An Open Source Punjabi Resource Grammar

Shafqat Mumtaz Virk **Muhammad Humayoun** **Aarne Ranta**
University of Gothenburg, Sweden University of Savoie, France University of Gothenburg, Sweden
virk@chalmers.se humayoun@gmail.com aarne@chalmers.se

Abstract

We describe an open source computational grammar for Punjabi; a resource-poor language. The grammar is developed in GF (Grammatical framework), which is a tool for multilingual grammar formalism. First, we explore different syntactic features of Punjabi and then we implement them in accordance with GF grammar requirements, to make Punjabi the 17th language in the GF resource grammar library.

1. Introduction

Grammatical Framework (Ranta, 2004) is a special-purpose programming language for multilingual grammar applications. It can be used to write multilingual *resource* or *application* grammars (two types of grammars in GF).

Multilingualism of the GF grammars is based on the principle that same grammatical categories (e.g. noun phrases and verb phrases) and syntax rules (e.g. predication) can appear in different languages (Ranta, 2009a). A collection of all such categories and rules, which are independent of any language, makes the abstract syntax of GF grammars (every GF grammar has two levels: abstract and concrete). More precisely, the abstract syntax defines semantic conditions to form abstract syntax trees. For example the rule that a common noun can be modified by an adjective is independent of any language and hence is defined in the abstract syntax, e.g.:

Very big blue house
`fun`[1] `AdjCN : AP → CN → CN ;`

However, the way this rule is implemented may vary from one language to another; as each language may have different word order and/or agreement rules. For this purpose, we have the concrete syntax, which is a set of linguistic objects (strings, inflection tables, records) providing rendering and parsing. We may have multiple parallel concrete syntaxes for one abstract syntax, which makes the GF grammars multilingual. Also, as each concrete syntax is independent from others, it becomes possible to model the rules accordingly (i.e. word order, word forms and agreement features are chosen according to language requirements).

Current state-of-the-art machine translation systems such as Systran, Google Translate, etc. provide huge coverage but sacrifice precision and accuracy of translations. On the contrary, domain-specific or controlled multilingual grammar based translation systems can provide a higher translation quality, on the expense of limited coverage. In GF, such controlled grammars are called *application grammars*.

Writing application grammars from scratch can be very expensive in terms of time, effort, expertise and money. GF provides a library called the *GF resource library* that can ease this task. It is a collection of linguistic oriented but general-purpose *resource grammars*, which try to cover the general aspects of different languages (Ranta, 2009a).

Instead of writing application grammars from scratch for different domains, one may use resource grammars as libraries (Ranta, 2009b)[2]. This method enables to create the application grammar much faster with a very limited linguistic knowledge.

The number of languages covered by GF resource library is growing (17 including Punjabi). Previously, GF and/or its libraries have been used to develop a number of multilingual as well as monolingual domain-

[1] In GF code, `cat` and `fun` belong to abstract syntax. On the contrary, `lincat` and `lin` belong to concrete syntax.

[2] This idea is influenced by programming language API tradition in which, a standard general-purpose library is supported by the language. It is then used by programmers to write specific applications.

Proceedings of Recent Advances in Natural Language Processing, pages 70–76,
Hissar, Bulgaria, 12-14 September 2011.

specific application grammars (see GF homepage [3] for details on these application grammars).

In this paper, we describe the resource grammar development for Punjabi. Punjabi is an Indo-Aryan language widely spoken in Punjab regions of Pakistan and India. Punjabi is among one of the morphologically rich languages (others include Urdu, Hindi, Finish, etc) with SOV word order, partial ergative behavior, and verb compounding. In Pakistan it is written in *Shahmukhi*, and in India, it is written in *Gurmukhi* script (Humayoun, 2010). Language resources for Punjabi are very limited (especially for the one spoken in Pakistan). With the best of our knowledge this work is the first attempt of implementing a computational Punjabi grammar as open-source software, covering a fair enough part of Punjabi morphology and syntax.

2. Morphology

Every grammar in GF resource grammar library has a test lexicon, which is built through the lexical functions called the lexical paradigms; see (Bringert et el, 2011) for synopsis. These paradigms take lemma of a word and make finite inflection tables, containing different forms of the word, according to the lexical rules of that particular language. A suite of Punjabi resources including morphology and a big lexicon are reported by (Humayoun and Ranta, 2010). With minor required adjustments, we have reused morphology and a subset of that lexicon, as a test lexicon of about 450 words for our grammar implementation. However, the morphological details are beyond the scope of this paper and we refer to (Humayoun and Ranta, 2010) for more details on Punjabi morphology.

3. Syntax

While morphology is about types and formation of individual words (lexical categories), it is the syntax, which decides how these words are grouped together to make well-formed sentences. For this purpose, individual words, which belong to different lexical categories, are converted into richer syntactic categories, i.e. noun phrases (NP), verb phrases (VP), and adjectival phrases (AP), etc. With this up-cast the linguistic features such as word-forms, number & gender information, and agreements, etc, travel from individual words to the richer categories.

In this section, we explain this conversion from lexical to syntactic categories and afterwards, we demonstrate how to glue the individual pieces to make clauses. These are then can be used to make well-formed sentences in Punjabi. The following subsections explain various types of phrases.

3.1. Noun Phrases

A noun phrase (`NP`) is a single word or a group of words that does not have a subject and a predicate of its own, and does the work of a noun (Verma, 1974). Now we show the structure of noun phrase in our implementation, followed by the description of its different parts.

Structure: In GF, we represent the `NP` as a record with three fields, labeled as: '`s`' , '`a`' and '`isPron`':

```
NP: Type={s      : NPCase => Str ;
          a      : Agr  ;
          isPron : Bool } ;
```

The label '`s`' is an inflection table from `NPCase` to string (`NPCase => Str`). `NPCase` has two constructs (`NPC Case`, and `NPErg`) as shown below:

```
NPCase = NPC Case | NPErg ;
Case   = Dir | Obl | Voc | Abl ;
```

The construct (`NPC Case`) stores the lexical cases (i.e. Direct, Oblique, Vocative and Ablative) of a noun[4]. As an example consider the following table for the noun "boy":

s .NPC Dir	=>	mʊnɖɑ:	مُنڈا
s .NPC Obl	=>	mʊnɖɛ	مُنڈے
s .NPC Voc	=>	mʊnɖi:a	مُنڈیا
s .NPC Abl	=>	mʊnɖɛo:ɳ	مُنڈیوں

Other than storing the lexical cases of a noun as shown in the above table, we also construct the ergative case (i.e. `NPErg` in the code above). We do it at the noun phrase level for the

[3] http://www.grammaticalframework.org/

[4] Punjabi nouns have four lexical cases.

following reason: In Urdu, the case markers that follow the nouns in the form of post-positions cannot be handled at lexical level through morphological suffixes and thus need to be handled at syntax level (Butt and King, 2002)[5]. It also applies to Punjabi. So we construct the ergative case of a noun by attaching ergative case marker 'nɛ' to the oblique case of the noun at NP level. For instance, the ergative form of our running example "boy" is:

```
s.NPErg => mʊnɖɛ nɛ_Erg مُنڈے نے
```

It is used for the subjects of perfective transitive verbs (see Section 3.5 for more details).

The label 'a' represents the agreement feature (Agr) and stores information about gender, number and person that will be used for agreement with other constituents. It is defined as follows:

```
Agr     = Ag Gender Number Person ;
```

In Punjabi, the gender can be *masculine* or *feminine*; number can be *singular* and *plural*; and person can be first, second casual, second with respect and third person near & far. These are defined as shown below:

```
Gender  =  Masc | Fem ;
Number  =  Sg | Pl ;
Person  =  Pers1 | Pers2_Casual |
           Pers2_Respect |
           Pers3_Near | Pers3_Far
```

Finally, the label 'isPron' is a Boolean parameter, which shows whether NP is constructed from a pronoun. This information is important when dealing with the exceptions in ergative behavior of verbs for the first and second person pronouns in Punjabi. For example consider the following constructions:

mi:ɳ_I ro:ʈi:_bread kʰadi:_ate
میں روٹی کھادی
I ate bread.
tu:ɳ_you ru:ʈi:_bread kʰaɖi:_ate
تُوں روٹی کھادی
You ate bread.
au:nɛ:_He ru:ʈi:_bread kʰadi:_ate
اونے روٹی کھادی
He ate bread.

mʊnɖɛ:_boy nɛ:_ErgMarker ru:ʈi:_bread kʰadi:_ate
مُنڈے نے روٹی کھادی
The boy ate bread.

From the above examples, we can see that, when we have the first or second person pronoun as subject, the ergative case marker is not used (first two examples). On the contrary, it is used in all other cases. So for our running example, i.e. the noun (boy, mʊnɖɑ:), the label 'isPron' is false.

Construction: First, the lexical category noun (N) is converted to an intermediate category, common noun (CN) through the UseN function.

```
fun UseN : N → CN ;      -- mʊnɖɑ:, boy
```

CN is a syntactic category, which is used to deal with the modifications of nouns by adjectives, determiners, etc. Then, the common noun is converted to the syntactic category, noun phrase (NP). Three main types of noun phrases are: (1) common nouns with determiners, (2) proper names, and (3) pronouns. We build these noun phrases through different noun phrase construction functions depending on the constituents of NP. As an example consider (1). We define it with a function DetCN given below:

Every boy, hər_every mʊnɖɑ:_boy
```
fun DetCN : Det → CN → NP ;
```

Here (Det) is a lexical category representing determiners. The above given function takes the determiner (Det) and the common noun (CN) as parameters and builds the NP, by combining appropriate forms of the determiner and the common noun agreeing with each other. For example if 'every' and 'boy' are the parameters for the above given function the result will be a NP: every boy, *hər mʊnɖɑ:*. Consider the linearization of DetCN:

```
lin DetCN det cn = {
 s=\\c => detcn2NP det cn c det.n;
 a = agrP3 cn.gdet.n ;
 isPron = False } ;
```

As we know from the structure of NP (given in the beginning of §3.1) 's' represents the inflection table used to store different forms of NP built by the following line from the above code:

```
s = \\c => detcn2NP det cn c det.n;
```

[5]This also explains the reason for NPErg to be separate from "NPC Case".

Notice that the operator ('\\') is used as shorthand to represent different rows of the inflection table 's'. An alternative but a verbose code segment for the above line will be:

```
s = table {
NPC Dir=>detcn2NP det cn Dir det.n;
NPC Obl=>detcn2NP det cn Obl det.n;
NPC Voc=>detcn2NP det cn Voc det.n;
NPC Abl=>detcn2NP det cn Abl det.n}
```

Where the helper function `detcn2NP` is defined as:

```
detcn2NP : Determiner → CN → NPCase
→ Number → Str =
\dt,cn,npc,n → case npc of {
  NPC c => dt.s  ++ cn.s!n!c ;
  NPErg => dt.s++cn.s!n!Obl++"nɛ"};
```

Also notice that the selection operator (the exclamation sign `!`) is used to select appropriate forms from the inflection tables (i.e. `cn.s!n!c`, which means the form of the common noun with number 'n' and case 'c' from the inflection table `cn.s`).

Other main types of noun phrases (2) and (3) are constructed through the following functions.

```
fun UsePN   : PN → NP ;   Ali, əli:
fun UsePron : Pron → NP ; he, æ:h
```

This covers only three main types of noun phrases, but there are other types of noun phrases as well, i.e. adverbial post-modified `NP`, adjectival modified common nouns etc. In order to cover them we have one function for each such construction. Few of these are given below; for full details we refer to (Bringert et el, 2011).

Paris today, $aj_{\_today}\ pi{:}rəs_{\_Paris}$

```
fun AdvNP : NP → Adv → NP ;
```

Big house, $ʊadda{:}_{\_big}\ g^{ʰ}ər_{\_house}$

```
fun AdjCN : AP → CN  → CN ;
```

3.2. Verb Phrases

A verb phrase (`VP`), as a syntactic category, is the most complex structure in our constructions. It carries the main verb and auxiliaries (such as adverb, object of the verb, type of the verb, agreement information, etc), which are then used in the construction of other categories and/or clauses.

Structure: In GF, we represent a verb phrase as a record, as shown below:

```
VPH : Type = {
    s:VPHForm => {fin, inf : Str};
    obj : {s : Str ; a : Agr} ;
    subj: VType ;
    comp: Agr =>Str;
    ad  : Str ;
    embComp : Str} ;
```

The label 's' represents an inflection table which keeps a record with two string values, i.e. `{fin, inf : Str}` for every value of `VPHForm`, which is defined as shown below:

```
VPHForm =
  VPTense VPPTense Agr|VPInf|VPStem ;
VPPTense=
  PPres|VPPast|VPFutr|VPPerf;
```

The structure of `VPHForm` makes sure that we preserve all inflectional forms of the verb. In it we have three cases: (1) Inflectional forms inflecting for tense (`VPPTense`) and number, gender, person with `Agr` defined on page 3. (2) The second constructor (`VPInf`) carries the infinitive form. (3) On the contrary, `VPStem` carries the root form. The reason for separating these three cases is that they cannot occur at the same time.

The label 'inf' stores the required form of the verb in that corresponding tense, whereas 'fin' stores the copula (auxiliary verb).

The label 'obj' on the other hand, stores the object of the verb and also the agreement information of the object. The label 'subj' stores information about transitivity of the verb with `VType`, which include: intransitive, transitive or di-transitive:

```
VType = VIntrans|VTrans|VDiTrans ;
```

The label 'comp' stores the complement of the verb. Notice that it also inflects in number, gender and person (with `Agr` defined on page 3), whereas the label 'ad' stores the adverb.

Finally, 'embComp' stores the embedded complement. It is used to deal with exceptions in the word order of Punjabi when making a clause. For instance, if a sentence or a question sentence is a complement of the verb then it takes a different position in the clause; i.e. it comes at very end of the clause as shown in the example with bold-face:

$oo_{\_she}$ $kehendi_{\_say}$ $ai_{\_Aux}$ $keh_{\_that}$ $main_{\_I}$ $roti_{\_bread}$ $khanda_{\_eat}$ $waN_{\_Aux}$

She says that I (masculine) eat bread.

On the contrary, if an adverb is used as a complement of verb then it comes before the main verb, as shown in the following example:

oo_she kehendi_say ai_Aux keh_that oo_she tez_briskly chaldi_walks ai_Aux
She says that she walks briskly

Construction: Lexical category verb (V) is converted to syntactic category verb phrase (VP) through different VP construction functions. The simplest is:

```
fun UseV : V → VP ;
lin  UseV v = predV v ;
```

The function predV converts the lexical category V to the syntactic category VP:

```
predV : Verb → VPH = \verb -> {
s = \\vh => case vh of {
  VPTense VPPres (Ag g n p) => {
     fin =copula CPresent n p g;
     inf =verb.s!VF Imperf p n g} ;
  VPTense VPPast (Ag g n p) => {
   fin = [] ;
   inf =verb.s!VF Perf p n g};
  VPTense VPFutr (Ag g n p) => {
   fin = copula CFuture n p g ;
   inf =  verb.s ! VF Subj p n g };
  VPTense VPPerf (Ag g n p) => {
    fin = [] ;
    inf = verb.s!Root ++ cka g n} ;
  VPStem => { fin = [] ;
      inf =  verb.s ! Root };
  _ => {fin = [] ;
        inf = verb.s!Root}};
 obj = {s = [] ; a = defaultAgr} ;
 vType = VIntrans ; ad   = [] ;
 embComp = [] ; comp = \\_ => []};
```

The lexical category V has three forms (corresponding to perfective/imperfective aspects and subjunctive mood). These forms are then used to make four forms (VPPres, VPPast, VPFutr, VPPerf in the above code) at the VP level, which are used to cover different combinations of tense, aspect and mood of Punjabi at clause level.

As an example, consider the explanation of the above code in bold-face. It builds a part of the inflection table represented by 's' for VPPres and all possible combination of gender, number and person (Ag g n p). As shown above, the imperfective form of lexical category V (VF Imperf p n g) is used to make present tense at VP-level. The main verb is stored in the field labeled as 'inf' and the corresponding auxiliary verb (copula) is stored in the label 'fin'.

All other parts of VP are initialized to default or empty values in the above code. These parts will be used to enrich the VP with other constituents, e.g. adverb, complement etc. This is done in other VP construction functions including but not limited to:

Want to run, dʊɽna_run ʧahna_want
```
ComplVV : VV → VP → VP;
```

Sleep here, ai:ʈʰe:_here sʊna_sleep
```
AdvVP       : VP → Adv → VP;
```

3.3. Adjectival Phrases

At morphological level, Punjabi adjectives inflect in number, gender and case (Humayoun and Ranta, 2010). At syntax level, they agree with the noun they modify using the agreement information of the NP. Adjectival phrase (AP) can be constructed simply from the lexical category adjective (A) through the following function:

```
PositA  : A  → AP ;  (Warm, gərəm)
```

Or from other categories such as:

```
Warmer than I, mi:re:_ɹ to:ɳ_than gərəm_warm
ComparA : A  → NP → AP ;
```

3.4. Adverbs and Closed Classes

The construction of Punjabi adverbs is very simple because "they are normally unmarked and don't inflect" (Humayoun and Ranta, 2010). We have different construction functions for Adverbs and other closed classes both at lexical and syntactical level. For instance, consider the construction of adverbs with two functions (but not limited to):

Warmly, gərəm dʒʊxi:
```
fun PositAdvAdj : A → Adv ;
```

Very quickly, boht_very ti:zi_quickly de nal_coupla
```
fun AdAdv  : AdA → Adv → Adv ;
```

3.5. Clauses

While a phrase is a single word or group of words, which are grammatically linked to each other, a clause on the other hand, is a single phrase or group of phrases.

Different types of phrases (e.g. NP, VP, etc) are grouped together to make clauses[6]. Clauses are then used to make sentences. In GF tense system the difference between a clause and a sentence is: A clause has a variable tense while a sentence has a fixed tense.

We first construct clauses and then just fix their tense in order to make sentences. The most important construction of a clause is:

```
PredVP : NP → VP → Cl; -- Ali walks
```

The clause (Cl) has the following type:

```
Clause : Type =
   {s : VPHTense => Polarity =>
Order =>Str} ;
```

Where:

```
VPHTense = VPGenPres|VPImpPast
|VPFut|VPContPres|VPContPast
|VPContFut|VPPerfPres|VPPerfPast
|VPPerfFut|VPPerfPresCont|VPSubj
|VPPerfPastCon|VPPerfFutCont ;
Polarity = Pos | Neg
Order = ODir | OQuest
```

The tense system of GF resource library covers only eight combinations with four tenses (present, past, future and conditional) and two anteriorities (Anter and Simul). It does not cover the full tense system of Punjabi, which is structured around the aspect and the tense/mood.

We make sentences in twelve different tenses (VPHTense in the above given code) at clause level to get a maximum coverage of the Punjabi tense system. Polarity is used to construct positive and negative, while Order is used to construct direct and question clauses.

We ensure the SOV agreement by saving all needed features in NP. These are made accessible in the PredVP function.

A distinguishing feature of Punjabi SOV agreement is ergative behavior where transitive perfective verb may agree with the direct object instead of the subject. Ergativity is ensured by selecting the agreement features and noun-form accordingly. We demonstrate this in the following simplified code segment:

```
subj agr : NPCase * Agr =
case vt of {
 VPImpPast  => case vp.subj of {
  VTrans    => <NPErg, vp.obj.a>;
  VDiTrans  => <NPErg, defaultAgr>;
   _         => <NPC Dir, np.a>}  ;
_     => <NPC Dir, np.a>}
```

For perfective aspect (VPImpPast), if the verb is transitive then it agrees with the object and therefore the ergative case of NP is used (VTrans in the above code).

For DiTransitive (i.e. VDiTrans in the above code) the agreement is set to default but the ergative case is still needed.

In all other cases, specified with the wild card "_" above, the agreement is made with the subject (np.a), and we use the direct case (i.e. NPC Dir).

After selecting the appropriate forms of each constituent (according to the agreement features) they are grouped together to form the clause. For instance, consider the following simplified code segment combining different constituents of a Punjabi clause:

```
np.s!subj ++ vp.obj.s ++ vp.ad ++
vp.comp!np.a  ++ nahim ++ vps.inf
++ vps.fin ++ vp.embComp;
```

Where:

(1) np.s!subj is the subject; (2) vp.obj.s is the object (if any); (3) vp.ad is the adverb (if any); (4) vp.comp!np.a is verb's complement; (5) nahim is the negative clause constant; (6) vps.inf is the verb; (7) vps.fin is the auxiliary verb; (8) vp.embComp is an embedded complement.

4. Coverage and Limitations

The grammar we have developed consists of 40 categories and 190 syntax functions. It covers only a fair enough part of the language. The reason for this limitation is approach of the common abstract syntax defined for all the languages in the GF resource library. Indeed it is not possible to have an abstract syntax, which is common to, and covers all features of all languages. Consequently, the current grammar does not cover all aspects of Punjabi.

However, this does not put any limitation on the extension of a language resource. It can be extended by implementing language specific features as extra language-specific modules. However these features will not be accessible

[6]Verb phrases alone can also be used as clause some times.

through the common API, but can be accessed in the Punjabi application grammars.

5. Evaluation and Future Work

It is important to note that completeness is not the success criteria for this kind of grammar based resource but accuracy is (Ranta 2009b). Evaluating a resource grammar is just like evaluating a software library in general. However, this type of evaluation is different from evaluation of a natural language processing application in general, where testing is normally done against some corpus. To evaluate the accuracy, we use the Punjabi resource grammar to translate, and observe, a test suite of examples[7] from English to Punjabi and vice versa. We achieved an accuracy of 98.1%. The reason for not having 100% accuracy is that our current grammar does not cover all aspects of the language. One such aspect is compound verbs of Punjabi, formed by nouns and the auxiliary verb 'to be' (*hona:*). In this case, its gender must agree with the inherent gender of the noun. We have not yet covered this agreement for compound verbs and therefore, produce incorrect translations. An interesting (yet wrong) example would be:

barif honda pe:a ae: (It is raining)
Instead of "honda pi:a", it should be "hondi: pai:"

Another such feature is the repetitive use of verb in Punjabi (e.g. mʊnɖa_boy ru:ndɛ:_weping ru:ndɛ:_weping su:ɳ_slept gi:a_coupla, مُنڈا روندے روندے سوں گیا, the boy slept weeping). Coverage of such language specific details is one direction for the future work.

6. Related Work and Conclusion

In general language resources for Punjabi are very limited; especially for the one spoken in Pakistan and written in *Shahmukhi*. Furthermore, most of the applications related to Punjabi are designed only for the Punjabi, written and spoken in India; hence, only support the *Gurmukhi* script. A review of such applications is given in (Lehal, 2009).

There are some attempts to interchange between these scripts with transliteration systems. However, the current systems only seem to provide partial solutions, mainly because of the vocabulary differences (Humayoun and Ranta, 2010).

A transfer-based machine translation system reported in (Lehal, 2009) translates between Punjabi and Hindi only. On the contrary, the Punjabi resource grammar is based on Interlingua approach, which makes it possible to translate between seventeen languages in parallel. With the best of our knowledge this work is the first attempt to implement a computational Punjabi grammar as open source.

We have described the implementation of the computational grammar for Punjabi. It might be a useful resource, and may encourage other researchers to work in this direction.

As the resource grammar does not cover full features of Punjabi, although it is not possible to use it for parsing and translation of arbitrary text, it is best suited for building domain specific application grammars.

[7] See (Bringert et el, 2011) for this test suite of examples.

References

B. Bringert, T. Hallgren, A. Ranta. 2011. *GF Resource Grammar Library Synopsis*. www.grammaticalframework.org/lib/doc/synopsis.html.

M. Butt, H. Dyvik, T. H. King, H. Masuichi, C. Rohrer. 2002. *The Parallel Grammar Project*. In Proceedings of COLING-2002. Workshop on Grammar Engineering and Evaluation.

M. Humayoun and A. Ranta. 2010. *Developing Punjabi* Morphology, *Corpus and Lexicon*. The 24th Pacific Asia conference on Language, Information and Computation. pp: 163-172.

G. S. Lehal. 2009. *A Survey of the State of the Art in* Punjabi *Language Processing*, Language In India, Volume 9, No. 10, pp. 9-23.

A. Ranta. 2004. *Grammatical Framework: A Type-*Theoretical *Grammar Formalism*. Journal of Functional Programming, 14(2), pp. 145-189.

A. Ranta. 2009a. *Grammatical Framework: A Multilingual Grammar Formalism, Language and Linguistics Compass*, Vol. 3.

A. Ranta. 2009b. *Grammars as Software Libraries*. In Y. Bertot, G. Huet, J-J. Lévy, and G. Plotkin (eds.), *From Semantics to Computer Science*, Cambridge University Press, pp. 281-308.

M. K. Verma. 1974. *The Structure of the Noun Phrase in English and Hindi by Review* author(s): R. K. Barz, L. A. Schwarzschild Journal of the American Oriental Society, Vol. 94, No. 4, pp. 492-494.