Learning Specifications of Interaction Protocols and Business Processes and Proving their Properties

Apprendimento di specifiche di protocolli di interazione e processi di business e verifica delle loro proprietà

Marco Alberti, Marco Gavanelli, Evelina Lamma, Fabrizio Riguzzi, and Sergio Storari
**SOMMARIO/ABSTRACT**

Questo articolo descrive le nostre recenti attività di ricerca per apprendere (con tecniche di Programmazione Logica Induttiva) specifiche modellate in programmazione logica e per verificare (attraverso una procedura di dimostrazione abduttiva) le proprietà di sistemi così specificati. I sistemi realizzati qui descritti sono stati applicati rispettivamente per l’apprendimento e la verifica di proprietà di protocolli di interazione in sistemi multi-agente, servizi Web, protocoll di screening e processi di business.

In this paper, we overview our recent research activity concerning the induction of Logic Programming specifications, and the proof of their properties via Abductive Logic Programming. Both the inductive and abductive tools here briefly described have been applied to respectively learn and verify (properties of) interaction protocols in multi-agent systems, Web service choreographies, careflows and business processes.

**Keywords:** Computational logic, Induction, Abduction, Interaction protocols, Careflows, Business processes.

1 **Introduction**

Thanks to its declarative semantics and its underlying proof theory, Logic Programming, and Computational Logic (CL, for short) in a broader sense, have been proved high-level formal languages for specification and verification. The adoption of logic for computer programming was promoted and improved in the late seventies also in Italy by a clever community. Logic Programming is grounded on a purely declarative representation language, and a theorem-prover or model-generator (like in Answer Set Programming) as the problem-solver. The main task of the problem-solver is the verification that an (existential) query holds in the given specification. Variants of the problem-solver can also be exploited to enrich the representation language and empower the reasoning with new features, such as hypothetical and non-monotonic reasoning, or to prove properties arising from the specification itself. Induction techniques can also be applied, to learn (general and formal) specification from logs and extensional databases or to further abstract specifications.

In this paper, we describe the recent activity carried out at ENDIF, University of Ferrara concerning the induction of CL-based specifications, and the proof of their properties. To this purpose, in the former activity we exploit Inductive Logic Programming techniques (ILP for short), and the DPML algorithm [13] in particular. This algorithm learns a specification expressed in a CL-based language from labeled traces (a database of events recording happened interactions or activities). The target language, named \( \text{SCIFF} \), was originally defined for the specification of interaction protocols in the context of the UE IST-2001-32530 Project (named SOCS), and has been later adopted to specify web service choreographies [1], careflows [12] and business processes [5]. A system is specified in the \( \text{SCIFF} \) language by a knowledge base (a logic program) and a set of \( \text{SCIFF} \) forward rules, called integrity constraints. Each integrity constraint relates occurring events (in the body) with an expected behaviour (typically in the head) in terms of expectations about events. Expectations can be positive (for mandatory events) or negative (forbidden events). Given a \( \text{SCIFF} \) specification, the compliance of the system to the specifications can be checked on-the-fly through the \( \text{SCIFF} \) proof-procedure [3], that abduces the expected behaviour and verifies its matching with the actual one.

The adoption of a CL-based language in specifying a system paved also the way to follow a proof-theoretic approach for proving or disproving properties of the given \( \text{SCIFF} \) specification. To this purpose, we exploit abduction, and in particular an extension of the \( \text{SCIFF} \) proof-procedure called \( g\text{-SCIFF} \) [2]. \( g\text{-SCIFF} \) is an abductive proof-procedure which, starting from a goal, verifies, in a generative manner by abduction, whether there exists
a scenario (i.e., a set of generated events) supporting the goal, consistent with the given integrity constraints, and not self-contradictory (e.g., an event does not unify with any forbidden one). In this case, this scenario represents a witness for the goal, and also corresponds to extensions identified by the declarative semantics.

The paper is organized as follows. In Section 2 we briefly introduce the SCIFF language. In Section 3, we show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

The CIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

The CIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

The CIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

The CIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, where they focus on interactive conformance checking. SCIFF combines occurred events and to generate corresponding expectations. To show how learning from interpretations can be exploited to learn a SCIFF theory, and also discuss some experimental results. In Section 4, we present g-SCIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in Section 5.
new generality relation, we can obtain a generalization $D$ of an IC $C$ by adding a literal to the body, adding a disjunct to the head, removing a literal from a disjunct in the head or adding a literal to a disjunct in the head. This generalization operator is used by DPML to search the space of ICs from specific to general.

The literals to be added are defined by the language bias, an intensional definition of the search space. In DPML the language bias is a set of assertions in the form of pairs $(BS, HS)$, where $BS$ is a set that contains the literals that can be added to the body and $HS$ is a set that contains the disjuncts that can be added to the head.

Inducing SICIF theories is also interesting because it has been shown [6] that other declarative process languages such as DecSerFlow [17] or ConDec [16] can be mapped to SICIF. Therefore, if we can ensure that the form of the learned ICs corresponds to one of the constraints of these languages, we could learn such constraints by first learning ICs and then translating them into DecSerFlow or ConDec. By providing DPML with a language bias that suitably restricts the search space of ICs, DPML returns a theory with ICs in the desired form, that can be automatically translated into one of the above declarative process languages (see also [12]).

We implemented the whole process of induction plus translation in the DecMiner [12] plug-in of ProM. DecMiner assists the user in all the phases of the learning process, from the definition of the language bias, to the labeling of traces, to the translation of the mined ICs into ConDec constraints.

In particular, the language bias is automatically generated by DecMiner considering a subset of activities $A$ and a subset of ConDec constraints $T$ chosen by the user.

DPML and DecMiner have been tested on artificial and real datasets. The artificial datasets were randomly generated from two process models, namely the NetBill protocol [8] and an electronic auction protocol [4]. The real dataset regards the healthcare process of cervical cancer screening in the Emilia-Romagna Italian region. DPML and DecMiner results were compared with those of the $\alpha$-algorithm and MPM. The sets of ICs returned by DPML/DecMiner can also be used to check (intensional) properties. This can be done by exploiting the g-SICIF proof-procedure described in the following.

### 4 Proving properties by g-SICIF

The SICIF proof-procedure addresses the important software engineering task of checking compliance during runtime (or a-posteriori using an event log), i.e., whether the agents behave in a compliant manner with respect to a given interaction protocol or specification. However, this does not exhaust the possible uses of abductive reasoning: the event literals composing the history can be assumed as well, in order to foresee all the possible evolutions of the system under test. Knowing the specification (in terms of an abductive program), one could (in principle) generate all the histories that the system can support and then study them for common patterns or to formally prove properties of the system.

Of course, explicitly generating all the histories is not feasible, since the number of histories compliant to a protocol are typically infinite for protocols of practical use. However, we can generate compliant histories in intensional way, and then reason upon them: the hypothetical events can contain variables, possibly subject to CLP constraints. In order to generate compliant histories, SICIF has been improved and extended to a generative version, called g-SICIF. g-SICIF considers $H$ literals as abducibles, and contains a new transition, called fulfillment, that fulfills the expectations by abducing matching events:

$$E(X, T) \rightarrow H(X, T).$$

g-SICIF is provably sound: all generated histories fulfill the given specifications.

In the literature, properties are often classified as safety or liveness properties. A safety property is a universal property: intuitively, it ensures that nothing bad will ever
happen (whenever the protocol/specification is respected). A liveness property is, instead, existential: it ensures that something good will eventually happen. A liveness property can be passed to g-SCIFF as a goal containing positive expectations: if the g-SCIFF proof-procedure succeeds in proving the goal, the generated history witnesses that there exists a way to obtain the goal while being conformant to the protocol. A safety property $\phi$ can be negated (as in model checking), and then passed to g-SCIFF as a goal $G \equiv \neg \phi$. If the g-SCIFF proof-procedure succeeds in finding a history $HAP$ (i.e., $\models_{HAP} \neg \phi$), we have a counterexample: the history $HAP$ satisfies the protocol and does not enjoy the safety property $\phi$.

The g-SCIFF proof-procedure is implemented in SICS-tus 4, making extensive use of Constraint Handling Rules [10] to implement its transitions. SCIFF and g-SCIFF come in a same package, that can be freely downloaded from the web\(^1\): the g-SCIFF behaviour is activated by simply setting an option.

The g-SCIFF proof-procedure has been applied to the formal verification of various systems and protocols. g-SCIFF was able to derive the flawness of the Needham-Schroeder security protocol [2], and the good atomicity property of the NetBill protocol [2]. It is also a basic component of the A'LoWS framework [1], for the proof of interoperability between Web services.

The g-SCIFF proof-procedure operates top-down in a deductive and abductive manner, by manipulating the specification driven by the goal, as usual in Logic Programming, and also generating expectations as SCIFF does and, by fulfillment an (intensional) set of events needed to support the goal. This way, g-SCIFF can be used to prove properties of any SCIFF protocol. For example, one may wonder if the protocol allows a massage service not to be followed by a shiatzu package offer. By expressing this combination as a g-SCIFF query, the user can ask g-SCIFF to generate an intensional history that satisfies the query while fulfilling the protocol. In fact, g-SCIFF generates such a history, with the constraint that the massage type must not be shiatzu.

5 Conclusions

We have presented the CL-based language SCIFF for the specifications of complex systems with interacting entities, such as multi-agent systems, business processes or web services. Moreover, we have discussed how techniques from Inductive Logic Programming were applied for inducing SCIFF theories which can be then translated into graphical languages. Finally, the abductive g-SCIFF proof procedure can be used for proving properties of specifications, either learned or provided by the user.

Acknowledgements

We are in debt with the Artificial Intelligence group at DEIS, University of Bologna who shared with us most of the activity here reported.

REFERENCES


\^1http://lia.deis.unibo.it/sciff/
Contacts

Marco Alberti, marco.alberti@unife.it
Marco Gavanelli, marco.gavanelli@unife.it
Evelina Lamma, evelina.lamma@unife.it
Fabrizio Riguzzi, fabrizio.riguzzi@unife.it
Sergio Storari, sergio.storari@unife.it

tutti affiliati al

Dipartimento di Ingegneria,
Università di Ferrara
Via Saragat, 1
44100 Ferrara