TIL and Logic Programming

Martina Číhalová, Nikola Ciprich, Marie Duží, Marek Menšík

VŠB-Technical University Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic
m.tina.cihal@gmail.com, nikola.ciprich@linuxbox.cz,
marie.duzi@vsb.cz, mensikm@gmail.com

Abstract. The paper introduces a method of transition from TIL into Prolog system and vice versa, in order to utilize Prolog inference machine in the deductive system of TIL. We specify a subset of the set of TIL constructions the elements of which can be encoded in Prolog language, and introduce the method of translation from TIL into Prolog. Since Prolog is less expressive than TIL, we have to build up a TIL functional overlay that makes it possible to realize the reverse transition from Prolog into TIL in a near to equivalent way.

Key words: TIL, $\mathcal{T}IL$-`Script` language, inference machine, Prolog

1 Introduction

Transparent Intensional Logic (TIL) is a highly expressive logical system apt for the logical analysis of natural language.[1] In our project 'Logic and Artificial Intelligence for Multi-agent Systems' we deal with the problem of agents' communication with each other as well as with their environment. Human-computer communication should be smooth and near to isomorphic to natural-language communication. For this reason we voted for the language of TIL *constructions* as a specification and communication language of a multi-agent system. We develop the $\mathcal{T}IL$-`Script` language, a computational variant of TIL, which serves as a content language of agents' messaging.[2] $\mathcal{T}IL$-`Script` is a TIL dialect using only ASCII characters and slightly adjusted semantics. On the other hand, a great expressive power is inversely proportional to an easy implementation of a suitable automatic deductive system. Since TIL is a logic based on the infinite ramified hierarchy of types, it is impossible to create a complete logical calculus and make use of a standard automatic theorem prover. Though TIL deductive system has been theoretically specified, its implementation is still a work in progress. For this reason we decided to specify a subclass of TIL and to utilize a first-order inference machine such as Prolog. This is a feasible way due to TIL being a fully compositional system and from this point of view extensional calculus.

[1] See, for instance, [7] and [8]. [2] For details on the project, see [4]. $\mathcal{T}IL$-`Script` and agents' messaging is FIPA compliant. For FIPA standards see [6]. More on $\mathcal{T}IL$-`Script` , see [2] or [3].

Petr Sojka, Aleš Horák (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2008, pp. 17–30, 2008.

The goal of this paper is to specify a subset of TIL constructions which can be encoded in Prolog, together with the algorithm of their translation. The transition from TIL into Prolog and *vice versa* has to be specified in a near to equivalent way, i.e., without the loss of important information encoded by TIL constructions. Thus prior to the translation proper, TIL functional overlay of the Prolog machine has to be specified. In particular, we have to take into account that standard Prolog does not work with modal and temporal parameters, and with the types of denoted entities. Moreover, $\mathcal{TIL}$-`Script` is a functional language whereas Prolog a relational one, which is also a problem that has to be dealt with prior to translation proper. In the paper we propose a general strategy of adjusting TIL constructions into the form that can be processed by Prolog.

The paper is organized as follows. Section 2 briefly describes basic principles of programming in logic using Prolog. The main Section 3 is a description of the transition from TIL into Prolog. We first describe the translation of simple sentences into Prolog facts, rules and goals. Then we specify the method of translating wh-questions and yes-no questions. Finally, an algorithm of the specification of TIL subset of constructions transferable into Prolog is presented. Concluding Section 4 is an outline of future research.

2 A Brief Description of the Prolog System

Prolog can be defined as a couple (K, I), where K is the program base consisting of facts, rules and queries encoded in a first-order language, and I is an inference machine that makes it possible to derive consequences of K.[3] The inference machine is based on the general resolution algorithm for the first-order predicate logic (FOL). An input for the resolution algorithm is a set of formulas in the Skolem clausal form. Since FOL is only partly decidable,[4] the clauses of a Prolog language are limited to the decidable subset of FOL, namely the set of *Horn clauses*. To put this description on a more solid ground, we define:

Literal is an atomic formula or its negation. For instance, $p(x, y)$, $\neg q(f(x))$ are literals. *Clause* is a disjunstion of (positive and/or negative) literals:

$$C = l_1 \vee l_2 \vee \ldots \vee l_n \vee \neg m_1 \vee \ldots \vee \neg m_k\,.$$

Skolem clausal from of a formula is a conjunctive normal form without existential quantifiers

$$SCF = \forall x_1 \ldots \forall x_m (C_1 \wedge \ldots \wedge C_n),$$

where $C_1, \ldots, C_n$ are clauses.

[3] For details on Prolog see, e.g. [1]. [4] As a consequence of Gödel's incompleteness theorem, there is no proof calculus deciding FOL. Church proved that there are calculi that partly decide FOL; this means that if a formula F is logically valid then it is provable, whereas if F is not logically valid then it can be the case that there is no finite proof of F in the calculus.

Each FOL formula F can be transformed into *SCF* in such a way that the transformation is consistency preserving. In other words, if F has a model then *SCF* has a model as well. And if *SCF* is a contradiction then F is a contradiction as well. Thus the proof by general resolution is an indirect proof. The proof method is guarded by the following rules:

General *quantifier elimination*: $\forall x A(x) \vdash A(t/x)$, where the term t is substitutable for the variable x.

General *resolution rule*: Let A_i, B_i, l_1, l_2 be atomic formulas of FOL, σ a collision-less substitution such that $l_1\sigma = l_2\sigma$. Then the following rule is valid

$$\frac{A_1 \vee \ldots \vee A_m \vee l_1, B_1 \vee \ldots \vee B_n \vee \neg l_2}{A_1\sigma \vee \ldots \vee A_m\sigma \vee B_1\sigma \vee \ldots \vee B_n\sigma}$$

An *SCF* formula is not satisfiable if and only if the empty clause (#) is derivable from *SCF* by step-by-step application of the general resolution rule.

Method of Programming in Logic (Prolog) is a special case of the general resolution method with three major restrictions. First, it works only with *Horn clauses*, i.e. clauses with at most one positive literal:

$$HC = l \vee \neg m_1 \vee \ldots \vee \neg m_k.$$

Second, Prolog mostly applies a *goal driven*, *depth-first* strategy with *backtracking* for generating resolvents. Though this strategy is not complete, it is more effective than other strategies evaluating the computational tree completely. Third, *negation* is dealt with as a failure to prove (the Close World Assumption).

Notation in Prolog.
Rule (Conditional statement): P:- Q_1,...,Q_n.

Atomic formula $P = p(t_1,\ldots,t_m)$ is the *head* of the rule, t_i are its formal parameters; atomic formulas Q_1,...,Q_n is the *body* (*tail*) of the rule with the *subgoals* Q_i.

Note that conditional statement is equivalent to

$$(Q_1 \wedge \ldots \wedge Q_n) \supset P \text{ or i}(\neg Q_1 \vee \ldots \vee \neg Q_n \vee P).$$

Fact: P. A Horn clause without negative literals, i.e., an atomic formula.
Goals: ?- $Q_1, \ldots, Q_n$. A Horn clause without a positive literal: $\neg Q_1 \vee \ldots \vee \neg Q_n$.

Note that this formula is equivalent to $(Q_1 \wedge \ldots \wedge Q_n) \supset$ False.
An empty clause: #. (Contradiction, success).

Logic program is a sequence of rules and facts. Prolog inference machine derives answers to the questions (goals) by applying respective substitutions in order to instantiate variables in goals and generate resolvents. Derivation of an empty clause is a success; the respective substitution is then an answer to the goal.

3 Transition from TIL into Prolog

When encoding TIL constructions in Prolog, we focus on the constructions that construct empirical entities of type $o_{\tau\omega}$, $(o\alpha)_{\tau\omega}$ and $(o\alpha_1 \ldots \alpha_n)_{\tau\omega}$, i.e., propositions, properties and relations-in-intension, respectively, or mathematical facts of type o. Constructions of propositions are translated as Prolog facts/rules or *Yes–No questions* according whether a respective message is of a kind 'Inform' or 'Query'. Constructions of properties and relations are translated as *Wh-questions*.[5]

When creating a Prolog base of facts and rules we have to record the types of entities that receive mention in TIL constructions. In particular, types of intensions have to be distinguished from types of extensions. To this end we use a special Prolog predicate 'type'. Prolog base of facts and rules then contains Prolog translation of TIL constructions $v-$constructing True (or False in case of an incorrect data collection), i.e., values of the propositions in a given world w at a time t of data collection. When querying on this base, we use a construction of an intension and ask for its value in a state of affairs as recorded in or entailed by the Prolog base.

3.1 Building up a Prolog base of facts and rules

Prolog base of facts and rules is created by transferring indicative messages of the kind 'Inform', the content of which is encoded in the $\mathcal{TIL}$-`Script` language.

For instance, the proposition that Charles is a professor constructed by the Closure

$$\lambda w \lambda t [{}^0\mathit{Professor}_{wt}\ {}^0\mathit{Charles}]$$

and encoded in $\mathcal{TIL}$-`Script` by[6]

```
[\w:World \t:Time [['Professor@wt 'Charles]]].
```

is translated into a Prolog fact

```
prof(charles).                                    (1)
```

By means of the Prolog predicate 'type' we remember TIL type of *Professor*, i.e., the type $(o\iota)_\tau\omega$ of an individual property, in order to reconstruct the original proposition in the reverse translation into $\mathcal{TIL}$-`Script` .

The proposition that all professors are employees constructed by the Closure

$$\lambda w \lambda t\ \forall x [[{}^0\mathit{Professor}_{wt}\ x] \supset [{}^0\mathit{Employee}_{wt}\ x]]$$

is translated into the Prolog rule (accompanied by the respective type predicates)

```
empl(X):- prof(X).                                (2)
```

[5] For details on the analysis of questions and answers in TIL see, e.g., [5]. [6] For details on the $\mathcal{TIL}$-`Script` language see, e.g., [1] or [2].

Similarly the proposition that cars and bicycles are mobile agents constructed by

$$\lambda w \lambda t\ \forall x[[[^0Car_{wt}\ x] \vee [^0Bike_{wt}\ x]] \supset [[^0Mobile\ ^0Agent]_{wt}\ x]]$$

is translated into the Prolog rule (accompanied by the respective type predicates)

```
mobile_agent(X):- car(X); bike(X).
```
(3)

So far so good. However, TIL is based on the functional approach whereas Prolog is a relational language. Thus the translation of constructions containing constituents $v-$constructing functions of types $(\alpha\beta)$ and attributes (empirical functions) of types $(\alpha\beta)_{\tau\omega}$, where $\alpha \neq o$, is not so straightforward. We have to first transform them into constructions of relations of type $(o\alpha\beta)_{\tau\omega}$ by introducing an auxiliary variable.

For instance, the fact that Charles' only car is a mobile agent is constructed by

$$\lambda w \lambda t[[^0Mobile\ ^0Agent]_{wt}\ [^0The[^0Car\_of_{wt}\ ^0Charles]]]$$

Types. $Mobile/((o\iota)_{\tau\omega}(o\iota)_{\tau\omega})$ – modifier; $Agent/(o\iota)_{\tau\omega}$; $Car\_of((o\iota)\iota)_{\tau\omega}$; $The/(\iota(o\iota))$ – the singulariser ('the only ...').
The respective code in the $\mathcal{T}IL$-`Script` language is this:

```
[\w:World [\t:Time [
        ['Mobile 'Agent]@wt ['Sing ['Car_of@wt 'Charles]]
]]].
```

We have to simplify this construction by ignoring the singulariser, and transform this Closure into the Prolog rule

```
mobile_agent(Y):- car_of(Y,charles).
```

Gloss. For all y, if y is a car of Charles then y is a mobile agent.

In general, the pre-processing of constructions of empirical functions in TIL is driven by this strategy. If $Attr \rightarrow (\alpha\beta)_{\tau\omega}$ is a construction of an empirical function, $P \rightarrow (o\alpha)_{\tau\omega}$ a construction of an α-property and b an entity of type β, then the construction of a proposition of the form

$$\lambda w \lambda t[P_{wt}\ [Attr_{wt}\ ^0b]]$$

is modified into $(x \rightarrow \alpha)$

$$\lambda w \lambda t[\forall x[[x = [Attr_{wt}\ ^0b]] \supset [P_{wt}\ x]]]$$

which is then translated into the Prolog rule

```
p(X) :- attr(X,b).
```

Moreover, TIL functional overlay over the so-created Prolog base of facts and rules must also make it possible to distinguish between analytically necessary facts and rules and empirical ones.

For instance, the above proposition that cars and bicycles are mobile agents could be specified as an analytical one, meaning that *necessarily*, all cars or bicycles are mobile agents. The respective TIL analysis is then this Closure

$$\forall w \forall t \forall x [[[^0Car_{wt}\, x] \vee [^0Bike_{wt}\, x]] \supset [[^0Mobile\ ^0Agent]_{wt}\, x]].$$

However, the resulting Prolog rule will be the same as the above rule (3). In such a case we must remember in the TIL ontology that the property of being a mobile agent is a *requisite* of the properties of being a car and being a bicycle.

Another problem we meet when translating TIL constructions into Prolog is the problem of *equivalences.* For instance, inserting a formula of the form $(p \equiv q) \Leftrightarrow (p \supset q) \wedge (q \supset p)$ into the Prolog base might cause the inference machine to be kept in an infinite loop. Yet we need to record such equivalences in TIL ontology. This is in particular the case of a meaning refined by *definition.*

Imagine, for instance, the situation of an agent a who does not have in its ontology the concept of a car park with vacancies. Then a may learn by asking the other agents that "A car park with vacancies is a car park some of whose parking spaces nobody has occupied yet". The content of a 'query' message asking for the definition of 'car park with vacancies' is [^{0}*Unrecognized* ^{0}[^{0}*Vac* ^{0}*Car_Park*]]. The reply message content is

[^{0}*Refine* ^{0}[^{0}*Vac* ^{0}*Car_Park*]
$^0[\lambda w \lambda t \lambda x [[^0Car\_Park_{wt}\, x] \wedge \exists y [[^0Space\_of_{wt}\, y\, x] \wedge \neg [^0Occupied_{wt}\, y]]]]]$.

Thus the constructions [^{0}*Vac* ^{0}*Car_Park*] and

$$[\lambda w \lambda t \lambda x [[^0Car\_Park_{wt} x] \wedge \exists y [[^0Space\_of_{wt}\, y x] \wedge \neg [^0Occupied_{wt}\, y]]]]$$

are *ex definitione* equivalent by constructing one and the same property. This fact can be encoded in Prolog only by a general rule:

```
vac_park(X,Y) :- park(X), space_of(Y,X), non_occupied(Y).
```

Gloss. (For all X, Y), X has a parking vacancy Y if X is a car park and Y is a space of X and Y is not occupied.

3.2 Querying in Prolog

As stated above, messages of the kind 'Query' with the semantic content constructing propositions represent *Yes–No questions* on the truth-value of the respective proposition in a given state-of-affairs. The semantic core of a Yes–No question (the content of the respective message) is thus the same as that of a corresponding indicative sentence; a construction of a proposition.

For instance, the above example of the proposition that "Charles is a professor" is encoded by a message with the same content as the corresponding query "Is Charles a professor?". The two messages differ only in their kind. The former is of the kind 'Inform', the latter of the kind 'Query'.

However, the Prolog code will differ. We now *ask* whether this proposition is entailed by the recorded Prolog knowledge base. To this end we must translate

the message as a *negative* Prolog fact so that Prolog inference machine can check whether the negated proposition contradicts the base:

```
?- prof(charles).
```

In this case Prolog answers `Yes` providing the fact (1) is contained in its base.

We can also put Yes–No questions on the existence of individuals with such and such properties. In such a case Prolog would answer not only simple `Yes`, but also *which* individuals (according to its knowledge) are such and such.

For instance, the query "Are some employees professors?" is analysed by TIL Closure ($Employee$, $Professor/(o\iota)_{\tau\omega}$; $x \rightarrow \iota$)

$$\lambda w \lambda t [\exists x [[^0 Employee_{wt}\, x] \wedge [^0 Professor_{wt}\, x]]].$$

The $\mathcal{TIL}$-`Script` encoding is then

```
[\w:World [\t:Time [Exists x:Indiv [
        'And ['Empl@wt x] ['Prof@wt x]
]]]].
```

And the respective Prolog code is this

```
?- empl(X), prof(X).
```

Now Prolog tries to succeed in deriving these two goals. To this end Prolog searches its knowledge base for facts and rules containing the `empl` or `prof` in their head, and if possible Prolog unifies the variable `X` with the respective arguments of the found rules/facts. In our case `empl(X)` first matches with (2) and then `prof(X)` with (1) resulting in the success by substituting `charles` for `X`. Thus Prolog answer is `Yes, X=charles`.

When raising '*wh-questions*' we ask for the value of an α-property or an α-relation-in-intension in a given world w at time t. As explained in [5], the analysis is driven by a possible answer. If an answer is a set of individuals, we construct a property. In general, if an answer is a set of tuples of $\alpha_1, \ldots, \alpha_m$-objects, we construct a relation-in-intension of type $(o\alpha_1 \ldots \alpha_m)_{\tau\omega}$.

For instance, the answer to the question "Which professors are older than 60 years?" can be {Materna, Duží}, i.e., the set of individuals. Thus we have to construct the property of individuals:

$$\lambda w \lambda t \lambda x [[^0 Professor_{wt}\, x] \wedge [^0 > [^0 Age_{wt}\, x]\, {}^0 60]],$$

where Age is an attribute of type $(\tau\iota)_{\tau\omega}$. Again, the respective $\mathcal{TIL}$-`Script` code is

```
[\w:World [\t:Time [ \x:Indiv [
        'And ['Prof@wt x] ['> ['Age@wt x] '60]
]]]].
```

In order to translate this question into Prolog, we have to convert the attribute *Age* into a binary relation using an auxiliary variable in a similar way as described above. The resulting construction is

$$\lambda w \lambda t \lambda x [\exists y [[{}^0\mathit{Professor}_{wt}\, x] \wedge [[y = [{}^0\mathit{Age}_{wt}\, x]] \wedge [{}^0> \; y \; {}^0 60]]]].$$

This Closure is transformed into three Prolog goals as follows:

```
?- prof(X), age(Y,X), Y>60.
```
[7]

Now there is another problem generated by Prolog relational approach. Whereas a functional program would return the value of the above property recorded in a given knowledge base state, i.e., a *set* of individuals, Prolog answer in case of the success in meeting these goals will be, e.g., `Yes, X=Duzi, Y=60`. If we want another individual instantiating the property, we have to keep entering semicolon, '`;`', until obtaining the answer `No`.

In order to overcome this difficulty, we may use Prolog '*findall*/3' predicate which has three arguments: (*what, from where, to where*). However, in such a case we have to specify *one* goal (*from where*). Here is how. We create a new unique name of the property the instances of which are asked for. For instance, the above property can be coined 'Aged_professor', and the equality definition "Aged professors are professors older than 60 years" inserted as follows:

$$\lambda w \lambda t \lambda x [{}^0\mathit{Aged\_professor}_{wt} x] = \lambda w \lambda t \lambda x [[{}^0\mathit{Professor}_{wt} x] \wedge [{}^0> [{}^0\mathit{Age}_{wt}\, x] {}^0 60]].$$

However, as explained above, in Prolog we cannot deal with equivalences, because equivalence would cause Prolog attempting to succeed in meeting a goal *ad infinitum.* Thus we have to insert a new *rule* the head of which is the *definiendum* and tail is the *definiens* of the inserted definition. The resulting Prolog code is

```
?-assert(aged_prof(X):-prof(X), age(Y,X), Y>60.),
    findall(X,aged_prof(X),Result),
    retract(aged_prof(X):-prof(X), age(Y,X), Y>60.).
```
[8]

The predicate `assert` serves for inserting the auxiliary rule into Prolog base, and `retract` for erasing it.

The process of generating a generic property corresponding to a 'wh--question' is this:

1) Transform the TIL compound construction of a property into an equivalent construction by inserting a new unique primitive concept of the property. If $C^1, \ldots, C^m$ are the respective constituents of the compound Closure, then the schema of TIL pre-processing is as follows:
   $\lambda w \lambda t \lambda x [{}^0\mathit{C\_new}_{wt}\, x] = \lambda w \lambda t \lambda x [[C^1_{wt} x] \wedge \ldots \wedge [C^m_{wt}\, x]]$.
2) Translate into Prolog: `c_new(X):-c1(X),...,ck(X).`

[7] Due to Prolog depth-first, left-to-right evaluation strategy the order of the goals is important here. The variable `Y` has to be instantiated first with the value of `age_of X`, and then compared with `60`.

[8] The method of processing the attribute *Age_of* is similar as above and it is described in Section 3.3.1.

3) If the rule 'cnew' is contained in Prolog base, then
`findall(X, c_new(X), Result).`
Otherwise, i.e., if 'cnew' is not contained in Prolog base, then
```
?-not(c_new(X)), assert(c_new(X):-c1(X), ..., ck(X)),
    findall(X, c_new(X), Result),
    retract(c_new(X):-c1(X), ..., ck(X)).
```

3.3 Schema of the transition from TIL into Prolog

It should be clear by now that prior to the translation of TIL constructions into Prolog, a lot of pre-processing has to be done in TIL. These modifications vary according to the structure of constructions and the type of entities constructed by primitive concepts, i.e., Trivialisations of non-constructive entities. These primitive concepts fully determine a conceptual system within which we work, and they have to be contained in the respective ontology together with their types. The choice of a conceptual system depends, of course, on the domain area and also on application goals and problems to be solved.

By way of summary we now present a general schema for TIL pre-processing and translation of particular constructions into Prolog code according to the type of entities that receive mention by constituents of a construction. Recall that types of the entities have to be also remembered by Prolog, which we do not indicate here.

Questions are expressed as constructions of intensions, the extension of which in a particular world w and time t we want to know. In case of wh-question the respective constructed entity is a property or relation. In case of Yes–No question it is a proposition. In case of mathematical questions we consider constructions of mathematical functions the value of which we want to compute. In what follows we use double arrow '$\Rightarrow$' for 'translated or pre-processed into'.

3.3.1 Analysis and translation of simple facts or *wh-queries*. First we present a schema of pre-processing simple constructions that do not contain constituents of truth-value functions and quantifiers. Let constructions *P, Attr, Rel* and *Calc* be constructions of such a simple form.

1. Constitents of *propositions.* Let $P \rightarrow o$, Then $\lambda w \lambda t P \Rightarrow$ `?-p.` or `p.`
   Example. "Is Charles a professor?";
   $\lambda w \lambda t\, [{}^0Prof_{wt}\, {}^0Charles] \Rightarrow$ `?- prof(charles).`
2. Constituents of *α-properties*. Let $P \rightarrow (o\alpha)_{\tau\omega}$; $x \rightarrow \alpha$. Then
   $\lambda w \lambda t \lambda x [P_{wt} x] \Rightarrow$ `?-p(X).`
   Example. "Who are our the associate professors?";
   $\lambda w \lambda t \lambda x [[{}^0Associate\, {}^0Prof]_{wt}\, x] \Rightarrow$ `?-assoc_prof(X).`
3. Constituents of *(αβ)-attributes*. Let $Attr \rightarrow (\alpha\beta)_{\tau\omega}$, $P \rightarrow (o\alpha)_{\tau\omega}$; b/β; $\alpha \neq o$; $x \rightarrow \beta$; $y \rightarrow \alpha$.
   Now we have to distinguish two cases.

a) The constituent is used in an 'Inform' message, the content of which is of the form $\lambda w\lambda t\,[P_{wt}[Attr_{wt}\,b]]$. Then
$\lambda w\lambda t\,[P_{wt}\,[Attr_{wt}\,b]] \Rightarrow \lambda w\lambda t\,[\forall y[[^0 = y\,[Attr_{wt}\,b]] \supset [P_{wt}\,y]]]$
$\Rightarrow$ `p(Y) :- attr(Y,b).`
Example. "Charles' father is a professor".
$\lambda w\lambda t\,[{}^0Professor_{wt}[{}^0Father\_of_{wt}\,{}^0Charles]] \Rightarrow$
$\lambda w\lambda t\,[\forall y[[^0 = y\,[{}^0Father_{wt}\,{}^0Charles]] \supset [{}^0Professor_{wt}\,y]]]$
$\Rightarrow$ `prof(Y) :- father(Y,charles).`

b) The constituent is used in a 'Query message' and it is of the form $\lambda w\lambda t\lambda x[P_{wt}[Attr_{wt}\,x]]$. Then
$\lambda w\lambda t\lambda x[P_{wt}[Attr_{wt}\,x]] \Rightarrow \lambda w\lambda t\lambda x[\exists y[[^0 = y[Attr_{wt}\,x]] \wedge [P_{wt}\,y]]]$
$\Rightarrow$ `?- attr(Y,X), p(Y).`
Example. "Whose father is a professor?".
$\lambda w\lambda t\lambda x[{}^0Professor_{wt}[{}^0Father\_of_{wt}\,x]] \Rightarrow$
$\lambda w\lambda t\lambda x[\exists y[[^0 = y[{}^0Father_{wt}\,x]] \wedge [{}^0Professor_{wt}\,y]]]]$
$\Rightarrow$ `:- father(Y,X), prof(Y).`

4. Constituents of *relations-in-intension*. Let $Rel \rightarrow (o\alpha\beta)_{\tau\omega}$; $x \rightarrow \alpha$; $y \rightarrow \beta$. Then
$\lambda w\lambda t\lambda x\lambda y[Rel_{wt}\,xy] \Rightarrow$ `?-rel(X,Y).`
Example. "Who is affiliated to whom?";
$\lambda w\lambda t\lambda x\lambda y[{}^0Affiliate_{wt}xy] \Rightarrow$ `?-affiliate(X,Y).`
5. Constituents of *mathematical functions*. Let $Calc \rightarrow (\tau\tau)$; $x, y \rightarrow \tau$; m/τ. Then
$[Calc\ {}^0m] \Rightarrow$ `?- Y is calc(m).`
$\lambda x\,[Calc\,x] \Rightarrow \lambda x\lambda y[^0 = y[Calc\,x]] \Rightarrow$ `?-Y is calc(X).`
In general, constituents of $n-$ary mathematical functions are transferred as follows. Let $Calc^n/(\tau\tau\ldots\tau)$. Then
$\lambda x_1\ldots x_n[Calc\,x_1\ldots x_n] \Rightarrow \lambda x_1\ldots x_n\lambda y[^0 = y[Calc\,x_1\ldots x_n]] \Rightarrow$
`?- Y is calc(X1,...,Xn).`
Example. "Which is the value of the square root of 144?";
$[{}^0Square\_root\ {}^0144] \Rightarrow$ `?- Y is square_root(144).`

3.3.2 Analysis and translation of more complex *wh-queries*. Since Prolog queries can be only Horn clauses without a positive literal, i.e., clauses of a form $\neg l_1 \vee \neg l_2 \vee \ldots\ \neg l_m$, their general form is $\neg(l_1 \wedge l_2 \wedge \ldots\ l_m)$, which in Prolog notation is recorded as ?- l_1, l_2, ...,l_m. The clauses l_1,...,l_m are the goals to be met.

Thus a TIL construction to be translated must be a conjunctive construction of a property or relation, or of an existential proposition. Let $C_1, \ldots, C_m$ be simple constructions not containing constituents of truth-value functions and quantifiers. Then the general form of TIL construction translatable into a Prolog *wh*-question is:

$$\lambda w\lambda t\lambda x_1\ldots x_n[C_1 \wedge \ldots \wedge C_m], \text{ or } \lambda w\lambda t[\exists x_1\ldots x_n[C_1 \wedge \ldots \wedge C_m]].$$

Schema of the resulting Prolog wh-query is

`?- c1, ..., ck.`

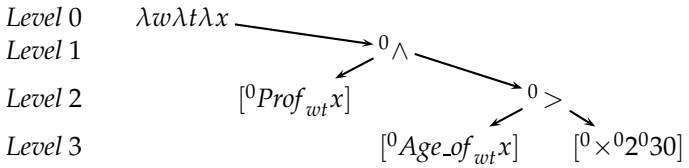
where the `c1, ..., ck` are simple goals created according to the items (1)–(5) of the proceeding paragraph.

According to the complexity of a construction C we first create the structure tree of C, leafs of which are simple constituents of C. The pre-processing and translation of C is then executed bottom up, from leafs to the root of the tree.

Example. "Who are the professors older than 2×30 years?".
The analysis is a construction of the property of being a professor older than 2×30 years:

$$\lambda w \lambda t \lambda x [{}^0\wedge [{}^0Prof_{wt}\, x][{}^0> [{}^0Age\_of_{wt}\, x][{}^0\times\, {}^0 2\, {}^0 30]]].$$

The structure tree is this:



Beginning with the bottommost level 3, we transform leafs containing attributes and mathematical functions in compliance with the above rules (3) and (5), respectively, into constituents $[{}^0 = y\ [{}^0Age\_of_{wt}\, x]]$ and $[{}^0 = z[{}^0\times {}^0 2\ {}^0 30]]$. The auxiliary variables y, z become the arguments of the parent node at level 2: $[{}^0 > y\, z]$. The node $[{}^0Prof_{wt}\, x]$ remains without change, as well as the parent node ${}^0\wedge$ of level 1, because they $v-$construct a truth-value of type o. Finally we have to adjust the root level 0 in compliance with (3) above into $\lambda w \lambda t \lambda x \lambda y \lambda z$ ... The resulting construction is

$$\lambda w \lambda t \lambda x \lambda y \lambda z [{}^0\wedge [{}^0Prof_{wt}\, x][{}^0\wedge [{}^0= y[{}^0Age\_of_{wt}\, x]][{}^0\wedge [{}^0= z[{}^0\times\, {}^0 2\, {}^0 30]][{}^0>\ y\, z]]]].$$

The respective Prolog code is then

```
?- prof(X),age_of(Y,X), Z is ×(2,30), Y>Z.
```

As stated above, if we want *all* the professors older than 2×30 years, we have to insert a new unique name of the property by the equation definition $\lambda w \lambda t \lambda x [{}^0Aged\_professor_{wt} x] =$

$\lambda w \lambda t \lambda x [\exists y \exists z [{}^0\wedge [{}^0Prof_{wt}\, x]$
$[{}^0\wedge [{}^0= y[{}^0Age\_of_{wt}\, x]][{}^0\wedge [{}^0= z[{}^0\times {}^0 2 {}^0 30]][{}^0> yz]]]]].$

The resulting Prolog code is

```
?-assert(aged_prof(X):-prof(X),age_of(Y,X),Z is ×(2,30),Y>Z.),
findall(X,aged_prof(X),Result).
```

3.4 Specification of a TIL subset transferable into Prolog code

Up to now we considered only constructions $v-$constructing entities of a type $(o\beta)$, the structure tree of which is conjunctive. Using a terminology of predicate logic, we considered only constructions which are in a generalised

conjunctive normal form $\lambda w \lambda t\ \lambda x_1 \dots x_m\ [C_1 \wedge \dots \wedge C_m]$, where $C_1, \dots, C_m$ are simple constructions not containing constituents of truth-value functions and quantifiers.

In order to specify the subset of TIL that can be encoded in Prolog, we have to specify restrictions on TIL propositional constructions implied by Prolog, and describe an algorithm of transferring a closed construction $C \rightarrow o_{\tau\omega}$ into the Skolem clausal form (SCF). Here is how.

In what follows we now use '$C(x,y,\dots)$' as a schema of a construction containing free variables $x, y, \dots$

1. *Eliminate w,t.* $[\lambda w \lambda t C(w,t)] \Rightarrow C$
2. *Eliminate unnecessary quantifiers* that do not quantify any variable.
   (For instance, Composition of the form $[[\exists y \forall z P(z)] \wedge [\forall x \exists v Q(v)]]$ is adjusted into $[[\forall z P(z)] \wedge [\exists v Q(v)]]$.)
3. *Apply α-rule* in such a way that different λ-bound variables have different names.
   (For instance, Composition $[[\forall x P(x)] \wedge [\forall x Q(x)]]$ is adjusted into the Composition $[[\forall x P(x)] \wedge [\forall y Q(y)]]$.)
4. *Eliminate connectives* $\supset$ and $\equiv$ by applying the rules $[C \supset D] \vdash [\neg C \vee D]$ and $[C \equiv D] \vdash [[\neg C \vee D] \wedge [\neg D \vee C]]$.
5. *Apply de Morgan laws*:
   $\neg[C \wedge D] \vdash [\neg C \vee \neg D], \neg[C \vee D] \vdash [\neg C \wedge \neg D]$,
   $\neg[\exists x \neg C(x)] \vdash [\forall x C(x)], \neg[\forall x \neg C(x)] \vdash [\exists x C(x)]$.
6. If the resulting construction C contains now existential quantifiers, then *reject* C as non-transferable.
   (For instance, Composition of the form $[[\exists y P(y)] \wedge [\forall x \exists v Q(x,v)]]$ is not transferable into Prolog.)
7. *Move general quantifiers to the left.*
   (For instance, Composition $[[\forall x P(x)] \wedge [\forall y Q(y)]]$ is adjusted into the Composition $[\forall x [\forall y\ [P(x) \wedge Q(y)]]]$.)
8. *Apply distributive laws*:
   $[[C \wedge D] \vee E] \vdash [[C \vee E] \wedge [D \vee E]$, $[C \vee [D \wedge E]] \vdash [[C \vee D] \wedge [C \vee E]$.

The resulting construction is now of the form

$$\forall x_1 [\forall x_2 \dots [\forall x_n [C_1 \wedge \dots \wedge C_m]]]],$$

where C_i are clauses, i.e., disjunctions of simple 'positive/negative' constructions (not containing constituents of truth-value functions $\wedge, \vee, \supset, \equiv$ and quantifiers). If for some i the clause C_i contains more than one positive disjunct, then *reject* the construction as not transferable into Prolog code. Otherwise, insert the translation of $C_1, \dots, C_m$ into Prolog program base.

Example. Consider the base of sentences.

- "All mobile agents are cars or bicycles".
- "If Charles is driving on highway D1 then he is a mobile agent".
- "Paul is not a mobile agent or it is not true that if he lives in New York he drives his car".

– "It is not true that some agents are neither mobile nor infrastructure agents".

a) Type-theoretical analysis.
Mobile,Infra(structure)/$((o\iota)_{\tau\omega}(o\iota)_{\tau\omega})$; *Agent*, *Car*, *Bike*, *High_Way*, *Employee*/$(o\iota)_{\tau\omega}$; *Live_in*, *Car_of*/$(o\iota\iota)_{\tau\omega}$; *Charles*, *Paul*, $D1, NY/\iota$.
b) Synthesis and pre-processing.

First sentence. $\lambda w\lambda t\ [\forall x[[^0Mobile\ ^0Agent]_{wt}\ x] \supset [[^0Car_{wt}\ x] \vee [^0Bike_{wt}\ x]]]$
$\Rightarrow$ (Step 1) $[\forall x[[^0Mobile\ ^0Agent]x] \supset [[^0Car x] \vee [^0Bike x]]]$
$\Rightarrow$ (Step 4) $[\forall x \neg[[^0Mobile\ ^0Agent]x] \vee [^0Car x] \vee [^0Bike x]]$

This construction is not transferable into Prolog code, because it contains two positive constituents, namely $[^0Car x]$ and $[^0Bike x]$.

Second sentence.
$\lambda w\lambda t\ [[^0Drive_{wt}\ ^0Charles\ ^0D1] \wedge [^0High\_Way_{wt}\ ^0D1]] \supset [[^0Mobile\ Agent]_{wt}\ ^0Charles]]$
$\Rightarrow$ (1) $[[^0Drive\ ^0Charles\ ^0D1] \wedge [^0High\_Way\ ^0D1]]] \supset [[^0Mobile\ Agent]\ ^0Charles]]$
$\Rightarrow$ (4) $[\neg[^0Drive\ ^0Charles\ ^0D1] \vee \neg[^0High\_Way\ ^0D1]] \vee [[^0Mobile\ Agent]\ ^0Charles]]$.
The construction is transferable into a Prolog rule:

```
mobile_agent(charles):-drive(charles,d1),high_way(d1).
```

Third sentence.
$\lambda w\lambda t[\neg[[^0Mobile\ ^0Agent]_{wt}\ ^0Paul] \vee$
$\neg[[^0Live\_in_{wt}\ ^0Paul\ ^0NY] \supset [^0Drive_{wt}\ ^0Paul\ [^0Car\_of_{wt}\ ^0Paul]]]$
$\Rightarrow$ (1) $[\neg[[^0Mobile\ ^0Agent]\ ^0Paul] \vee$
$\neg[[^0Live\_in\ ^0Paul\ ^0NY] \supset [^0Drive\ ^0Paul\ [^0Car\_of\ ^0Paul]]]$
$\Rightarrow$ (4,5) $[\neg[[^0Mobile\ ^0Agent]\ ^0Paul] \vee$
$[[^0Live\_in\ \ ^0Paul\ ^0NY] \wedge \neg[^0Drive\ ^0Paul\ [^0Car\_of\ ^0Paul]]]$
$\Rightarrow$ (8) $[\neg[[^0Mobile\ ^0Agent]\ ^0Paul] \vee [^0Live\_in\ ^0Paul\ ^0NY]] \wedge$
$[\neg[[^0Mobile\ ^0Agent]\ ^0Paul] \vee \neg[^0Drive\ ^0Paul\ [^0Car\_of\ ^0Paul]]]$
The construction is transferable into a Prolog rule and three goals:

```
Live_in(paul,ny):-mobile_agent(paul).
:-mobile_agent(paul), drive(paul,Y), car_of(Y,paul).
```
[9]

Fourth sentence.
$\lambda w\lambda t\neg[\exists x[[^0Agent_{wt}\ x] \wedge \neg[[^0Mobile\ ^0Agent]_{wt} x] \wedge \neg[[^0Infra\ ^0Agent]_{wt}\ x]]]$
$\Rightarrow$ (1) $\neg[\exists x\ [[^0Agent x] \wedge \neg[[^0Mobile\ ^0Agent]x] \wedge \neg[[^0Infra\ ^0Agent]x]]]$
$\Rightarrow$ (5) $[\forall x\ [\neg[^0Agent x] \vee [[^0Mobile\ ^0Agent]x] \vee [[^0Infra\ ^0Agent]x]]]$
The construction is not transferable into a Prolog rule.

Remark. The first step of pre-processing a construction consists in elimination of variables w, t. The result is an improper construction due to wrong typing. For instance, the Composition $[^0Agent x]$ is not well formed, because the property *Agent*/$(o\iota)_{\tau\omega}$ has to be extensionalised first, and only then applied to an individual. Yet, since together with the resulting Prolog rules, facts and goals we remember TIL types, the reverse translation into TIL will be correct.

[9] Since *Car_of* is an attribute, the Composition $[^0Drive\ ^0Paul\ [^0Car\_of\ ^0Paul]]$ is processed by means of the auxiliary variable y; see Section 3.3.1.

4 Conclusion

We have introduced a method of building up an interface between the functional $\mathcal{TIL}$-`Script` language and relational Prolog language. By the transition of TIL into Prolog we gain the inference machine of Prolog. The value we have to pay is rather high. We have to build a powerful TIL functional overlay in order not to lose information, and to modify TIL constructions into the form that can be processed by Prolog. Of course, due to the high expressive power of TIL, only a subset of TIL constructions is transferable. Thus we also specified an algorithm that decides which constructions are transferable, and as a result it produces an adjusted construction specified in the Prolog code.

The direction for future research is clear. We have to extend the method to involve partiality and hyper-intensional features of TIL in its full power. To this end the restrictions applied by Prolog seem to be too tight. Thus we will extend the method into an implementation of the full TIL inference machine. Yet the clarity of this direction does not imply its triviality. The complexity of the work going into building such an inference machine is almost certain to guarantee that complications we are currently unaware of will crop up. A sensible approach will be to develop the inference machine by involving the methods of functional programming and extend them to involve partiality and hyper-intensional, i.e., procedural features.

Acknowledgements. This research has been supported by the program 'Information Society' of the Czech Academy of Sciences within the project "Logic and Artificial Intelligence for multi-agent systems", No. 1ET101940420.

References

1. Bratko, I.: *Prolog Programming for Artificial Intelligence.* Addison-Wesley, 2001.
2. Ciprich, N., Duží, M., Košinár, M.: $\mathcal{TIL}$-`Script` : Functional Programming Based on Transparent Intensional Logic. In: *RASLAN 2007*, Sojka, P., Horák, A., (Eds.), Masaryk University Brno, 2007, pp. 37–42.
3. Ciprich, N., Duží, M. and Košinár, M.: *The $\mathcal{TIL}$-`Script` language*. To appear in the Proceedings i of the 18th European Japanese Conference on Information Modelling and Knowledge Bases (EJC 2008), Tsukuba, Japan 2008.
4. Duží, M. Ďuráková, D., Děrgel, P., Gajdoš, P., Müller, J.: Logic and Artificial Intelligence for multi-agent systems. In: Marie Duží, Hannu Jaakkola, Yasushi Kiyoki and Hannu Kangassalo (editors), *Information Modelling and Knowledge Bases* XVIII, Amsterdam: IOS Press, pp. 236–244.
5. Duží, M. and Materna, P. Reprezentace znalostí, analýza tázacích vět a specifikace dotazů nad konceptuálním schématem HIT. In: Dušan Chlápek (Ed.), *Datakon 2002*, Brno, pp. 195–208.
6. Foundation for Intelligent Physical Agents. `http://www.fipa.org/`.
7. Tichý, P. *The Foundations of Frege's Logic*. Walter de Gruyter, Berlin-New York, 1988.
8. Tichý, P. (2004): *Collected Papers in Logic and Philosophy*, V. Svoboda, B. Jespersen, C. Cheyne (Eds.), Prague: Filosofia, Czech Academy of Sciences, and Dunedin: University of Otago Press.