Induction of Recursive Transfer Rules

Henrik Boström
Dept. of Computer and Systems Sciences
Stockholm University
Electrum 230, 164 40 Kista, Sweden
henke@dsv.su.se

Abstract

Transfer rules are used in bi-lingual translation systems for transferring a logical representation of a source language sentence into a logical representation of the corresponding target language sentence. This work studies induction of transfer rules from examples of corresponding pairs of source-target quasi logical formulae (QLFs). The main features of this problem are: i) more than one rule may need to be produced from a single example, ii) only positive examples are provided and iii) the produced hypothesis should be recursive. In an earlier study of this problem, a system was proposed in which hand-coded heuristics were employed for identifying non-recursive correspondences. In this work we study the case when non-recursive transfer rules have been given to the system instead of heuristics. Results from a preliminary experiment with English-French QLFs are presented, demonstrating that this information is sufficient for the generation of generally applicable rules that can be used for transfer between previously unseen source and target QLFs. However, the experiment also shows that the system suffers from producing overly specific rules, even when the problem of disallowing the derivation of other target QLFs than the correct one is not considered. Potential approaches to this problem are discussed.

1 Introduction

In transfer-based translation, source language input is first analysed resulting in a logical representation of the (preferred) meaning of the input utterance. Next, the source logical formula is transferred into a logical representation in the target language. Finally, the such obtained logical formula is then used for generating the target text. Each language is processed with its specific morphology, grammar, lexicon, etc., with the transfer being the only bridge between them. The Spoken Language Translator (SLT) [2] is a transfer-based translation system, that is based on the Core Language Engine (CLE) [1] for mapping between natural language phrases and (quasi) logical formulas (QLFs). It uses so-called transfer rules for the mapping between source and target QLFs.

A transfer rule specifies a pair of logical form patterns, where the first pattern represents a form in one language and the second pattern represents a form in the other language. The patterns can include so-called transfer variables showing the recursive correspondence between parts of the matching logical forms. Many transfer rules are only responsible for transferring word senses, e.g. `trule(leave_Depart,partir_Leave)`, while others, the so-called structural transfer rules, are more complex, e.g. the following rule, which is taken from [8][1]:

```
trule([stop_ComeToRest,A,S1],
      [faire_Make,A,S2,term(q(_,bare,sing),_,
                            X^[escale_Stop,X])]):-
      trule(S1,S2).
```

Transfer rules are applied recursively, and this process follows the recursive structure of the source QLF. Normally, the transfer rules are hand-crafted through inspection of a set of non-transferable QLF pairs. Their creation is a tedious and time-consuming task. The main problem addressed in this work is how to use inductive logic programming (ILP) techniques for automatic derivation of transfer rules from examples of corresponding QLF pairs, such as:

```
qlf_pair([imp, form(_,verb(no,no,no,imp,y),A,
          B^[B,[list_Enumerate,A,
                  term(_,ref(pro,you,_,l([])),_,
                               C^[personal,C],_,_),
                  term(_,q(_,bare,plur),_,
                         D^[fare_Price,D],_,_)]],_)],
          [imp, form(_,verb(impera,no,no,impera,y),E,
           F^[F,[indiquer_Show,E,
                  term(_,ref(pro,vous,_,l([])),G,
                               H^[personal,H],_,_),
                  term(_,ref(def,le,plur,l([G-_])),_,
                          I^[tarif_Fares,I],_,_)]],_)]).
```

The main features of this problem are:

- More than one rule may need to be produced from a single example.
- Only positive examples are provided.
- The produced hypothesis should be recursive.

[1]The reader is assumed to be familiar with logic programming terminology [7] and standard Edinburgh syntax for logic programs [5].

The first problem is significant as most ILP systems (e.g. Progol [10] and FOIL [12]) produce at most one clause per example. We have developed one approach to this problem, a system called TRL (Transfer Rule Learner) [4], which works in three steps: first, it tries to identify non-recursive correspondences between the source and target QLFs, second, it generates a set of as general clauses as possible for each example and third, it specialises the clauses so that for each source QLF exactly one target QLF can be generated (cf. *output completeness* [9]).

In the previous work, the system relied on elaborated, hand-coded heuristics for the first step. In this work we assume no such heuristics, but instead assume that the non-recursive transfer rules are known, and hence the task is to induce the recursive rules (initial work on inducing non-recursive transfer rules is described in [8]). In the next section, we present the basic algorithm used in TRL for solving this task. In section three, we present results from a preliminary experiment on learning transfer rules from English-French QLF pairs and point out some major difficulties that are revealed by this experiment. In section four we give some concluding remarks and discuss some possible directions for future work.

2 The Transfer Rule Learner

Given a set of pairs of input-output terms (QLF pairs), the rule generating component of TRL produces a set of clauses such that for each input term, (a variant of) the corresponding output term can be derived (the problem of excluding the derivation of other terms than the output term is assumed to be handled later by the rule specialisation component). The rule generating component is based on the following assumptions: i) there is some way to find a bijection from sub-terms in each input term to sub-terms in the corresponding output term that corresponds to all sub-terms that should be non-recursively transferred ii) only one (recursive) predicate is needed in the produced hypothesis, and iii) both arguments in the head of each clause produced must be compound (this ensures that the hypothesis terminates for all input).

Given an example pair, the objective of the rule generating component is to find rules that will transfer the source term into the target, and that these rules are as general as possible in order to cover as many similar cases as possible. To be able to achieve this, the corresponding terms should be distributed among several clauses, where each clause will be applicable to transferring (sub-)terms of QLF pairs. Since it is computationally infeasible to decide how to distribute a pair of terms between clauses such that a most general hypothesis is obtained, a greedy strategy is adopted: rules are generated in a top-down fashion (i.e. in the same order as they will be applied when deriving the target from the source), and each generated rule is a most general rule for the corresponding input-output pair (i.e. a rule with the most general head such that when instantiated with the input-output pair, the literals in the body correspond to input-output pairs for which transfer rules can be generated, i.e there are no variable connections to other literals or to the head (except for literals containing just a pair of transfer variables, which should have connections with the head only). It should be noted that there is always a unique clause with this property.).

In Figure 1 we show the rule generating component of TRL, that given an input-output pair, where sub-terms that should be non-recursively transferred have been replaced by transfer variables, generates a set of transfer rules that is as general as possible while still being able to recreate (a variant of) the output term from the input term. The function, called Find-Transfer-Rules, takes as input the input-output pair, a set of transfer variable pairs and a (initially empty) set of transfer rules that has been produced previous to the call.

---

function Find-Transfer-Rules($f(s_1, \ldots, s_m), g(t_1, \ldots, t_n), V, H$)
  $R := \{trule(f(x_1, \ldots, x_m), g(y_1, \ldots, y_n)\}$
  $\theta_S := \{x_1/s_1, \ldots, x_m/s_m\}$ and $\theta_T := \{y_1/t_1, \ldots, y_n/y_n\}$
  repeat
    if there is a term $s = x_i/f_i(u_1, \ldots, u_k) \in \theta_S(\theta_T)$, such that some variable in $u_1, \ldots, u_k$ occurs in R or $\theta_S(\theta_T) \setminus \{s\}$ or there are two distinct terms t_1 and t_2 in $\theta_T(\theta_S)$, where t_1 contains w_1 and t_2 contains w_2 such that $\{\{v_1, w_1\}, \{v_2, w_2\}\} \subseteq V$ for some variables v_1 and v_2 in s **then**
      $\theta_S(\theta_T) := \theta_S(\theta_T) \setminus \{s\} \cup \{z_1/u_1, \ldots z_k/u_k\}$
      $R := R\{x_i/f_i(z_1, \ldots, z_k)\}$ **else**
    if there is a pair (s, t), where $s = x_i/u_i \in \theta_S, t = y_j/v_j \in \theta_T$, and $\{u_i, v_j\} \in V$ **then**
      $R := R \cup \{\leftarrow trule(x_i, y_j)\}$, $\theta_S := \theta_S \setminus \{s\}$, and $\theta_T := \theta_T \setminus \{t\}$
    else if there is a pair (s, t), where $s = x_i/s_i \in \theta_S, t = y_j/t_j \in \theta_T$, s_i is compound and contains u_i, t_j is compound and contains v_j, where $\{u_i, v_j\} \in V$ **then**
      $R := R \cup \{\leftarrow trule(x_i, y_j)\}$, $\theta_S := \theta_S \setminus \{s\}$, and $\theta_T := \theta_T \setminus \{t\}$
      $H :=$ Find-Transfer-Rules(s_i, t_j, V, H) **else**
    $R := R\theta_S\theta_T$ and $\theta_S := \theta_T := \emptyset$
  until $\theta_S = \theta_T = \emptyset$
  return $H \cup \{R\}$

Figure 1: The function for finding recursive transfer rules.

An example

Assume the following is given as input to the algorithm Find-Transfer-Rules: the input term `s(f(V1,V2),g(V3))`, the output term `t(h(W3),W2,W1)` and the set of transfer variable pairs $V = \{\{\texttt{V1,W1}\}, \{\texttt{V2,W2}\}, \{\texttt{V3,W3}\}\}$. Then the initialisation steps result in the following:

$R =$ `trule(s(X1,X2),t(Y1,Y2,Y3)).`
$\theta_S = \{$`X1/f(V1,V2), X2/g(V3)`$\}$
$\theta_T = \{$`Y1/h(W3), Y2/W2, Y3/W1`$\}$

Since the transfer variables `V1` and `V2` that occur in the first term in θ_S correspond to transfer variables (`W1` and `W2`) that occur in two different terms in θ_T, it follows from the first *if*-statement that `X1` is substituted by a term `f(X3,X4)` in R, and that the terms `X3/V1` and `X4/V2` replace the first term in θ_S:

$R =$ `trule(s(f(X3,X4),X2),t(Y1,Y2,Y3)).`
$\theta_S = \{$`X3/V1, X4/V2, X2/g(V3)`$\}$
$\theta_T = \{$`Y1/h(W3), Y2/W2, Y3/W1`$\}$

Now, according to the second *if*-statement, two recursive calls are added to R in turn, subtracting `X3/V1` from θ_S and `Y3/W1` from θ_T after having added the first recursive call, and subtracting `X4/V2` from θ_S and `Y2/W2` from θ_T after having added the second recursive call:

$R =$ `trule(s(f(X3,X4),X2),t(Y1,Y2,Y3)):- trule(X3,Y3), trule(X4,Y2).`
$\theta_S = \{$`X2/g(V3)`$\}$
$\theta_T = \{$`Y1/h(W3)`$\}$

Following the third *if*-statement, a third recursive call `trule(X2,Y1)` is added to R, resulting in the following rule that is included in H:

```
trule(s(f(X3,X4),X2),t(Y1,Y2,Y3)):-
        trule(X3,Y3), trule(X4,Y2), trule(X2,Y1).
```

The algorithm Find-Transfer-Rules is then invoked recursively with the subterms `g(V3)` and `h(W3)` as input, resulting in the following rule that also is included in H:

```
trule(g(X),h(Y)):- trule(X,Y).
```

3 An Experiment

3.1 Experimental Data

540 English-French QLF pairs have been obtained from SRI-Cambridge. These were formed by running the SLT system which has accuracy of over 95% on the ATIS 2 corpus. Just one target QLF was generated for each source QLF. The system uses statistical methods to choose the best QLF which is both a good French sentence and a good translation of the original (according to weighted transfer rules).

In addition, the set of non-recursive transfer rules used by SLT was obtained from SRI as well, consisting of 1155 rules. These were used in the following way.

For each QLF pair, a bijection from a set of subterms in the source to a set of subterms in the target was formed using the non-recursive rules, and the corresponding subterms were replaced by transfer variables. Note that not all pairs of subterms for which there is a matching non-recursive rule can be replaced by transfer variables, due to that the same subterm may go into several subterms on the opposite side and due to that variable connections might get lost (i.e. some variable occurs both inside and outside the subterm) - in these cases the subterms were left unchanged.

The induced rules were in the current experiment forced to be on the following form: either the functor and arity of the source and target QLF should be identical, or the source should be a list with two elements and the target have the functor `form` and arity 5 or vice versa. This turned out to work better than allowing any form of the rules.

3.2 Experimental Results

Subsets of the set of pairs in which transfer variables have been introduced were given as input to TRL[2] (except for 40 pairs that were leaved out for testing). The rules produced by the system were then tried on the test set, and it was checked whether the first target QLF produced for each (test) source QLF, was a variant of the correct (test) target QLF (the rules were tested in the same order as they were generated). It was also checked whether the correct target QLF could be produced at all. The performance was compared to just storing the pairs that were given as input (which still are more general than the original set of QLF pairs as e.g. lexical items have been replaced by transfer variables).

Average results from running the experiment 10 times are summarised in Table 1. The number of examples given as input to TRL is shown in the first column. The coverage, measured as the fraction of the test set for which the target QLF can be generated at all, is shown in the second column. The third column shows the fraction of the test set for which the first target QLF generated is correct. The fourth and fifth column shows the number of rules generated

[2] The specialisation step described in [4] was omitted.

and the cputime[3] in seconds respectively. The last column shows the coverage obtained from just storing the pairs that were given as input.

No. ex.	Cov.	1^{st} cov.	No. clauses	Time (s.)	No. rec.
10	0.06	0.06	35.2	7.1	0.02
50	0.12	0.10	106.0	41.8	0.08
100	0.18	0.16	170.7	80.7	0.13
200	0.27	0.23	277.1	163.9	0.18
300	0.34	0.30	361.6	251.3	0.22
400	0.40	0.35	437.2	337.2	0.25
500	0.45	0.39	508.3	424.3	0.27

Table 1: Average results from 10 iterations.

It could be seen that although TRL in fact tries to generate as general rules as possible, it does not suffer significantly from producing too non-determinate rules, as indicated by the relatively small difference in coverage and 1^{st} coverage. The results rather indicate that TRL suffers from being overly specific. Some explanations for this problem are given in the next section.

3.3 Comments

The rules that are produced are of varying complexity. Here follows the initial sequence of rules produced in one of the sessions (they have been rewritten in a form which should be more readable - the variables in the actual rules produced are instantiated with terms on the form `$var(N)` and there are also recursive calls to `trule/2` instead of transfer variables):

```
trule([tr(1)|tr(2)],
      [tr(1)|tr(2)]).

trule([tr(1)],
      [tr(1)]).

trule(form(_,verb(pres,no,no,no,y),A,
        B^[B,form(_,tr(1),_,C^[C,v(A)|tr(2)],_),[tr(3),A|tr(4)]],_),
      form(_,verb(pres,no,no,no,y),D,
        E^[E,form(_,tr(1),_,F^[F,v(D)|tr(2)],_),[tr(3),D|tr(4)]],_)).

...
```

[3] TRL was implemented in SICStus Prolog v. 3 and was executed on a SUN Ultra 60.

```
trule([term(_,tr(1),A,tr(2),_,_),
                term(_,ord(ref(def,the,sing,l([A-_])),
                B^[cheap_NotExpensive,B],order,'N'('1'),sing),C,
                D^[and,[and,[tr(3),D],['one way_TravellingThereOnly',D]],
                form(_,tr(4),_,
                E^[E,form(_,prep(to),_,F^[F,C|tr(5)],_),
                form(_,prep(from),_,G^[G,C|tr(6)],_)],_)],_,_)],
      [term(_,tr(1),_,tr(2),_,_),
                term(_,ord(ref(def,le,sing,l([])),
                H^[cher_Expensive,H],reverse_order,'N'('1'),sing),I,
                J^[and,[and,[tr(3),J],[aller_simple_OneWay,J]],
                form(_,tr(4),_,
                K^[K,form(_,prep(implicit_to),_,L^[L,I|tr(5)],_),
                form(_,prep(implicit_from),_,L^[L,I|tr(6)],_)],_)],_,_)]).
```

Comment: The non-recursive transfer rule:

```
trule([and,[flight_AirplaneTrip,A],['one way_TravellingThereOnly',A]]
         ==['aller simple_OneWayFlight',A]).
```

has not been applied to the QLF-pair from which the above recursive rule stems, as the variables D and J occur outside the subterms.

```
trule(A^[name_of,A|tr(1)],
      B^[name_of,B|tr(1)]).
```

...

```
trule(term(_,q(_,all,plur),A,B^[and,[and,[tr(1),B],
                form(_,verb(no,no,yes,no,y),C,
                D^[D,form(_,prep(for),_,E^[E,v(C)|tr(2)],_),
                [leave_GoAwayFrom,C,v(A)|tr(3)]],_)],
                [island,form(_,verb(pres,no,no,no,y),F,
                G^[G,form(_,tr(4),_,H^[H,v(F)|tr(5)],_),
                [depart_Leave,F,v(A)]],_)]],_,_),
      term(_,q(_,tous_les,plur),I,J^[and,[and,[tr(1),J],
                [island,form(_,verb(pres,no,no,no,y),K,
                L^[L,form(_,conj(pp,implicit_and),_,
                M^[M,form(_,prep(pour),_,N^[N,v(K)|tr(2)],_),
                form(_,prep(de_Directional),_,O^[O,v(K)|tr(3)],_)],_),
                [partir_Leave,K,v(I)]],_)]],
                [island,form(_,verb(pres,no,no,no,y),P,
                Q^[Q,form(_,tr(4),_,R^[R,v(P)|tr(5)],_),
                [partir_Leave,P,v(I)]],_)]],_,_)).
```

Comment: `leave_GoAwayFrom` has not been replaced by a transfer variable as it can only go into `quitter_Leave`, which is not present. This causes `depart_Leave` to be left too as there are two occurrences of `partir_Leave`.

It should be noted that the technique is heavily dependent on the initially introduced transfer variables - if these are not placed properly the resulting recursive rules will most likely be inaccurate. In the current system, the strategy for introducing the initial transfer variables is quite simple-minded. In particular, a subterm that occurs more than once in a QLF is never replaced by a transfer variable. Much could be won if this restriction is relaxed. For example, one could use some elaborate heuristic for coupling non-unique sub-terms. But it should be noted that in some cases all couplings should be rejected. For example, consider the following QLF-pair:

```
qlf_pair([imp,form(_,adv(please_BeGratefulIf),_,
                A^[A,form(_,verb(no,no,no,imp,y),B,
                C^[C,[list_Enumerate,B,term(_,ref(pro,you,_,l([])),_,
                D^[personal,D],_,_),term(_,q(E,only,_),F,
                G^[and,[sub,G,term(_,q(E,bare,plur),_,
                H^[flight_AirplaneTrip,H],_,_)],
                [island,form(_,verb(pres,no,no,no,y),I,J^[J,
                form(_,prep(after),_,
                K^[K,v(I),term(_,time(timeofday),_,
                L^[and,[hour_num,L,'N'('11')],
                [and,[minute_num,L,'N'('0')],
                [day_part,L,morning]]],_,_)],_),
                [depart_Leave,I,v(F)]],_)]],_,_)]],_)],_)],
         form(_,interjection('s\'il vous plait_Interjection'),_,
                M^[M,[imp,form(_,verb(impera,no,no,impera,y),N,
                O^[O,[indiquer_Show,N,term(_,ref(pro,vous,_,l([])),P,
                Q^[personal,Q],_,_),term(_,q(_,seulement,_),R,
                S^[and,[sub,S,term(_,ref(def,le,plur,l([P-_])),_,
                T^[vol_Flight,T],_,_)],
                [island,form(_,verb(pres,no,no,no,y),U,
                V^[V,form(_,prep(aprs),_,W^[W,v(U),
                term(_,time(timeofday),_,
                X^[and,[hour_num,X,'N'('11')],
                [minute_num,X,'N'('0')]],_,_)],_),
                [partir_Leave,U,v(R)]],_)]],_,_)]],_)]],_)).
```

None of the occurrences of `imp` in the source QLF and the (only) occurrence of `imp` in the target QLF should be replaced by a transfer variable.

4 Concluding Remarks

We have presented the application of a prototype system, called TRL, to the problem of inducing recursive transfer rules, given examples of corresponding QLF pairs and a set of non-recursive transfer rules. An initial experiment demonstrates that this information is sufficient for the generation of generally applicable rules that can be used for transfer between previously unseen source and target QLFs. However, the experiment also demonstrates that the system suffers from producing overly specific rules, even when the problem of disallowing the derivation of other target QLFs than the correct one is not considered. This is mainly due to the inability of appropriately using the non-recursive rules for introducing transfer variables prior to the generation of the recursive rules.

One immediate approach to this problem is to relax the conservative condition that a subterm that occurs more than once may never be replaced by a transfer variable, e.g. by using some heuristic for selecting which subterm in the source that should go into a particular subterm in the target (and vice versa).

When the problem of producing overly specific rules in the first phase of TRL has been overcome, there are several possibilities for handling the problem with having more than one candidate target QLF that can be generated for a particular source QLF. One is to specialise the program by introducing new predicate symbols, e.g. as in [3]. Another possibility is to look at probabilistic extensions, such as *stochastic logic programs* [11, 6], and choose the target QLF that is given the highest probability.

Another approach to the problem of learning transfer rules from QLF-pairs is to reduce the complexity of the learning task by utilising the grammar rules and lexica that are used when generating the source and target QLFs (these were not available in this study). Since the source and target QLFs can be reconstructed given parse trees that refer only to identifiers of the grammar rules and lexical items, the transfer rule learning problem can be reduced to the problem of learning a mapping from such parse trees for source sentences into parse trees for target sentences. While still a challenging problem, the non-determinism inherent in the task is significantly reduced by using this indirect approach compared to inducing rules directly from the QLF pairs.

Acknowledgements

This work was supported by ESPRIT LTR project no. 20237 Inductive Logic Programming II and the Swedish Research Council for Engineering Sciences (TFR). The author would like to thank to David Milward and Stephen Pulman at SRI-Cambridge for providing the data and for helpful discussions.

References

[1] Alshawi H. (ed.), *The Core Language Engine*, MIT Press (1992)

[2] Agnäs M-S., Alshawi H., Bretan I., Carter D., Ceder K., Collins M., Crouch R., Digalakis V., Ekholm B., Gambäck B., Kaja J., Karlgren J., Lyberg B., Price P., Pulman S., Rayner M., Samuelsson C. and Svensson T., *Spoken-Language Translator: First-Year Report*, SICS research report, ISRN SICS-R–94/03–SE (1994)

[3] Boström H., "Predicate Invention and Learning from Positive Examples Only", *Proceedings of the 10th European Conference on Machine Learning*, Springer-Verlag (1998) 226–237

[4] Boström H. and Zemke S., "Learning Transfer Rules by Inductive Logic Programming (Preliminary Report)", Dept. of Computer and Systems Sciences, Stockholm University and Royal Institute of Technology (1996)

[5] Clocksin W.F. and Mellish C.S, *Programming in Prolog*, Springer Verlag (1981)

[6] Cussens J., "Loglinear models for first-order probabilistic reasoning", *Proc. of Uncertainty in Artificial Intelligence* (1999)

[7] Lloyd J. W., *Foundations of Logic Programming*, (2nd edition), Springer-Verlag (1987)

[8] Milward D. and Pulman S., "Transfer learning using QLFs", Technical report, SRI International (1997)

[9] Mooney R. J. and Califf M. E., "Learning the Past Tense of English Verbs Using Inductive Logic Programming", in Wermter S., Riloff E. and Scheler G. (eds.), *Symbolic, Connectionist and Statistical Approaches to Learning for Natural Language Pr ocessing*, Springer-Verlag (1996) 370–384

[10] Muggleton S., "Inverse entailment and Progol", *New Generation Computing* **13** (1995)

[11] Muggleton S., "Stochastic Logic Programs", in De Raedt L. (ed.), *Advances Inductive Logic Programming*, IOS Press (1995) 254–264

[12] Quinlan J. R., "Learning Logical Definitions from Relations", *Machine Learning* **5** (1990) 239–266