditions under which the user performs actions. This work is part of a more ambitious effort to learn privacy policies on mobile devices, where the learning task is complemented with statistical learning and classification of raw data, and policy enforcement [18]. Larger scale experiments planned in the project [1], will give us insights on the scalability of the proposed approach.

Further work includes the development of components for efficient access to sensory data, caching and acceleration of the learning process, handling of noisy data and windowing techniques for the examples, mindful of existing solution for concept drift (e.g. [6], [24]). The changing nature of the concepts modelled demands for techniques able to reduce the complexity of the rules after repeated revisions. We are currently investigating probabilistic extensions [4] to address concept drift and two other open issues: establishing a preference criteria to choose the best between a set of minimal solutions and balancing the exploitation of the learned rules with the exploration of revisions. Finally, we are mindful of the complexity of the implementation of such algorithms but we will use valuable lessons and experience acquired through work on policy enforcement and distributed abductive reasoning on mobile devices [12] to improve the scale-down and efficiency of our current implementation.

Acknowledgements This work is funded by the UK EPSRC (EP/F023294/1) and supported by IBM Research as part of their Open Collaborative Research (OCR) initiative and Research Councils UK.

References

1. Bandara, A., Nuseibeh, B., Price, B., Rogers, Y., Dulay, N., et al.: Privacy rights management for mobile applications. In: 4th Int. Symposium on Usable Privacy and Security. Pittsburgh (2008)
2. Brodie, C., Karat, C., Karat, J., Feng, J.: Usable security and privacy: a case study of developing privacy management tools. In: SOUPS '05: Proc. of the 2005 symp. on Usable privacy and security, pp. 35–43. ACM, New York, NY, USA (2005)
3. Cityware: Urban design and pervasive systems. http://www.cityware.org.uk/
4. De Raedt, L., Thomas, G., Getoor, L., Kersting, K., Muggleton, S. (eds.): Probabilistic, Logical and Relational Learning - A Further Synthesis, 15.04. - 20.04.2007. IBFI, Schloss Dagstuhl, Germany (2008)
5. Eagle, N., Pentland, A.: Reality mining: sensing complex social systems. Personal and Ubiquitous Computing **10**(4), 255–268 (2006)
6. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., Di Mauro, N.: Incremental learning and concept drift in inthelex. Intell. Data Anal. **8**(3), 213–237 (2004)
7. Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Multistrategy theory revision: Induction and Abduction in INTHELEX. Mach. Learn. **38**(1-2), 133–156 (2000)
8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: R. Kowalski, K. Bowen (eds.) Logic Programming, pp. 1070–1080. MIT Press (1988)
9. Kakas, A., Kowalski, R., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992)
10. Lloyd, J.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
11. Lupu, E., Dulay, N., Sloman, M., Sventek, J., Heeps, S., Strowes, S., Twidle, K., Keoh, S.L., Schaeffer-Filho, A.: Amuse: autonomic management of ubiquitous e-health systems. Concurr. Comput. : Pract. Exper. **20**(3), 277–295 (2008). DOI http://dx.doi.org/10.1002/cpe.v20:3
12. Ma, J., Russo, A., Broda, K., Clark, K.: DARE: a system for Distributed Abductive REasoning. J. Autonomous Agents and Multi-Agent Systems **16**, 271–297 (2008)
13. Minker, J.: An overview of nonmonotonic reasoning and logic programming. Tech. Rep. UMIACS-TR-91-112, CS-TR-2736, University of Maryland, College Park, Maryland 20742 (August 1991)
14. Moyle, S.: An investigation into theory completion techniques in inductive logic. Ph.D. thesis, University of Oxford (2003)
15. Muggleton, S.: Inverse entailment and Progol. New Generation Comput. J. **13**, 245–286
16. Muggleton, S.: Learning from positive data. In: 6th Int. Workshop on Inductive Logic Programming, pp. 358–376. Springer Verlag, London, U.K. (1996)
17. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. J. of Logic Programming **19/20**, 629–679 (1994)
18. Ponder2: The ponder2 policy environment. www.ponder2.net
19. Ray, O.: Nonmonotonic abductive inductive learning. In: Journal of Applied Logic. (Elsevier, in press) (2008)
20. Richards, B., Mooney, R.J.: Automated refinement of first-order horn-clause domain theories. Machine Learning **19**(2), 95–131 (1995)
21. Russo, A.: A hybrid abductive inductive proof procedure. Logic J. of the IGPL **12**, 371–397(27)
22. Sacca, D., Zaniolo, C.: Stable models and non-determinism in logic programs with negation
23. Shanahan, M.: The event calculus explained. In: Artificial Intelligence Today, pp. 409–430 (1999)

24. Widmer, G.: Learning in the presence of concept drift and hidden contexts. In: Machine Learning, pp. 69–101 (1996)
25. Wogulis, J., Pazzani, M.: A methodology for evaluating theory revision systems: Results with Audrey II. In: 13th IJCAI, pp. 1128–1134 (1993)