Incremental Learning and Concept Drift in INTHELEX

F. Esposito, S. Ferilli, N. Fanizzi
T.M.A. Basile, N. Di Mauro
Dipartimento di Informatica
Università di Bari
via E. Orabona, 4 - 70125 Bari - Italia
{esposito, ferilli, fanizzi, basile, nicodimauro}@di.uniba.it

Abstract

Real-world tasks often involve a continuous flow of new information that affects the learned theory, a situation that classical batch (one-step) learning systems are hardly suitable to handle. On the contrary, incremental (also called "on-line") techniques are able to deal with such a situation by exploiting refinement operators. In many cases deep knowledge about the world is not available: Either incomplete information is available at the time of initial theory generation, or the nature of the concepts evolves dynamically. The latter situation is the most difficult to handle since time evolution needs to be considered. This work presents a new approach to learning in presence of concept drift, and in particular a special version of the incremental system INTHELEX purposely designed to implement such a technique. Its behavior in this context has been checked and analyzed by running it on two different datasets.

1 Introduction

The recent, enormous growth in available computational power, and the corresponding reduction of hardware components expensiveness, allowed researchers in the Artificial Intelligence (AI) field to develop and tune methods, techniques and systems that are able to cope with increasingly difficult tasks and environments with ever-increasing performance and effectiveness.

At the same time, the spread and development of computer technology in almost every concern of human activities has generated a great demand for information systems that can deal with a number of tasks, whose knowledge-intensive nature and whose exploitation in changing and/or not fully understood environments pose new challenges to the computer science researchers and practitioners. A prototypical example can be obtained by considering how the Internet technology has caused big changes in the Computer Science disci-

pline. Such changes are partly related to the interest of organizations and firms, publishing their business and information on-line, in capturing their (potential) users interests and expectations, in order to be able to comply with their needs and fulfill their demands.

Mixing together these two development directions, the result is improved AI systems that are capable of facing a wide range of real-world tasks. In this landscape, a central role is played by Machine Learning (ML) techniques, that are fundamental when an automatic system must adapt itself to unknown settings and environments. In particular, real-world tasks often involve a continuous flow of new information that affects the learned theory, a situation that classical batch (one-step) learning systems are hardly suitable to handle, since changes in the learned information can be only carried out by withdrawing the available theory and starting from scratch a new learning session that takes into account both the old and the new observations available. On the contrary, incremental (also called "on-line") techniques are able to deal with such a situation by exploiting refinement operators that can adjust the learned theory, without completely rejecting it, by generalizing/specializing the incorrect definitions according to the new available information. Obviously, the incremental setting implicitly assumes that the information (observations) gained at any given moment is incomplete, and thus that any learned theory is (could be) potentially susceptible of changes. This, in turn, prevents the possibility of taking unchangeable decisions and definitively cutting away, regarding them as useless, portions of the search space that seem to be outside the correct (target) concept definition region at a given time.

Another characteristic of real-world tasks and situations, that is worth pointing out, is that they are typically more complex than controlled ones, such as those occurring in artificial datasets purposely designed to test learning systems performance. In cases in which such a complexity grows significantly, a greatly enhanced flexibility and expressive power of the representations is needed, in order for them to be correctly and significantly handled. Such an expressive power sometimes cannot be provided by classical propositional, attribute-value or numeric representations, but requires the use of relations in order to describe observations whose shape and boundary cannot be predicted and fixed a priori. Classical examples are the Mutagenesis domain [28], or the paper document classification/understanding one, the family one and so on. Hence, the need of first-order logic as a representation language, and, as a consequence, the increasing interest in techniques and systems that can deal with such a setting.

The problem is, handling first-order descriptions requires a great computational effort, hence, again, the need for computational models, techniques and/or operators that can explore the search space in a more effective and efficient way, by reducing the combinatorial explosion of the allowed paths and focusing on the portion of the space that is more likely to contain the target solution. One example of desirable property is *ideality* [30], according to which refinement operators must be locally finite (i.e., they have to give their outcome exploiting finite time and space resources), proper (i.e., they should not loop on a solution that is equivalent to the starting point of the search space) and

complete (i.e., they should find a solution whenever it exists). The definition of such kind of operators is a fundamental problem in a logic framework for the inductive synthesis of theories from facts. Indeed, when the aim is to develop incrementally a logic program, that should be *correct* with respect to its *intended model* at the end of the development process, the *ideality* of the refinement operators plays a key role if the efficiency and the effectiveness of the design process is a non-negligible requirement. Unfortunately, when full Horn clause logic is chosen as representation language and either θ-*subsumption* or *implication* is adopted as generalization model, there exist no ideal refinement operators [32, 31, 30]. Thus, weaker models have to be found with a fair tradeoff between mechanizability and expressive power.

A further desirable property is inherent and full incrementality, i.e. the ability to learn correctly even when only one observation at a time is provided, and no previous version of the theory is available.

If a number of difficulties are related to learning concepts for which observations become available incrementally, still more complex is learning concepts whose definition may change in time. Indeed, such a case encompasses incrementality, in that examples of the different versions of the concept during its evolution must be given in the different stages (and times) the changing concept goes through. Additional difficulties with respect to the case of simple incrementality include deciding which, and how many, of the past examples should still constrain the exploration of the search space at a given moment. Indeed, according to the amount, "direction" and spread of the change, a number of different solutions should be exploited. In this paper, we will assume that the system trainer can trace concept drifts and provide examples accordingly. Thus, our focus of attention will be on how to handle drifting rather than on how to recognize it in a continuous flow of examples.

In the following we will first present our framework, based on Object Identity. Then, Section 3 will describe the architecture of the incremental learning system INTHELEX, followed in Section 4 by a deeper insigth into its refinement mechanisms. Successively, Section 5 will introduce our proposal on how to deal with the phenomenon of concept drift in an online learning setting, providing the algorithm that was embedded in INTHELEX in order to implement such a technique. In order to assess the quality of the proposed solution, two datasets were purposely designed on which to apply the resulting system, whose performance is reported and commented. Lastly, Section 6 will conclude the paper.

2 The Object Identity Framework

The presented research was carried out in the First-Order logic framework (see [17] as a reference to all the notions concerning logic programming). Specifically, a modification of Datalog [3], the function-free fragment of pure Prolog, was used as a representation language. Such a modification is obtained by imposing a further bias, based on the notion of Object Identity (*OI* for short) [5, 25].

Definition 2.1 (Object Identity) *Within a clause, terms (even variables) denoted with different symbols must be distinct.*

Such an assumption leads to a new representation language, called $Datalog^{OI}$, that is an instance of constraint logic programming [13]. It is also the basis for the definition of both an equational theory for Datalog clauses and a quasi-ordering upon them. In Datalog, the adoption of the Object Identity assumption can be viewed as a method for building an equational theory onto the ordering as well as onto the inference rules of the calculus (resolution, factorization and paramodulation) [20]. Such equational theory is very simple, since it consists of just one rewrite rule, in addition to the set of the axioms of Clark's Equality Theory (CET) [17], and specifically:

$\mathbf{t \neq s \in} \mathit{body}(C)$ **for each clause** C **in** $\mathcal{L}$ **and**
for all pairs t, s of distinct terms that occur in C **(OI)**

where $\mathcal{L}$ denotes the language that consists of all the possible Datalog clauses built from a finite number of predicates.

The (OI) rewrite rule can be viewed as an extension of both Reiter's *unique-names* assumption [22] and axioms (7), (8) and (9) of CET to the variables of the language. Under the Object Identity assumption, the Datalog clause

$$C = p(X) \ :- \ q(X, X), q(Y, a)$$

is an abbreviation for the Datalog^{OI} clause

$$C_{OI} = p(X) \ :- \ q(X, X), q(Y, a) \parallel [X \neq Y], [X \neq a], [Y \neq a]$$

where p, q denote predicate letters, X, Y are variables, a is a constant and the inequations attached to the clause can be seen as constraints on its terms. These constraints are generated in a systematic way by the (OI) rewrite rule. In addition, they can be dealt with in the same way as the other literals in the clause. Therefore, under Object Identity, any Datalog clause C generates a new Datalog^{OI} clause C_{OI} consisting of two components, called $\mathit{core}(C_{OI})$ and $\mathit{constraints}(C_{OI})$, where $\mathit{core}(C_{OI}) = C$ and $\mathit{constraints}(C_{OI})$ is the set of the inequalities generated by the (OI) rewrite rule, that is to say,

$$\mathit{constraints}(C_{OI}) = \{t \neq s \mid t, s \in \mathit{terms}(C), t, s \text{ distinct}\}$$

Therefore, Datalog^{OI} is a sublanguage of $\text{Datalog}^{\neq}$ (the extension of Datalog allowing inequality predicates in the body of clauses). Formally, a Datalog^{OI} program is made up of a set of Datalog^{OI} clauses of the form

$$q(X_1, X_2, \ldots, X_n) : -\varphi \parallel I$$

where q and φ are as in Datalog, I is the set of inequations generated by the (OI) rule and $n \geq 0$. The symbol "$\parallel$" means *and* just like ",", but is used for the sake of readability, in order to separate the predicates coming from the (OI) rewrite rule from the rest of the clause. In spite of the Object Identity bias, Datalog^{OI} has the same expressive power as Datalog as shown in the following propositions and proved in [25].

Proposition 2.1 $\forall C \in Datalog\ \exists C' = \{C_1, C_2, \ldots, C_n\} \subseteq Datalog^{OI}$: $T_C \uparrow \omega = T_{C'} \uparrow \omega$

that is, for each Datalog clause we can find a set of Datalog^{OI} clauses equivalent to it. It follows that:

Corollary 2.2 $\forall P \subseteq Datalog\ \exists P' \subseteq Datalog^{OI} : T_P \uparrow \omega = T_{P'} \uparrow \omega$.

that is, for any Datalog program we can find a Datalog^{OI} program equivalent to it.

Now, it is possible to introduce the ordering relation defined by applying the notion of Object Identity upon the classical relation of θ-subsumption, called θ_{OI}-subsumption [5, 25]. The following definition extends to Datalog the definition given in [6, 26] for constant-free (other than function-free) logic languages.

Definition 2.2 (θ_{OI}-subsumption ordering) *Let C, D be two Datalog clauses. We say that D θ-subsumes C under object identity (D θ_{OI}-subsumes C) if and only if (iff) there exists a substitution σ such that (s.t.) $D_{OI}.\sigma \subseteq C_{OI}$.*
In such a case, we say that D is more general than or equivalent to C (D is an upward refinement of C and C is a downward refinement of D) under object identity and we write $C \leq_{OI} D$. We write $C <_{OI} D$ when $C \leq_{OI} D$ and not$(D \leq_{OI} C)$ and we say that D is more general than C (D is a proper upward refinement of C) or C is more specific than D (C is a proper downward refinement of D) or D properly θ_{OI}-subsumes C. We write $C \sim_{OI} D$, and we say that C and D are equivalent clauses under object identity, when $C \leq_{OI} D$ and $D \leq_{OI} C$.

Like θ-subsumption, θ_{OI}-subsumption induces a quasi-ordering upon the space of the Datalog clauses, as stated by the following result [27].

Proposition 2.3 *Let C, D, E be Datalog clauses. Then:*

1. $C \leq_{OI} C$
2. $C \leq_{OI} D$ *and* $D \leq_{OI} E \Rightarrow C \leq_{OI} E$

that is, θ_{OI}-subsumption is both reflexive and transitive just like θ-subsumption. However, it is not anti-symmetric, hence the resulting space is not a lattice and, as a consequence, the *lub* and *glb* of two clauses could be not unique. A characterization of the notion of θ_{OI}-subsumption is:

Proposition 2.4 *Let C, D be two Datalog clauses, $C \leq_{OI} D \Leftrightarrow$*
$\exists \sigma s.t. core(D_{OI}).\sigma \subseteq core(C_{OI})$ *and* $constraints(D_{OI}).\sigma \subseteq constraints(C_{OI})$

An interesting fact is the existence, under such a weaker, but more mechanizable and manageable ordering, of ideal refinement operators, whose definition follows.

Definition 2.3 *Let C be a Datalog clause.*

- *Downward refinement operator ρ_{OI}:*
  $D \in \rho_{OI}(C)$ when exactly one of the following conditions holds:

1. $D \in C.\theta$, *where* $\theta = \{X/a\}$, $a \notin consts(C)$, $X \in vars(C)$
   i.e θ *is a substitution,* a *is a constant not occurring in* C *and* X *is a variable occurring in* C*;*
2. $D = C \cup \{\neg l\}$, *where* l *is an atom s.t.* $\neg l \notin C$.

- *Upward refinement operator* δ_{OI}*:*
  $D \in \delta_{OI}(C)$ *when exactly one of the following conditions holds:*
  1. $D \in C.\gamma$, *where* $\gamma = \{a/X\}$, $a \in consts(C)$, $X \notin vars(C)$
     i.e γ *is an anti-substitution,* a *is a constant occurring in* C *and* X *is a variable not occurring in* C*;*
  2. $D = C \backslash \{\neg l\}$, *where* l *is a n atom s.t.* $\neg l \in C$.

In other words, a specialization of a clause can be obtained by substituting a variable with a (new) constant or by adding a literal to its body. Conversely, a generalization of a clause can be obtained by substituting a constant with a (new) variable or by eliminating a literal from its body.

Proposition 2.5 *[5, 25] The refinement operators in Definition 2.3 are ideal for Datalog clauses ordered by* θ_{OI}*-subsumption.*

3 INTHELEX

INTHELEX (INcremental THEory Learner from EXamples) is a learning system for the induction of hierarchical theories from positive and negative examples which focuses the search for refinements by exploiting the Object Identity bias on the generalization model. It is fully and inherently incremental: this means that, in addition to the possibility of taking as input a previously generated version of the theory, learning can also start from an empty theory and from the first available example; moreover, at any moment the theory is guaranteed to be correct with respect to all of the examples encountered thus far. This is a fundamental issue, since in many cases deep knowledge about the world is not available. Incremental learning is necessary when either incomplete information is available at the time of initial theory generation, or the nature of the concepts evolves dynamically. The latter situation is the most difficult to handle since time evolution needs to be considered (see Section 5 for more details). In any case, it is useful to consider learning as a *closed loop* process, where feedback on performance is used to activate the theory revision phase [2]. INTHELEX can learn simultaneously various concepts, possibly related to each other, and is based on a closed loop architecture — i.e. the learned theory correctness is checked on any new example and, in case of failure, a revision process is activated on it, in order to restore completeness and consistency.

INTHELEX learns theories expressed as sets of Datalog^{OI} clauses (function free clauses to be interpreted according to the Object Identity assumption — see the previous section). It adopts a full memory storage strategy — i.e., it retains all the available examples, thus the learned theories are guaranteed to be

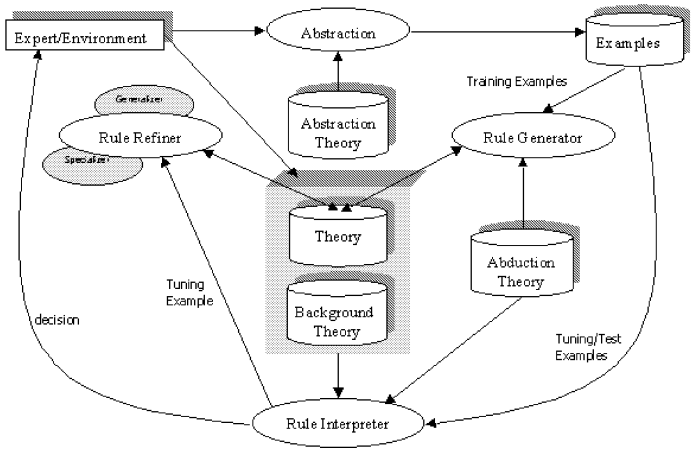


Figure 1: Inthelex Architecture

valid on the whole set of known examples — and it incorporates two inductive operators, one for generalizing definitions that reject positive examples, and the other for specializing definitions that explain negative examples. Both these operators, when applied, change the answer set of the theory, i.e. the set of examples it accounts for. Therefore, it is a system for theory revision rather than for theory restructuring, according to the definition of theory restructuring as a process that does not change the answer set of a theory [38].

3.1 Theory Revision

The process of theory revision, as performed by the system, is now briefly summarized. Figure 1 presents its logical architecture.

In order to perform its task, the system exploits a previous theory (if any) and a historical memory of all the past (positive and negative) examples that led to the current theory. It is important to note that a positive example for a concept is not considered as a negative example for all the other concepts (unless it is explicitly stated).

A set of examples of the concepts to be learned is provided by the *Expert*, possibly selected from the *Environment*. Examples are definite ground Horn clauses, whose body describes the observation by means of only basic non-negated predicates of the representation language adopted for the problem at hand, and whose head is the tag that defines the class to which the observed object belongs if it is a positive one. In case of a negative example, i.e. a description of an object that does not belong to a class, the head is negated. When learning many concepts, a single observation may stand as an example or a counterexample for more than one concept: The system allows the specification in the head of a list of all the classifications that can be associated to the observation described in the body. This does not affect the system's inherent

incrementality. Indeed, single classifications are processed separately, in the order they appear in the list, so that the teacher can still decide which concepts should be taken into account first and which should be taken into account later. Whenever a new example is taken into account, it is also stored in the *historical memory*.

The whole set of examples can be subdivided into three subsets, namely *training*, *tuning* and *test* examples, according to the way in which examples are exploited during the learning process. Specifically, training examples (if any), previously classified by the Expert, are exploited by the Rule Generator to build a theory that is able to explain them. The initial theory can also be provided by the Expert. Subsequently, such a theory, plus the Background Knowledge, are checked by the Rule Interpreter against new available examples. The Rule Interpreter takes the set of inductive hypotheses and a tuning/test example as input, and produces a decision. The decision is compared to the correct one and, in case of incorrectness, the cause of the wrong decision can be located.

Test examples are exploited only to check the predictive capabilities of the theory on new observations. Conversely, tuning examples are exploited incrementally by the *Rule Refiner* to modify incorrect hypotheses according to a data-driven strategy. The Rule Refiner consists of two distinct modules, a *Rule Specializer* and a *Rule Generalizer*. In particular, when a positive example is not covered, the Rule Generalizer produces a revised theory obtained in one of the following ways (listed by decreasing priority) such that completeness is restored:

- replacing a clause in the theory with one of its Least General Generalizations under Object Identity against the problematic example;
- adding a new clause to the theory, obtained by properly turning constants into variables in the problematic example;
- adding the problematic example as a positive exception.

When, on the other hand, a negative example is covered, the Rule Specializer outputs a revised theory that restores consistency by performing one of the following actions (by decreasing priority):

- adding positive literals that are able to characterize all the past positive examples of the concept (and exclude the problematic one) to the clauses that concur to the example coverage (starting from the lowest possible level);
- adding a negative literal that is able to discriminate the problematic example from all the past positive ones to the top-level clause in the theory by which the problematic example is covered;
- adding the problematic example as a negative exception.

An exception contains a specific reference to the observation it represents, as it occurs in the tuning set; new incoming observations are always checked with

respect to the exceptions before the rules of the related concept. This does not lead to rules which do not cover any example, since exceptions refer to specific objects, while rules contain variables, so they are still applicable to other objects than those in the exceptions. Section 4 will present a more detailed explanation of these processes.

Algorithm 1 shows the procedure implementing the Rule Refiner. It concerns the tuning phase of the system, where $M = M^+ \cup M^-$ represents the set of all negative (M^-) and positive (M^+) processed examples, E is the example currently examined, T represents the theory generated so far according to M plus the background knowledge. The *Generalize* and *Specialize* procedures implement the Rule Generalizer and Rule Specializer, respectively, used by the Rule Refiner.

Algorithm 1 Tuning Procedure

Procedure Tuning (E: example; T: theory; M: historical memory);
if E is a positive exampe AND $\neg$ *covers*(T,E) **then**
  Generalize(T,E,M^-)
else
  if E is a negative example AND *covers*(T,E) **then**
    Specialize(T,E,M^+)
$M := M \cup \{E\}$

It is worth noting that INTHELEX never rejects examples, but always refines the theory. Moreover, it does not need to know *a priori* what is the whole set of concepts to be learned, but it learns a new concept as soon as examples about it are available.

3.2 Multistrategy Learning

This purely inductive procedure has been further developed by providing it with additional multistrategy capabilities, according to the Inferential Theory of Learning framework [18], in order to improve effectiveness and efficiency of the learning task. Namely, *deduction* exploits the provided Background Knowledge (i.e. some partial concept definitions known to be correct, and hence not modifiable) to recognize known objects in an example description. *Abstraction* can be cast as the process of focusing on what is relevant in an observation. Indeed, ignoring the details about the objects belonging to a class may facilitate the generation of rules for that class. *Abduction* is used to complete the observations, whenever possible, in such a way that the examples they represent are explained (if positive) or rejected (if negative). This prevents the refinement operators from being applied, as long as possible, leaving the theory unchanged.

For deduction the system exploits a *dependency graph*, describing the dependence relationships among the concepts to be learned (see Figure 2 for a graphical example concerning the vehicle domain). The relations among the concepts represented in such a graph are expressed as a set of clauses like the

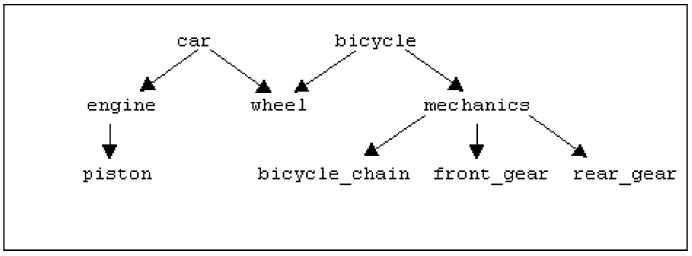


Figure 2: Dependencies graph

following:

`car`(X) $\leftarrow$ `engine`(X), `wheel`(X).

`engine`(X) $\leftarrow$ `piston`(X).

`bicycle`(X) $\leftarrow$ `wheel`(X), `mechanics`(X,Y).

`mechanics`(X,Y) $\leftarrow$ `bicycle_chain`(X,Y,Z), `front_gear`(X,Y),
`rear_gear`(X,Y).

"concept $car/1$ depends on concepts $engine/1$ and $wheel/2$"; "concept $engine/1$ depends on concept $piston/1$"; "concept $bicycle/1$ depends on concepts $wheel/1$ and $mechanics/2$"; "concept $mechanics/2$ depends on concepts $bicycle\_chain/3$, $front\_gear/2$ and $rear\_gear/2$". Note that in the graph the variables are used as placeholders to indicate the arity of concepts. Whenever a new example is taken into account, and before it is stored in the historical memory, if necessary, it undergoes a saturation phase. If any of its sub-concepts in the dependency graph can be recognized in its description according to the definitions learned by the system up to that moment and the Background Knowledge, literals concerning those concepts are added (properly instantiated) to the example description.

The background knowledge is expressed in the same way as the rules of the theory (i.e., Horn function-free clauses in Prolog form), but the body of its clauses starts with a "`true`" predicate in order to recognize them and prevent them from being modified by the refinement operators. An example of such a clause is:

`wheel`(X) $\leftarrow$ `true`, `circular`(X), `has_rim`(X,C), `has_tire`(X,T).

and is to be interpreted as "An object X is a wheel if it is circular, it has a rim C and a tire T". Given the above setting, and the following example:

`bicycle`(bb) $\leftarrow$ `has_saddle`(bb,s), `has_pedals`(bb,p), `has_frame`(bb,f),
`part_of`(bb,$w1$), `circular`($w1$), `has_rim`($w1$,$r1$),
`has_tire`($w1$,$t1$), `part_of`(bb,$w2$), `circular`($w2$),
`has_rim`($w2$,$r2$), `has_tire`($w2$,$t2$).

since concept 'bicycle' depends on concept 'wheel', and in the description for bicycle *bb* it is possible to recognize two wheels ($w1$ and $w2$), according to the definition in the background knowledge, the example can be saturated in order to explicitly represent such a situation in the following way:

```
bicycle(bb) ← has_pedals(bb,p), has_saddle(bb,s), has_frame(bb,f),
              part_of(bb,w1), wheel(w1), part_of(bb,w2), wheel(w2),
              circular(w1), has_rim(w1,r1), has_tire(w1,t1),
              circular(w2), has_rim(w2,r2), has_tire(w2,t2).
```

In a similar fashion, an abstraction phase, performed by a proper module, is able to preprocess the examples in order to describe them in a higher-level language, by eliminating superfluous details according to a given Abstraction Theory. Specifically, the implemented abstraction operators [39] allow to eliminate superfluous details, group specific component patterns into compound objects, reduce the number of object attributes, ignore the number of instances of a given object and obtain a coarser grain-size for attribute values. Again, the abstraction operators are represented as clauses, such that whenever their body is recognized in an example, the corresponding literals are dropped from the example and replaced by those in the head (properly instantiated). Unlike the background knowledge, the body of clauses that represent the abstraction theory does not start with a "`true`" predicate, and there is no reference to the dependency graph in the application of abstraction. Referring again to the previous example about bicycle *bb*, and supposing that the background knowledge clause is now in the Abstraction Theory, the example can be abstracted in the following way:

```
bicycle(bb) ← has_pedals(bb,p), has_saddle(bb,s), has_frame(bb,f),
              part_of(bb,w1), wheel(w1), part_of(bb,w2), wheel(w2).
```

Lastly, INTHELEX is also able to exploit a special abductive proof procedure to manage situations in which not only the set of all observations is partially known, but each observation could be incomplete too. In particular, an algorithm by Kakas and Mancarella [14], modified by Esposito *et al.* [7], has been adopted for hypothesizing unknown facts to be added to an observation, provided they are consistent with given integrity constraints. Specifically, to be able to abduce literals, the system needs to know the set of integrity constraints (IC) that the hypothesized information must fulfill, and the list of predicates on which abductions can be made, called *abducibles* (A). For instance, supposing to have the following set of abducibles:

$$A = \{circular(X), has\_rim(X, R), has\_pedals(X, Y)\}$$

an integrity constraint stating that "an object X is either circular or square (but it cannot be both at the same time)":

$$IC : [circular(X), square(X)]$$

and the following example:

bicycle(bb) ← has_saddle(bb,s), has_frame(bb,f), part_of(bb,$w1$),
has_rim($w1$,$r1$), has_tire($w1$,$t1$), part_of(bb,$w2$),
circular($w2$), has_tire($w2$,$t2$).

the abduction procedure is able to complete the example description according to the following hypothesis (previously generated by the system or contained in the background knowledge):

bicycle(X) ← has_saddle(X,Y), has_pedals(X,Z), has_frame(X,W),
part_of(X,A), circular(A), has_rim(A,R), has_tire(A,S),
part_of(X,B), circular(B), has_rim(B,T), has_tire(B,V).

by adding the predicates: has_pedals(bb,$s0$), has_rim($w1$,$t1$), circular($w2$).

Algorithm 2 shows the tuning procedure, augmented with such functionality, where $AbsT$ is the abstraction theory provided by the user; $SatE$ and $AbsE$ are, respectively, the example generated by the saturation and abstraction phases from the original example E; D is the set of literals returned by the abductive derivation of example $AbsE$ in theory T (that also includes the background knowledge).

Algorithm 2 Theory Tuning

Procedure Tuning (E: example; T: theory; M: historical memory);
$AbsE$:= Abstract(E,$AbsT$)
D := $AbsE$
if ∃ Abductive_Derivation($AbsE$,T,D) **then**
  Add to D the abduced literals
  Add D to body($AbsE$)
else
  $SatE$:= Saturate($AbsE$,T)
  if $SatE$ is a positive example AND ¬ $covers$(T,$SatE$) **then**
    Generalize(T,$AbsE$,M^-)
  else
    if $SatE$ is a negative example AND $covers$(T,$SatE$) **then**
      Specialize(T,$AbsE$,M^+)
M := $M \cup AbsE$

4 Inductive Refinement Operators in INTHELEX

Whenever a positive example is not explained by the currently available theory (*omission* error), it is necessary to generalize the incomplete concept definition (i.e., the set of clauses defining that concept). Conversely, when a negative example is explained (*commission* error), it becomes necessary to specialize all the inconsistent clauses. This points out that the process of diagnosis of an incorrect theory is performed at different levels of granularity, according to the type of error found. Specifically, if an omission error occurs, a single

clause cannot be blamed for not explaining the example, but the whole set of clauses defining that concept is equally responsible; if a commission error occurs, the diagnosis can be carried up to the level of specific clauses that explain the example. Omission errors can be solved by exploiting an upward refinement operator, while, dually, downward refinement operators can cope with commission errors.

4.1 Upward refinement

The upward refinement operator extends the concept of *least general generalization* (*lgg*) introduced by Plotkin [21] to cope with the ordering induced by θ_{OI}-subsumption.

Definition 4.1 (Least general generalization under Object Identity) *A* least general generalization under θ_{OI}-subsumption *of two clauses is a generalization which is not more general than any other such generalization, that is, it is either more specific than or not comparable to any other such generalization. Formally:*

$$lgg_{OI}(C_1, C_2) = \\ = \{C \mid C_i \leq_{OI} C, i = 1, 2 \text{ and } \forall D \text{ s.t. } C_i \leq_{OI} D, i = 1, 2 : \neg(D <_{OI} C)\}$$

Note that the *lgg* is no longer unique under θ_{OI}-subsumption, since the space of Datalog clauses is not a lattice when ordered by θ_{OI}-subsumption [26, 27] while it is when ordered by θ-subsumption.

Example 4.1 *Given:*

$$C_1 = bicycle(a) \ :- \ wheel(v, a), wheel(w, a), spikes(r, v), spikes(s, w).$$

$$C_2 = bicycle(b) \ :- \ wheel(v, b), wheel(Y, b), spikes(R, Y).$$

we have:

$$lgg_{OI}(C_1, C_2) = \{bicycle(X) \ :- \ wheel(v, X), wheel(Z, X), spikes(T, Z).\}$$

while the clause

$$E = bicycle(X) \ :- \ wheel(v, X), wheel(Z, X).$$

is a generalization but not a least general one.

Implementing the theoretical ideal generalization operator just as it is defined would require to compute the Plotkin's *lgg*, to generate all of its subsets (if n is the number of literals in Plotkin's *lgg*, this means generating 2^n distinct subsets) and, for each of them, to check them both internally (to ensure OI and linkedness) and externally (comparing it with all the others to ensure minimality). Progressively generating and expanding the graph of generalizations, starting from Plotkin's one, until all lgg_{OI} are found, would avoid the generation of all possible subsets, but would also generate duplicates, thus not reducing the

Algorithm 3 Generalization

Procedure Generalize (E: positive example, T: theory, M^-: negative examples);
$L :=$ (non BK) clauses in the definition of the concept which E refers to
generalized := false; $gen\_lgg := 0$
while ((¬ generalized) AND ($L \neq \emptyset$)) **do**
  $C :=$ *choose_clause*(L) ; $L := L \backslash \{C\}$
  while (($\exists$ another $l \in lgg_{OI}(C, E)$) AND (¬ generalized)
  AND ($gen\_lgg \leq max\_gen$)) **do**
    $gen\_lgg := gen\_lgg + 1$
    if (*consistent*$(T \backslash \{C\} \cup \{l\}, M^-)$) **then**
      generalized := true ; $T := T \backslash \{C\} \cup \{l\}$
if (¬ generalized) **then**
  $G :=$ *turn_constants_into_variables*(E)
  if *consistent*$(T \cup \{G\}, M^-)$ **then**
    $T := T \cup \{G\}$
  else
    $T := T \cup \{E\}$ {positive exception}

exponential complexity of the process. Both these solutions are too computationally expensive. Thus, our implementation works as follows: The lgg_{OI} is initialized as the literal generalizing the two heads; then, literals in Plotkin's *lgg* are progressively added to the partial generalization as long as they are linked to it and fulfill the OI requirement. When a generalization is found, it is tested with respect to all the past negative examples. If none of them is explained, then the generalized clause is replaced in the theory by the generalization. Otherwise, an intelligent backtracking on the literals in Plotkin's *lgg* is performed, which yields the next generalization avoiding to build equivalent ones.

The generalization process, described in Algorithm 3, is started whenever a positive example is not explained. Starting from the current theory, the misclassified example and the past negative examples, it ends with a new (revised) theory. First of all, since (as already pointed out) there is no single clause responsible for not explaining the given example, the system must perform a step of blame assignment, choosing a clause to be generalized among those making up the involved concept description. This is done by the *choose_clause* function, that currently just selects a clause at random.

Then, the system tries to compute one of the lgg_{OI}'s of this clause and the example (see Algorithm 4). If it is consistent (which is checked by the boolean function *consistent*) with all the past negative examples, then it replaces the chosen clause in the theory, or else another generalization in the lgg_{OI} is generated and tested (if any). In case no consistent generalization of that clause is found, a new clause is chosen to compute the lgg_{OI}. For each clause, the following bounds can be imposed, according to the degree of generalization to be obtained and the computational budget allowed:

- maximum number of elements in the lgg_{OI} to be generated (*max_gen* in Algorithm 3)
- maximum number of elements in the lgg_{OI} to be discarded[1] (*max_disc* in Algorithm 4)
- maximum number of literals allowed in the generalization[2] (*max_lits* in Algorithm 4)
- minimum number of literals allowed in the generalization (*min_lits* in Algorithm 4)

If no clause in an incomplete definition can be generalized so that the resulting theory is complete and consistent, the system checks if the example itself, with the constants properly turned into variables (which is obtained by means of a function *turn_constants_into_variables*), is consistent with the past negative examples. Such a clause is added to the theory, or else the example itself is added as an exception to the theory.

4.2 Downward refinement

In our framework, specializing means merely adding proper literals to an inconsistent clause, in order to avoid it explaining a negative example. Revisions performed by this operator are always minimal [37]: indeed, a specialization obtained by turning a variable into a constant is not provided for since all clauses in the theory contain only variables as arguments.

Starting from the current theory, the misclassified example and the past positive examples, the specialization algorithm yields a new (revised) theory in which the consistency property has been restored by adding proper literals to inconsistent clauses, according to the ideal operator in Definition 2.3. The space in which the literals to be added should be searched for is potentially infinite and, in any case, its size is so large that an exhaustive search is not feasible. The operator can focus the search into the portion of the space that contains the solution of the diagnosed commission error, as a result of an analysis of its algebraic structure. The search is firstly performed in the space of positive literals, that contains information coming from positive examples used to synthesize the current theory, but not yet exploited by it. If the search in this space fails, the algorithm autonomously performs a *representation change*, that allows it to extend the search to the space of negative literals, built by taking into account the negative example that caused the commission error [27]. The pseudo-code procedure is described in Algorithm 5.

First of all, the process detects all the clauses that caused the inconsistency (i.e., those occurring in the SLD-derivation of the example). Then, the system tries to specialize one at the lowest possible level (which corresponds to a clause

[1] Which happens if it is not consistent with all the past negative examples or its construction do not respect the generality limits imposed.

[2] A literal is discarded if it does not comply with the linkedness and OI requirements.

Algorithm 4 lgg

Function lgg_oi(E: positive example, C: clause): clause;
$S_{neg} := S_{pos} := L := Subst_{lgg} := \emptyset; disc\_lits := 0$
for all $n \in$ negative literals of C **do**
  $S := \{\theta | n\theta \in E\}$; $S_{neg} := \{\sigma = \theta\gamma | \theta \in S_{neg}, \gamma \in S\}$
for all $p \in$ positive literals of C **do**
  if $\exists\theta$ s.t. $p\theta \in E$ **then**
    $T := \{\theta \mid p\theta \in E\}$; $S_{pos} := \{\sigma = \theta\gamma \mid \theta \in S_{pos}, \gamma \in T\}; L := L \cup \{p\}$
  else
    $disc\_lits := disc\_lits + 1$
$lgg_{OI} := head(C)$; $Subst_{lgg} := \theta$ s.t. $head(C)\theta = head(E)$
$K := \{k \in L$ s.t. $vars(lgg_{OI}) \subseteq vars(K)\ \}$
$lgg_{OI} := lgg_{OI} \cup \{K\}; L := L \setminus \{K\}$
$Subst_{lgg} := \{\sigma = \theta\delta \mid \theta \in Subt_{lgg}, \delta \in \{\delta' \mid K\delta' \subseteq E\}\}$;
while (($\mid lgg_{OI} \mid < max\_lit$) AND ($L \neq \emptyset$) AND ($disc\_lits < min\_lit$)
AND ($disc\_clauses < max\_disc$)) **do**
  $c := l \in L$ s.t. $linked(\{l\} \cup lgg_{OI})$
  if ($lgg_{OI} \cup \{c\}$) fulfills the OI requirements **then**
    $M := \{l \in L \mid vars(l) \subseteq vars(lgg_{OI} \cup \{c\})\ \}$; $L := L \setminus \{M\}$
    $lgg_{OI} := lgg_{OI} \cup \{c\} \cup \{M\}$
    $Subst_{lgg} := \{\sigma\theta\gamma \mid \sigma \in Subst_{lgg}, \theta \in \{\theta' \mid c\theta' \in E\}, \gamma \in \{\gamma' \mid M\gamma' \subseteq E\}\}$
  else
    $disc\_lits := disc\_lits + 1$
    if $L := L \setminus \{c\}$ linked wrt negative literals of C **then**
      $disc\_clauses := disc\_clauses + 1$
if (($\mid lgg_{OI} \mid > max\_lit$) AND ($Subst_{lgg}$ is compatible with S_{neg})) **then**
  return (lgg_{OI})

defining a concept whose level is the lowest in the dependency graph), in order to refine more "basic" concepts. Indeed, they may appear in the definition of many higher level concepts; the underlying heuristic is that the latter will hopefully benefit from the revised/better definition of the former. Since the downward refinements we are looking for must satisfy the property of maximal generality, this suggests to search for one (or more) positive literal(s) which can discriminate all the past positive examples from the current negative one. Specifically, if there exists one (or a combination of) literal(s) that, when added to the body of the clause to be specialized, is able to discriminate from the negative example that caused the inconsistency, then the downward refinement operator should be able to find it. If the derivation of an example in a theory is not unique, a single specialization step could be insufficient to restore the consistency of the theory. For such a reason, when a specialization of a clause is reached, the example is tested again to check if another SLD-derivation, different from the previous one, exists for it. If so, the process is iterated until no more derivations of the example are found. Note that only one step of specialization

Algorithm 5 Specialization

Procedure Specialize(E: negative example; T: theory; M^+: positive examples);
specialized := false ; $gen\_spec := 0$
while $\exists$ a (new) derivation D of E from T **do**
  $L :=$ Input program clauses in D sorted by decreasing depth in derivation
  while ((¬ specialized) AND ($\exists C \in L$) and ($gen\_spec \leq max\_gen$)) **do**
    while (($\exists$ another $S \in \rho_{OI}\_pos(C, E)$) AND (¬ specialized)) **do**
      $gen\_spec := gen\_spec + 1$
      if ($complete(T \setminus \{C\} \cup \{S\}, M^+)$) **then**
        $T := T \setminus \{C\} \cup \{S\}$
  if (¬ specialized) **then**
    $C :=$ first clause in the derivation of E
    while (($\exists$ another $S \in \rho_{OI}\_neg(C, E)$) AND (¬ specialized)) **do**
      if ($complete(T \setminus \{C\} \cup \{S\}, M^+)$) **then**
        $T := T \setminus \{C\} \cup \{S\}$
  if (¬ specialized) **then**
    $T := T \cup \{E\}$ {negative exception}

is needed whenever the SLD-derivation of the example involves just one clause. Such a process can be formalized in Algorithm 6.

If no clause in the SLD-derivation of the example can be specialized by adding literals coming from the positive examples, an attempt is made to add a negative literal to the first clause of the SLD-derivation (the one related to the concept the example is an instance of). Specifically, the literals present in the negative example that caused the commission error, but not in the clause to be specialized, are taken into account. Indeed, if any, their negation added to the clause would be able to discriminate the negative example from all the past positive ones. Algorithm 7 shows this process in more detail.

A set of parameters for limiting the search are considered:

- maximum number of positive literals to be added (*max_lit* in Algorithm 6)
- maximum number of specialization to be generated (*max_gen* in Algorithm 5)

If none of the clauses obtained makes the theory complete and consistent again, then the system adds the negative example to the theory as an exception. In such a case is not necessary to iterate the specialization step since an exception specifically refers to the example it represents. Hence, it cannot be derived in any other way.

5 Incremental Learning and Concept Drift

So far we have discussed how the system performs online learning through refinement operators. The underlying assumption was that there exists a target

Algorithm 6 rho_pos

Function rho_oi_pos(C: clause; E negative example): clause;
$Pos :=$ past positive examples θ_{OI}-subsumed by C
$P :=$ shortest example in Pos
$C' := C$
$A := \emptyset$ {list of literals added to C so far}
while ($(\exists\theta$ s.t. $C\theta \subseteq P)$ AND $(added\_lit < max\_lit)$) **do**
  $Subst := \{\theta \mid C\theta \subseteq P\}$; $A\_Subst := \{\sigma^{-1} \mid \sigma \in Subst\}$
  $NewP := P\theta$ s.t. $\theta \in A\_Subst$
  $Res\_pos := body(C) \setminus body(NewP)$
  while $((Res\_pos \neq \emptyset)$ AND $(added\_lit < max\_lit))$ **do**
    $Path := \{L \subseteq A \mid linked(L)\ \}$; $Res\_pos := Res\_pos \setminus Path$
    $p := l \in Res\_pos$ s.t. $linked(\{l\} \cup C')$
    $Res\_pos := Res\_pos \setminus \{p\}$; $A := A \cup \{p\}$; $C' := C' \cup \{p\}$
    if $(explains(T \cup \{C'\}, E))$ **then**
      $added\_lit := added\_lit + 1$
    else
      return C'

Algorithm 7 rho_neg

Function rho_oi_neg(C: clause; E negative example): clause;
$Subst := \theta$ s.t. $C\theta \subseteq E$; $Res\_neg := body(E\theta^{-1}) \setminus body(C)$
while $Res\_neg \neq \emptyset$ **do**
  $n := l \in Res\_neg$ s.t. $vars(l) \subseteq vars(C)$
  $Res\_neg := Res\_neg \setminus \{n\}$
  $C' := C \cup \{not(n)\}$
  if $(\neg\ explains(C', E))$ **then**
    return C'

meaning of the concepts to be learned which is stable with respect to its context, i.e. external conditions that may influence the observation of examples. Extending the applicability of the learning framework by discarding this assumption makes incremental learning an even more complex task. Indeed, it is possible to identify at least three main subproblems of learning in the presence of changing conditions [29]: *detection*, *adaptation* and *characterization*. First, changes must be detected since they may yield changes in the current concept definitions. Then a proper adaptation of the concept characterization (i.e., of its model) must take place in order to comply with the change occurred.

The *context* of learning is represented by any information that is relevant to this process. Yet, providing all relevant aspects of a concept is often beyond the means of a teacher and, even though all the required features are included, complications may still arise in applications where the training examples do not permit the learning agent to detect their relevance to the learning problem [36]. Then, it is necessary to understand which specific hypotheses have to

be made for adapting with respect to a given change. These problems have been tackled so far by exploiting abstraction and abduction. As mentioned in Section 3, abstraction operators can focus learning towards the target concept on the ground of relevant items while abduction can be a means for trying to complete the observations, hypothesizing the sources of incompleteness.

However, the context of learning can change due to hidden factors whose most likely, yet not exclusive, representative is often the passing of time. Then, the accuracy of an incremental learner, that has been trained on examples from a single context, might turn out to be poor when the context changes. In the reversed perspective, a decay in the accuracy of the induced hypotheses may imply that a context change has happened. Detecting hidden changes of context has been tackled by techniques such as *contextual clustering* [10], which solve the problem of grouping observations that share the same context.

Problems arise when examples are collected in batches that belong to different contexts, such as different time periods. Indeed, gradual or also abrupt changes take place in the form of *concept drift* [24]. A typical example from the user modelling domain can be the case of using logs of customers' transactions in order to discover preferences, interests and/or behavioral patterns, all subject to changes along time. Tracking the change may be dramatically important for customization and modelling issues. The same happens in the case of information filtering problems, when both user interests and document contents may change over time. Even more so, in some domains, such as financial prediction or dynamic control, changes of context can be assumed to recur [35]. The learning systems should be able to detect the situations of discontinuity and adapt to the changed conditions. Moreover, suitable diagnostics may be employed to describe the nature of the changes that occurred. This could be exploited at a meta-level to discover recurring patterns and understand the causes (or just *clues*) for the change [34].

Tracking concept drift in incremental learning requires the system to continually audit the accuracy of the hypotheses produced in the past and adjust them whenever necessary. Most likely it may also be necessary to forget the oldest and outdated information. The process should turn out to be, at the same time, flexible and robust enough not to be misled by noise [33]. As regards the problem of collisions between the classification of similar examples, it has been investigated on discarding an example when a newer observation is available located in a similar region of the search space [23].

A solution to all such different problems posed by learning in changing domains would require a complete framework and algorithms for learning at different levels. This goes beyond the scope of the present work. The approach adopted in this paper does not address the problem of detecting the changes: rather, we want to adapt to the drift along the time, assuming that a change has taken place when the Expert judges the accuracy of the latest hypotheses to be poor; then, a set of new examples is made available to the system so that it can revise the current concept definitions accordingly. The system is not asked to forget all about the past observations, yet it should assume a decaying degree of reliability from the latest to the oldest ones. Turning to our previous case from

user modelling, the Expert (or the online system itself) may sense deficiencies in the performance of the current descriptions (the induced theory) that are employed for tracking user interests and explain how to detect them. In such a case, he should decide to provide a new group of training observations so that the system can update the theory.

In the following sections, we illustrate in more detail this approach to managing the concept drift in the context of online learning and present the algorithm that implemented this model in the system INTHELEX, showing several experiments of its usage.

5.1 Evolving Concept Definitions

The common way for adapting to the concept drift is to use a *window* of recent instances [19][9][35][15]. This has allowed even batch learners to be adapted to online learning tasks such as this one. In this context, the most important factor to be decided is the window size. For a window of constant size, the choice of its width spans from small windows, implying fast adaptivity, up to larger ones giving good generalizations in periods where the concepts has not changed. A compromise is trying to adapt the size of the window to the *extent* of the concept drift (that is proportional to the probability of a disagreement of two hypotheses on the same example). This extent is to be limited to a certain size to have the concept learnable [11]. A proper window on the training data should include those examples which are close enough to the target concepts. Previous approaches either determine the window size (which is always a strong commitment) or adopt complex heuristics based on several parameters, often to be tweaked depending on the specific application. The principal shortcoming of both approaches is that examples that are excluded from the selected window will not play a role in learning/refining the next hypothesis.

In our approach concept drift can be tracked by allowing the system to partially forget some examples that were previously taken into account (past contexts). Rather than trying to detect window widths and applying the online learning algorithm therein, we assume the change happened with the provision of a new batch of examples, which triggers the refinement of the current hypotheses based on both these new examples and the old ones, although less and less confidence is placed in the latter as long as they are more outdated.

It seems plausible to assume that, with the passing of time, some of the examples that were employed for training the learner may loose their pregnancy with respect to the evolving concept. This decay can be modelled by requiring that the online learner, although endowed with a complete memory of the training examples, should ensure the explanation of the past cases with a decreasing effectiveness which is proportional to their age. Indeed only the newer set would require a total fulfillment of correctness (full completeness and consistency) whereas older sets of examples would have to be treated as less adequate to the evolved concept, thus one would gradually require less correctness of the hypotheses with respect to them. We will suppose that the degree of forgetfulness of the learner towards the past examples may change along with functions

Algorithm 8 Hypothesis Refinement with respect to Concept Drift

```
Function Drift(E, Hyp, TrainingSets, n, alpha, beta): newHyp
{       E: New example
        Hyp: current hypothesis
        TrainingSets: array of training sets
        n: number of training sets
        alpha: decrease factor for completeness
        beta: decrease factor for consistency
        newHyp: new hypothesis                }
completenessRate := 1; {hypothesis complete wrt 100% of exs}
consistencyRate := 1; {hypothesis consistent wrt 100% of exs}
repeat
  newHyp := refine(E, Hyp, TrainingSet[1]);
  k := 1 ; ok := true
  while ok AND k ≤ n do
    ok := (completeness(newHyp,TrainingSets[k]) ≥ completenessRate)
          AND (consistency(newHyp,TrainingSets[k]) ≥ consistencyRate))
    completenessRate := alpha * completenessRate;
    consistencyRate := beta * consistencyRate;
    k := k + 1;
until ok
return newHyp;
```

whose decay can be determined by setting proper parameters.

5.2 Implementation in INTHELEX

The algorithm that implements the approach explained above, as realized in the incremental learning system INTHELEX, is depicted in Algorithm 8. The two parameters α and β in the algorithm, ranging in $[0, 1]$, represent the variation of degree of forgetfulness towards the past training examples measured, respectively, in terms of completeness and consistency. The standard behavior of the system (full completeness and consistency with respect to the entire memory) is obtained with $\alpha = \beta = 1$. A simplifying assumption is made that the decay varies exponentially. Yet, a different function forms could be adopted knowing more information about the way the concept drift is taking place. We are currently investigating also on specific methods for estimating the parameters.

The whole set of learning examples that are (incrementally) made available by the Expert in different moments can be ideally grouped according to the version of the drifting concept they refer to. Let us call *TrainingSets* the sequence of all these groups, and let us refer by and index to each single group therein (the Expert can specify when a drift occurs, this way causing the current group to be completed and a new one to be created). Thus, *TrainingSets*$[i]$ contains the set of examples that refer to the i-th version of the drifting concept. For the sake of readability, let us suppose that, at any moment, lower indexes represent more

recent versions of the concept. *TrainingSets*[1], which is the latest group, also determines the search of refined versions of the hypotheses in procedure *refine*() through the mechanisms described in the previous sections. Each new example provided by the Expert causes an update of the *TrainingSets* (by adding it to the latest group or, if it immediately follows a drift mark, by creating a new group made up of just this example), and, in case it is misclassified, this procedure may generate several plausible hypotheses *newHyp* that are able to explain it. The choice among these versions is carried out as follows.

The outer loop produces different hypotheses by always requiring full completeness and consistency with respect to the examples in *TrainingSets*[1], which yields the most recent information about the drifting concept. Each generated hypothesis is checked, by the inner loop, on the other (older) groups, by progressively adjusting the new rates of completeness and consistency as they will be required for each of them according to the exponential decay function referred to above. Indeed, as previously mentioned, we suppose that the new hypotheses, although induced from the latest group, should guarantee at least a limited rate of correctness with respect to the older ones. If the required threshold is not passed in any of them, then the control returns to the other loop that generates a new hypothesis, otherwise the algorithm ends by returning the computed refinement. In this way, the constraining effect of the past groups decays as long as older groups are taken into account. Hence, the returned hypothesis guarantees full correctness with respect to the latest group and partial correctness with respect to the others.

The actual implementation requires the inner loop to be repeated only for a limited number of the latest groups, until a minimal threshold of correctness is reached, since the following (older) groups would not be extremely constraining and thus become less important. Note that the inherent incrementality of INTHELEX is preserved, since the Expert must not specify all at once the entire history of data, but examples can be provided singularly and the system will properly add them to *TrainingSets*, automatically updating both the sequence (before handling it) and the current model (without completely rejecting it) accordingly. Note, also, that old data (previous groups) influence the behavior of the system in that the proper correctness threshold must be passed on each of them (inner loop of the algorithm) for the refined hypothesis to be accepted; thus, it would not be the same learning the theory only on the latest group, without regard to the others. Obviously, running a learning system only on the latest group would certainly yield a theory which is correct with respect to the current version of the drifting concept, but, differently from the proposed algorithm, no information on previous instances and models would be reused, which was one of the motivations for the use of incrementality to handle concept drifting.

5.3 Recent Related Methods

Recently, the research has come focusing on the problem of mining open-ended data streams entered in large and constantly growing databases (an example

source of millions of records per day), e.g. those containing website logs, bank transactions, telecommunication logs. Indeed, in these situations, maintaining the memory of all the examples becomes difficult (even in the presence of tertiary storage systems). Extracting datasets by sampling these databases cannot be interpreted as drawing from a stationary distribution, as assumed by many statistical learning algorithms. Indeed, when the examples can be assumed to come from an unbound stream rather than a limited (representative) set, saving memory becomes a crucial point. Then, it may be important to store summaries or synopses of the past examples, i.e. forms of retaining a partial memory of the past data, or sampling the stream in an intelligent way, trying to preserve the statistical properties of the sample with respect to the whole stream [1].

The solution appears to be the employment of anytime and incremental methods so that the risk loosing potentially valuable information be minimized. The shortcoming of these methods is that they often cannot guarantee that the quality of the learned model be the same as that of model learned on the same examples in batch mode. Hence, the need arises for systems that grant a limited, often fixed, amount of time and memory per each example for its processing [4] (such as the system CVFDT [12]). These approaches are able to emulate learning within a moving window with a lower complexity per example (almost constant).

Our method is similar, in the sense that incremental learning can affect those parts of the induced candidate theory which appear to be weaker with respect to the incoming examples. Besides, learning in a richer representation such as FOL, the main difference is the complexity in terms of space. Our method is full memory and thus appears to be unfit to scale with very large bases of examples. In our setting the incremental refinement involves the latest version of the theory (a form of partial memory) and most recent group of examples. Thus the complexity is bounded by the cardinality of this group, even though the candidate hypotheses produced are checked for correctness against the past examples. However it should be recalled that the consistency and completeness parameters can be tweaked so that the older examples give a smaller contribution in the induction of the newer candidate versions of the model. Moreover, the use of the threshold limits the number of examples that are taken into account in the tests required by the algorithm. An improvement might be that of exploiting a sort of partial memory. This could store those past examples which triggered the refinement process and/or the past versions of the theories that can act as synopses of the past examples and use this memory instead of them.

Outside the scope of supervised methods, other incremental learning algorithms for data streams have appeared in recent years (e.g. in *clustering* [8] or in mining association rules). However, most of the literature on online algorithms focuses on weak learners [16] because of the hardness of proving theoretical results about stronger learners, especially when a FOL representation is adopted, as in our case.

5.4 Experimentation of the Proposed Approach

The behavior of INTHELEX when dealing with concept drifting according to the previously exposed strategy has been checked and analyzed by running the system on two purposely designed datasets.

One dataset is a completely artificial one, inspired to that proposed in [24] and exploited also in [35], but extended in order to raise the level of difficulty. Specifically, instead of 3 attributes with 3 values each (for a total of 27 combinations), here each observation is described by means of 4 different descriptors (namely *size*, *color*, *shape* and *position*), each ranging between 4 possible values:

- size: *small*, *medium*, *large*, *very large*;
- color: *black*, *red*, *green*, *blue*;
- shape: *square*, *rectangular*, *circular*, *oval*;
- position: *up*, *down*, *left*, *right*.

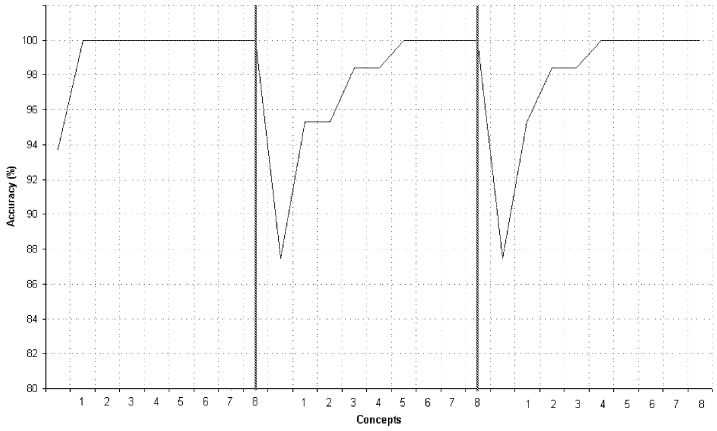
The resulting domain refers to a world made up of 256 objects (instead of the 27 in [24]), to be partitioned into positive and negative examples according to the target concepts to be learned. Three artificial target definitions were generated as made up of two descriptors with a fixed value, so that each of them splits the whole set into 16 positive instances and 240 negative ones. For each definition the whole set was partitioned into 8 subsets, each made up of 2 positive and 30 negative examples chosen at random. Then, INTHELEX was run on these subsets in sequence, so that the corresponding examples were accumulated in the historical memory. The predictive accuracy of the intermediate theories (i.e., the theories obtained after processing each subset) was checked on the whole set and tracing the evolution of the predictive accuracy as long as more and more examples were processed. After processing all the examples referring to a given concept (which ensures a 100% accuracy for that concept, according to the system characteristics), the new example set for the next concept was considered and employed, according to proper parameters for the required accuracy on past groups[3].

[3]Let us explain this in other words. INTHELEX was first run on the subset 1 of the first concept, and the resulting theory was tested on the whole set of examples for the first concept; then it was incrementally refined according to the subset 2 of the first concept (whose examples were added to the previous ones in the system's historical memory), and the resulting theory was again tested on the whole set of examples for the first concept; and so on until the subset 8 for the first concept was taken into account. At this stage, the drift occurred, and the historical memory contained all the examples for the first concept, that also constitute Group 1 for the drift-handling algorithm. The system was then provided with the subset 1 of examples for the second concept, by requiring 100% accuracy on them and less accuracy on those of the first concept, and the resulting theory was tested on the whole set of examples for the second concept; then such a theory was incrementally refined according to the subset 2 of the second concept (whose examples were added to all those considered so far in the system's historical memory), and the resulting theory was again tested on the whole set of examples for the second concept; and so on until the subset 8 for the second concept was taken into account. At this moment, another drift occurred: Group 2 for the drift-handling algorithm was complete, and the same procedure was applied to the third concept.

A first experimentation concerned three randomly generated concepts, each having a description that is completely different from that of the preceding one. In this case, parameters α and β were obviously set to 0, thus allowing the system to completely ignore previous examples. The resulting theories, and their evolution (reported in Figure 3), reveal that, independently of the parameter settings, the system is somewhat biased by the previous definitions, and cannot completely and abruptly withdraw them. This happened because the system setting allowed it only to modify learned clauses, without having the possibility of dropping any of them. This results in theories that initially show a natural drop in accuracy in correspondence to the first few examples of each drift (because the concept has changed but the theory has not yet accounted for it). After, as more and more subsets of examples for the new version of the concept are considered, they are progressively able to correctly catch the evolved concept, but with the additional use of "spurious", superfluous clauses inherited from the previous definitions. Such clauses actually do not cover any example, since all possible values for a given property (and, noticeably, only values for that property) are avoided by means of negated literals, and hence act as indicators of something 'strange' happening in the concept definition. This is not completely surprising, nor undesirable. Indeed, even if not still constrained by previous examples, a concept is hardly expected to completely and abruptly change in time, and hence a learned theory, supported by previous experience, should not be too easily thrown away or radically changed.

Hence, to check if the proposed technique was sound, a second experiment was carried out, in which the three concepts were generated by "chaining" them so that each definition keeps one of the two descriptors of the previous one, and changes the other. According to such a bias, the exact number of examples that do not change their class label across two consecutive hypotheses was computed, and found equal to 4 for positive examples and 218 for negative ones (which maps onto $\alpha = .25$ and $\beta = .95$, respectively). It is interesting to note that, in this case, the evolution of the induced theory (reported in Figure 4) was much cleaner (i.e., the theory does not carry on in successive stages "spurious" clauses inherited from previous states of the drifting concept). Moreover, each definition quickly (always after processing just the first subset, which is why no performance graph is provided in this case) evolved towards the correct one (i.e., the induced definitions exactly correspond to the true — underlying — ones). This completely confirmed and validated our expectations, and specifically the fact that if the concept goes through a normal and acceptable (i.e., neither abrupt nor drastic) evolution, and if such evolution were mathematically characterizable, then the proposed technique is perfectly able to follow the concept evolution and to correctly and plainly assimilate it in the learned definitions.

Lastly, another experiment, although less controlled, was carried out, with the aim of testing the proposed technique on a more real-world domain, in which the concepts are assumed not to completely and suddenly change along the time, and in addition the exact amount of evolution is not known and can only be estimated. Specifically, the considered problem concerned the evaluation of tests in University courses. The concepts to be learned are *Passed* (corresponding



Stage 1	Stage 2	Stage 3
target(A) :- size_small(A), color_red(A).	target(A) :- size_small(A), color_red(A), not(shape_rect(A)), not(shape_oval(A)), not(shape_circular(A)), not(shape_square(A)). target(A):- color_green(A), shape_circular(A).	target(A) :- size_small(A), color_red(A), not(shape_rect(A)), not(shape_oval(A)), not(shape_circular(A)), not(shape_square(A)). target(A):- color_green(A), shape_circular(A), not(size_medium(A)), not(size_large(A)), not(size_small(A)), not(size_very_large(A)). target(A) :- size_large(A), color_blue(A).

Figure 3: Evolution towards Completely Different Artificial Concepts

to marks 6–10 in a 10-valued scale), *Not passed* (corresponding to marks 0–4) and *Borderline* (corresponding to mark 5), where it is clear that the evaluation may slightly change across different tests in order to reflect the average quality of the students and the difficulty of the test itself. An artificial dataset was built, by randomly creating 40 hypothetical examination paper descriptions for each simulated test. It is worth pointing out that just the descriptions were generated, without any intended model and without labeling them with any mark. Parameters for describing each paper were the following:

- Number of exercises done (0–5);
- Amount of serious errors (0–5);
- Amount of other errors (0–5);
- Amount of time spent (3 values);

Stage 1	Stage 2	Stage 3
target(A) :- size_small(A), color_red(A).	target(A) :- color_red(A), shape_circular(A).	target(A) :- color_red(A), size_large(A).

Figure 4: Progressive Evolution of Artificial Concept

	Passed	Borderline	Non-passed
I test	15	4	21
II test	19	3	18
III test	10	7	23

Table 1: Mark Distribution across 3 University tests

- Presence of comments (Yes/No);
- Presence of references (Yes/No);
- Level of detail (3 values).

Three tests were generated, and then each of the corresponding observations was evaluated by the Professor according to the above parameters in order to assign it one of the marks according to the previously exposed guidelines. Thus, this experiment can well be considered as a real-world one, except that the examination paper descriptions were randomly generated rather than representing actual tests carried out at the University. Table 1 reports the distribution of examples in the tests. It is easy to note that the second test resulted easier to be solved by the students than the first one (because the number of passed is greater than in the first test), while the third turned out to be the most difficult of all (because it has the largest number of non-passed). Accordingly, it could be expected that the requirements for passing the exam are tighter in the second test, while the passing threshold should be lower in the others, and particularly in the third one. Another useful remark is that, in some cases, identical descriptions in different tests received a different mark, which further stresses the drift-handling algorithm. The results for different parameter settings are summarized in Table 2. The values for α and β are close to each other, and are both very high in order to obtain a conservative behavior (according to the intuition that a professor's marks, although sufficiently flexible to take into account the peculiarities of the different tests, should anyway follow a general guideline that does not change very much from a test to another). This probably explains why the total number of clauses in the resulting theories does not change, while, on the other hand, as the requirements for preservation of previous inductions and for consideration of previous examples become more demanding, the number of specializations by means of negative literals increases, indicating (according to the system behavior) a greater effort to reach the desired refinements.

Table 3 traces each concept along its evolution, reporting for each concept the number of clauses in the corresponding definition after each test (I, II and III stand for the first, second and third test, respectively). The parameter settings

α	β	new clauses	lgg	pos spec	neg spec
80	80	16	30	12	5
90	90	16	30	10	6

Table 2: Statistics on Learning University Test Marks

	I	II	III
Not passed	5	6	6
Passed	3	5	5
Borderline	3	3	5

Table 3: Evolution of the Number of Clauses in University Marks Theories

are not reported, since Table 2 has shown that it did not affect the addition of new clauses in this experiment. It is possible to note that the number of clauses in the definitions always increases, due to our choice of allowing only to change clauses, dropping none of them, in order to ensure continuity with the past stages of the concept. In particular, there are fewer clauses for concept *borderline*, which can be explained with the intuition that it represents only one mark, while the other two represent 5 marks each: hence, its definition might be simpler because it has to explain a narrower spectrum of cases. The need of additional clauses to express the intended concepts appears earlier in the definitions for *Not passed* and *Passed*, that since the second test switch from 5 to 6 and from 3 to 5 clauses, respectively; on the contrary, simple refinements were sufficient for *Borderline* after the second test, and only after the third one two more clauses were needed.

Lastly, after discussing the global evolution of the learned concepts, let us now look more closely at the evolution of single definition items. This is possible because INTHELEX allows to trace every single clause along its life. Each hypothesis is made up, in addition to definitions that change, also of stable parts, which suggests that the system does not indiscriminately change the concept definitions, but is able to limit evolutions to the proper items. For instance, since the beginning and throughout the three tests, the clauses saying that only 0 or 1 exercises done are not sufficient to pass the exam are always present in the definition of *not passed*. Similarly, a number of other clauses are stable only across two consecutive tests. In particular, *borderline* seemed the best concept to be traced, since it is presumably the most affected when concepts drift (for the same reasons explained above). Specifically, we traced changes in the second clause learned for such a concept with parameters $\alpha = \beta = 80\%$, obtaining the following result. After the first test:

```
borderline(A) ← exercises_done_3(A), serious_errors_5(A),
           other_errors_3(A), comments(A), references(A),
           detail_2(A), time_1(A).
```

After the second test:

```
borderline(A) ← serious_errors_5(A), references(A), detail_2(A),
           no_comments(A).
```

After the third test:

```
borderline(A) ← references(A),exercises_done_2(A),serious_errors_1(A).
```

A more detailed interpretation of such a result is possible by comparing it to our expectations coming from the knowledge of both the evaluation parameters and the change in tests difficulty. After the first test we have a very specific definition, including examination papers with about half exercises done but many errors, which overrides the presence of comments, references and details. Some drift is already present after the second test, where many literals were dropped and, most interestingly, the condition *comments* was changed into its opposite *no_comments* (which would not have been possible under normal conditions because of the consistency preservation requirement). Indeed, we would expect that, because the test is easier, also the presence of comments on exercises development becomes relevant to pass. For the same reasons, being the third test the most difficult overall, the requirements for passing it should be looser. Actually, this happens, in that now just 2 exercises with few serious errors are sufficient for a student to reach the borderline.

6 Conclusions

In this work we have concentrated our effort on the problem of incremental learning, also in the presence of a known concept drift. A technique for handling such situations has been proposed, and an implementation in an incremental learning system has been carried out. Experiments in different domains reveal that such a technique can effectively deal with the induction of concepts whose definition might change over time.

We have considered the Expert responsible for detecting the drift and providing of the new group of examples. As mentioned, other algorithms try to detect the drift from the data and adjust learning windows accordingly. Our algorithm could be improved by registering a time-stamp for each example, so that the function measuring the decay of correctness would be exploited more precisely in the determination of the strength of the constraint by each group.

References

[1] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[2] J.M. Becker. Inductive learning of decision rules with exceptions: Methodology and experimentation. Master's thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1985. B.S. diss., UIUCDCS-F-85-945.

[3] S. Ceri, G. Gottlöb, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Heidelberg, Germany, 1990.

[4] P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[5] F. Esposito, A. Laterza, D. Malerba, and G. Semeraro. Locally finite, proper and complete operators for refining Datalog programs. In Z.W. Raś and M. Michalewicz, editors, *Proceedings of ISMIS96*, volume 1079 of *LNAI*, pages 468–478. Springer, 1996.

[6] F. Esposito, D. Malerba, G. Semeraro, C. Brunk, and M. Pazzani. Traps and pitfalls when learning logical definitions from relations. In Z. W. Raś and M. Zemankova, editors, *Methodologies for Intelligent Systems*, number 869 in LNAI, pages 376–385. Springer-Verlag, 1994.

[7] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy Theory Revision: Induction and abduction in INTHELEX. *Machine Learning Journal*, 38(1/2):133–156, 2000.

[8] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 323–333, New York, 1998. Morgan Kaufmann.

[9] M.B. Harries and K. Horn. Detecting context drift in financial time series prediction using symbolic machine learning. In *Proceedings of the Eight Australian Joint Conference on Artificial Intelligence*, pages 91–98. World Scientific, Singapore, 1995.

[10] M.B. Harries, C. Sammut, and K. Horn. Extracting hidden contexts. *Machine Learning*, 32(2):101–126, 1998.

[11] D.P. Helmbold and P.M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(2):27–45, 1994.

[12] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[13] J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19:503–581, 1994.

[14] A.C. Kakas and P. Mancarella. On the relation of truth maintenance and abduction. In *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence - PRICAI90*, Nagoya, Japan, 1990.

[15] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann, Stanford, CA, 2000.

[16] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1997.

[17] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[18] R.S. Michalski. Inferential theory of learning. developing foundations for multistrategy learning. In R.S. Michalski and G. Tecuci, editors, *Machine Learning. A Multistrategy Approach*, volume IV, pages 3–61. Morgan Kaufmann, San Mateo, CA, 1994.

[19] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experiance with a learning personal assistant. *Communications of the ACM*, 37:81–91, 1994.

[20] G. D. Plotkin. Building-in equational theories. In B. Meltzer and Donald Michie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, 1972.

[21] G.D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[22] R. Reiter. Equality and domain closure in first order databases. *Journal of the ACM*, 27:235–249, 1980.

[23] C. Salganicoff. Densitive adaptive learning and forgetting. In *Proceedings of the Tenth Internatinal Conference on Machine Learning*, pages 276–283. Morgan Kaufmann, San Mateo, CA, 1993.

[24] J.C. Schlimmer and R.H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(2):317–354, 1986.

[25] G. Semeraro, F. Esposito, and D. Malerba. Ideal refinement of datalog programs. In M. Proietti, editor, *Logic Program Synthesis and Transformation*, volume 1048 of *LNCS*, pages 120–136. Springer, 1996.

[26] G. Semeraro, F. Esposito, D. Malerba, C. Brunk, and M. Pazzani. Avoiding non-termination when learning logic programs: A case study with foil and focl. In L. Fribourg and F. Turini, editors, *Logic Program Synthesis and Transformation - Meta-Programming in Logic*, number 883 in LNCS, pages 183–198. Springer-Verlag, 1994.

[27] G. Semeraro, F. Esposito, D. Malerba, N. Fanizzi, and S. Ferilli. A logic framework for the incremental inductive synthesis of datalog theories. In N. E. Fuchs, editor, *Logic Program Synthesis and Transformation*, number 1463 in LNCS, pages 300–321. Springer-Verlag, 1998.

[28] Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

[29] C. Taylor and G. Nakhaeizadeh. Learning in dynamically changing domains: Theory revision and context dependency issues. In M. van Someren and G. Widmer, editors, *Proceedings of the 9th European Conference on Machine Learning*, pages 353–360. Springer, 1997.

[30] P. R. J. van der Laag. *An Analysis of Refinement Operators in Inductive Logic Programming*. PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

[31] P. R. J. van der Laag and S.-H. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In F. Bergadano and L. de Raedt, editors, *Machine Learning*, number 784 in LNAI, pages 307–322, Berlin, 1994. Springer-Verlag.

[32] P. R. J. van der Laag and S.-H. Nienhuys-Cheng. A note on ideal refinement operators in inductive logic programming. In S. Wrobel, editor, *Proceedings of the International Workshop on Inductive Logic Programming*, pages 247–260, 1994. GMD-Studien Nr. 237.

[33] G. Widmer. Combining robustness and flexibility in learning drifting concepts. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 368–372. Wiley, Chicester, UK, 1994.

[34] G. Widmer. Recognition and exploitation of contextual clues via incremental meta-learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1996.

[35] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[36] G. Widmer and M. Kubat. Special issue on context sensitivity and concept drift: Guest editors' introduction. *Machine Learning*, 32(2):83–84, 1998.

[37] S. Wrobel. *Concept Formation and Knowledge Revision*. Kluwer Academic Publishers, 1994.

[38] S. Wrobel. First order theory refinement. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 14–33. IOS Press, Amsterdam, 1996.

[39] J.-D. Zucker. Semantic abstraction for concept representation and learning. Desenzano del Garda, Italy, 1998.