FLAN for Chunking

“FLAN”

Generic Shell based on Finite State Transducer

[Manual - Version 0.53]

CONTENTS

"FLAN"

1. Generic Shell

1.1. Introduction

A generic shell "FLAN" based on FSTs has been developed at LTRC, IIIT, Hyderabad funded by Outside Echo. The purpose behind the design of this powerful shell is to create a "free" tool which can be used by people across the world to build morphological and other analyzers for their languages. The augmented FSTs can be put in a cascade where the output of one phase, feeds as input to the next phase. Such a feed is so designed that even the selected features can be transferred across. Such phases in the cascade enable the shell to be used for handling morphology as well as chunking and parsing.

1.2. Formalism of the Shell

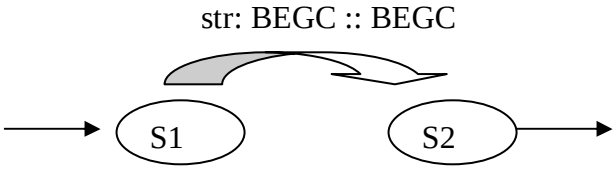
The formalism adopted here is augmented transition transducer with conditions, actions and feature structures.

(i) FST

To introduce the augmented transition transducer, we begin with finite state transducers (FSTs). A FST has a set of states and arcs between them. The arcs are labeled by input symbols and possibly output symbols. One of the states is called the starting state. During a typical operation of the machine, it makes a transition from the current state to a new state by consuming a token from the given input string (of tokens) provided the input token matches the input symbol labeled on the arc. The output symbol on the arc is sent to output buffer.

The machine completes successfully if the machine is in an accepting state when the input string is fully consumed. Thus, the machine always begins its operation in the starting state and if on consumption of the input string, it ends in an accepting state, it is said to complete successfully. On successful completion, the output buffer is actually sent to the output.

Example:

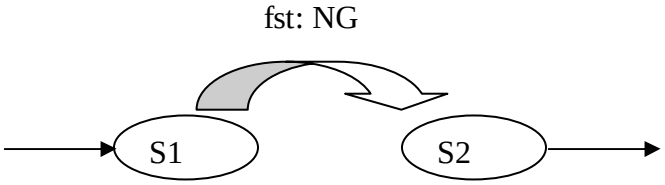


Here, the string 'BEGC' itself is the input, which produces 'BEGC' as output. The symbol '::' indicates the output to be produced. Absence of the symbol '::' means null output for the arc, in which case the input is consumed but there is no output.

(ii) Recursive FSTs

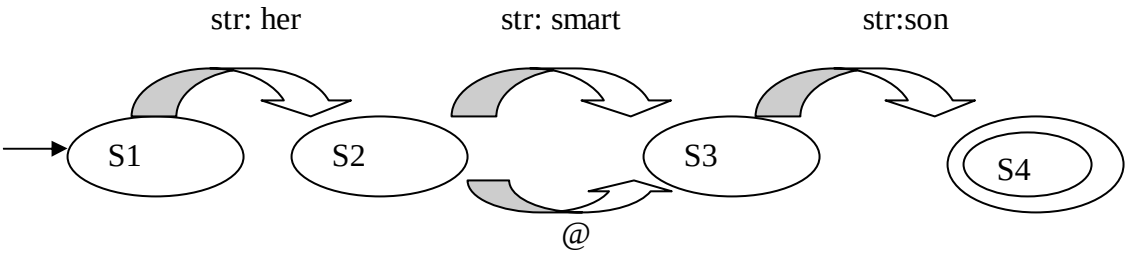
Recursive FSTs are an enrichment of FSTs. They have all the properties described above, and in addition allow the following: Each FST has a name and the input symbol on the arc can be labeled with not just input tokens but by names of other FSTs. When an arc with such a label is attempted to be traversed by a machine in operation, the named FST is operated from its starting state (thus consuming possibly many tokens). The arc can be successfully taken only if the named machine completes successfully.

Example:



Here, a finite state machine named NG is called which is:

fst: NG

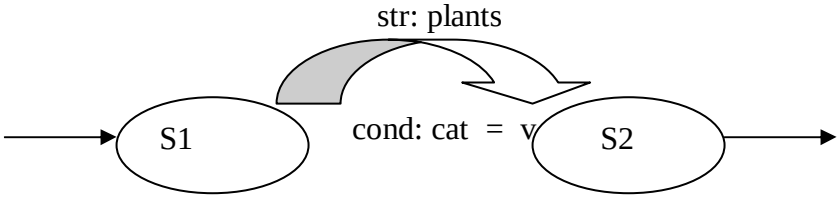


An arc is labelled @ to indicate zero suffix or to indicate a skip of that particular transition.

(iii) Augmented FST

In augmented FSTs, arcs are also permitted to have conditions and actions on feature structures. A feature structure is associated with every machine when it starts its operation. Values of features can be set by actions, and tested by conditions. An arc can be traversed if the conditions associated with the arc are satisfied besides the matching of the input symbol.

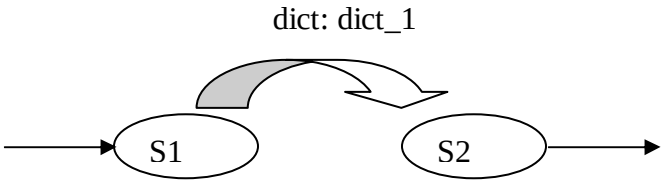
Example:



Here, if the input string satisfies the condition of belonging to the category verb, then it is successfully consumed.

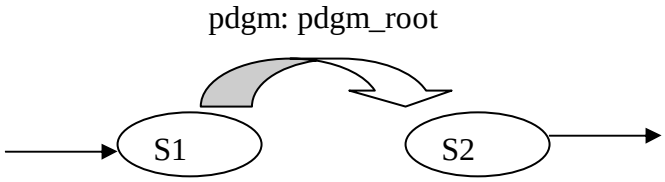
Traversing through an arc may also require a dictionary and a paradigm look-up especially, when the shell is being used for morphological decomposition. This is represented as shown below where the dictionary specified 'dict_1' is looked up and matched with the input string.

Example:

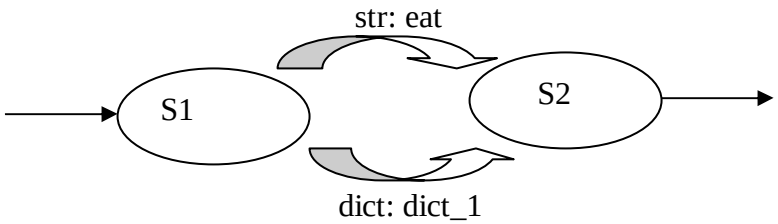


Here, a dictionary is called for look up of the data and the name of the dictionary is dict_1.

Similarly, the paradigm arc looks up paradigm tables with the mentioned name.
Example:



The machine is called non-deterministic if for an input token it is possible to traverse more than one arc from a state. In such a case, more than one analysis is possible and the machine may produce more than one (alternative) output for the same input string.



2. Chunking

2.1. Using the Shell for Chunking

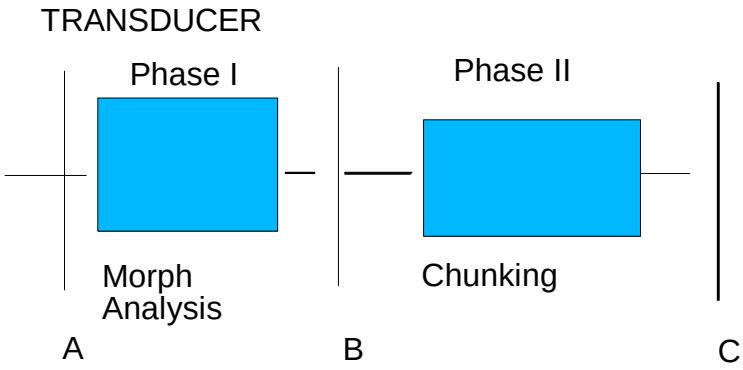
The shell allows a series of transducers to be put together where the output of one phase becomes the input for the next phase. The advantage of multiple phases is that the language analysis can be done modularly. In this section, the way the generic shell can be used for chunking is described. The shell together with language data will take a tagged sentence as input and mark the noun and verb groups.

The current model is developed using English and Hindi sample data. As the shell consists of augmented transition transducers (like ATNs) with conditions, actions and feature structures, the linguist has to provide data in terms of such transducers for his language.

The standard format for input and output is:

String<feature structure>separator

where the feature structure may be optional and a separator can be a space (default separator), nil or anything else.



A = input string i.e. a token or a sequence of tokens
B = sequence of tokens, each with a feature
C = noun and verb grouping

Fig. 1

This can be illustrated with examples as shown below. The input sentence is analysed and tagged in the first phase and then the tagged sentence serves as the input for the chunking phase.

English
"Her smart brother goes to a posh school"

INPUT:
Her<root=she/cat=pron/g=f/n=s/p=3> smart<root=smart/cat=adj>
brother<root=brother/cat=noun/g=m/n=s/p=3> goes<root=go/cat=verb/tense=pres>
to<root=to/cat=prep> a<root=a/cat=det> posh<root=posh/cat=adj>
school<root=school/cat=noun>.

OUTPUT:
BEGC<cat=NG> Her<root=she/cat=pron/g=f/n=s/p=3> smart<root=smart/cat=adj>
brother <root=brother/cat=noun/g=m/n=s/p=3> ENDC<cat=NG> BEGC<cat=VG>

goes<root=go/cat=verb/tense=pres> ENDC<cat=VG> to<root=to/cat=prep>
BEGC<cat=NG> a<root=a/cat=det> posh<root=posh/cat=adj>
school<root=school/cat=noun> ENDC<cat=NG>

Hindi
"usakI kahAnI adhUrI hE"

INPUT:
usakI<root=vaha/cat=pron/case=poss/g=f/n=sg/p=3>
kahAnI<root=kahAnI/cat=noun/g=f/n=sg> adhUrI<root=adhUrA/cat=adj/g=f>
hE<root=honA/cat=verb/n=sg/p=3/tense=pres>

OUTPUT:
BEGC<cat=NG> usakI<root=vaha/cat=pron/case=poss/g=f/n=sg/p=3>
kahAnI<root=kahAnI/cat=noun/g=f/n=sg> ENDC<cat=NG>
adhUrI<root=adhUrA/cat=adj/g=f> BEGC<cat=VG>
hE<root=honA/cat=verb/n=sg/p=3/tense=pres> ENDC<cat=VG>

For implementing the above, we need at least two resources –

a. A generic shell which allows FSTs and other data to be defined.
b. Language specific data such as dictionaries and language specific FSTs.

2.2. Data Specifications for Chunking

Phase List
A sample phase.lst file containing information about the paths where the data for the multiple phases is stored has to be given. [Currently the information should be stored in FLAN_ver0.53/phases.lst]. If there is only one phase, then the path of that phase has to be given.

Standard input and output format:
The standard format for input and output is:

String<feature structure>separator

where the feature structure may be optional and a separator can be a space (default separator), nil or anything else.

Specifications:
For chunking the two most important components are:

a. Rules specifying noun and verb grouping
b. FSTs to represent these rules

As the shell uses string, dictionary, paradigm or another fst as input label, currently the rules may be given using these as the input label. For example, to represent the phrase “the smart boy” as a noun group, one can give the rule either as:

(i) noun group = the smart boy (i.e. by using “str”)
or
(ii) noun group = determiner adjective noun (i.e. by using “dict” where the dictionary specified contains the words of the given category)

The specifications are given for giving the following:
a. Dictionaries
c. Transition tables for FSTs

[These should be provided in the directory FLAN_ver0.53/FST/data]

2.2.1. Dictionaries

The format for the dictionaries is:
Word [TAB] <features>

For example: dict_V
eat <root=eat/ cat=V>

In the dictionary, the features are given as attribute value pairs separated by slash ‘/’ within angular brackets ‘<>’. Optionality is indicated by ‘|’.

The dictionary of a particular grammatical category contains a list of word forms (paradigms) with their feature structures, if any. The format is:

Word [TAB] Features

Examples
English: dict_verb

eat <root=eat/ cat=V>
eats <root=eat/ cat=V/tense=present>
ate <root=eat/ cat=V/tense=past>
eaten <root=eat/ cat=V/aspect=perfective>
eating <root=eat/ cat=V/aspect=progressive>

Hindi: dict_noun

laDakA <root=laDakA/cat=noun/n=sg>
laDake < root=laDakA/cat=noun/n=pl>

The dictionaries for all grammatical categories have to be provided following the specifications given. The format of the dictionaries mentioned in the fsts above would be:

a.dict_pron
she <root=she/cat=pron/g=f/n=sg/p=3/case=nom>
her <root=she/cat=pron/g=f/n=sg/p=3/case=acc>
her <root=she/cat=pron/g=f/n=sg/p=3/case=poss>
his <root=he/cat=pron/g=m/n=sg/p=3/case=poss>

b. dict_CD
one <root=one/cat=CD>

c. dict_PD
all <root=all/cat=PD>

d. dict_noun
houses <root=house/cat=noun/n=pl>
school <root=school/cat=noun//g=nt/n=sg>

e. dict_adj
smart <root=smart/cat=adj>
big <root=big/cat=adj>

f. dict_adv
very <root=very/cat=adv>
too <root=too/cat=adv>

g. dict_verb
have <root=have/cat=v/t=pres>
goes <root=go/cat=v/aspect=simple>
is <root=be/cat=v/tense=present>

h. dict_aux
have <root=have/cat=aux>
be <root=be/cat=aux>

i. dict_modal
may <root=may/cat=modal>
can <root=can/cat=modal>

j. dict_prep
of <root=of/cat=prep>
with <root=with/cat=prep>

k. dict_det
the <root=the/cat=det>

2.2.2. FSTs to represent Chunking rules

First, one has to write a grammar for one's language and then all possible constituents of noun and verb groups. All this has to be given in the form of transition tables. A transition table for a fst is illustrated below:

fst:main
S1
S1 S2
S2
S1 S2 fst:ENG-SYNTAX1 <> <>

The following is the format for the transition table.

The first line lists the initial state.
The second line lists all the states.
The third line lists the accepting states.
From the fourth line, the actual data is given using the following six fields separated by TABS:

SOURCE DESTINATION INPUT OUTPUT ACTION CONDITION

The last two fields should be given in angular brackets. If there is no output, then that field should be left blank.

For a given arc, the fields mean:

SOURCE = the state denoting the beginning of the arc
DESTINATION = the state denoting the end of the arc
INPUT = it can be a string, dictionary, list, paradigm or another fst
OUTPUT = it is a string with an optional feature structure
ACTION = setting any feature in the current machine
CONDITION = testing any feature in the current machine

The input labels can be any of the following:

str: xxx
fst: xxx
dict: xxx
pdgm: xxx

where ‘xxx’ is any name corresponding to the input label..

The output labels can be any of the strings(a,b,c) along with any of the feature structure (f1,f2,f3):

a. Any string (ex. a string named BEGC)
b. !attr = take the value of the attribute ‘attr’ from feature structure of the current arc
c. FS.attr = take the value of the attribute ‘attr’ from feature structure of current machine.

f1. <attr=value> ; the feature structure of the string
f2. <attr=value/*=!ALL> ; take the value to be printed from child machine, eg. dictionary
f3. <attr=value/*=FS.ALL> ; take the value to be printed from current machine

The field action can be represented by any of the following notation represented using an example:

a. g = f ; the feature structure is set in the current machine
b. g =!g ; the down_arrow symbol ‘!’ indicates that the value of the attribute is called either from a previous phase (in case of a string), or from a child machine (in case of a fst) or from the dictionary specified (in case of a dictionary)
c. * = !ALL ; the notation ‘*’ indicates that all attribute-value pairs have to be taken according to the down_arrow symbol ‘!’

Following these specifications, the chunking rules for English may be given as follows:

fst:main

```
S1
S1 S2
S2
S1    S2    fst:ENG-SYNTAX1 <>    <>    <>
```

fst:ENG-SYNTAX1

eg."Her smart brother goes to a posh school"
"One of my brothers is a truck driver"
%#NG=noun group, VG=verb group

```
S1
S1 S2 S3 S4 S5 S6 S7
S7
S1    S2    @          BEGC          <cat=NG>  <>
S2    S3    fst:NG                   <>    <>
S3    S4    fst:VG                   <>    <>
S4    S5    dict: dict_prep          <>    <>
S4    S5    @                        <>    <>
S5    S6    fst:NG                   <>    <>
S6    S7    @          ENDC          <cat=NG>  <>
```

fst: NG

eg. “one of those very big houses”/

```
S1
S1 S2 S3 S4 S5 S6 S7 S8
S8
S1    S2    dict:dict_CD          <>    <>
S1    S2    dict:dict_PD          <>    <>
S2    S3    dict:dict_prep        <>    <>
S2    S3    @                     <>    <>
S3    S4    dict:dict_det         <>    <>
S3    S4    dict:dict_pron        <>    <case=poss>
S3    S4    @                     <>    <>
S4    S5    dict:dict_adv         <>    <>
S4    S5    @                     <>    <>
S5    S6    dict:dict_adj         <>    <>
S5    S6    @                     <>    <>
S6    S7    dict:dict_noun        <>    <>
S6    S7    @                     <>    <>
S7    S8    dict:dict_noun        <>    <>
S7    S8    dict:dict_pron        <>    <>
```

fst:VG

```
S1
S1 S2
S2
S1    S2    fst:VG1 <>    <>    <>
```

fst:VG1

"ate"/"has eaten"/"had been written"/"should have been eating"/etc

```
S1
S1 S2 S3 S4 S5
S5
S1    S2    dict:dict_modal       <>    <>
S1    S2    @                     <>    <>
S2    S3    dict:dict_aux         <>    <>
S2    S3    @                     <>    <>
S3    S4    dict:dict_aux         <>    <>
```

NOTE: The strings BEGC and ENDC indicate the beginning of a chunk and ending of a chunk respectively. Giving them in the output field indicate that they must be output. Leaving the output field blank indicates that the input is consumed but there is no output.

More examples:

Example 1: *English*
"One of his brothers is a truck driver."

INPUT: one<root=one/cat=CD> of<root=of/cat=prep>
his<root=he/cat=pron/g=m/case=poss>
brothers<root=brother/cat=noun/g=m/n=pl> is<root=be/cat=verb/n=sg/tense=pres>
a<root=a/cat=det> truck<root=truck/cat=noun/n=sg> driver<root=driver/cat=noun/n=sg>.

OUTPUT:
BEGC<cat=NG> one<root=one/cat=CD> of<root=of/cat=prep>
his<root=he/cat=pron/g=m/case=poss> brothers<root=brother/cat=noun/g=m/n=pl>
ENDC<cat=NG> BEGC<cat=VG> is<root=be/cat=verb/n=sg/tense=pres>
ENDC<cat=VG> BEGC<cat=NG> a<root=a/cat=det> truck<root=truck/cat=noun/n=sg>
driver<root=driver/cat=noun/n=sg> ENDC<cat=NG>
.
Example 2: *Hindi*

"merA choTA beTA kala skUla se dhire se calA AyA thA"

INPUT:
merA<root=mEz/cat=pron/n=sg/p=1/case=poss> choTA<root=choTA/cat=adj/g=m>
beTA<root=beTA/cat=noun/g=m/n=sg/p=3> kala<root=kala/cat=noun>
skUla<root=skUla/cat=noun/n=sg> se<root=se/cat=prep>
dhire<root=dhire/cat=adv> se<root=se/cat=adv.p>
calA<root=calanA/cat=verb/aspect=perf/g=m/n=sg>
AyA<root=AnA/cat=verb/aspect=perf/g=m/n=sg>
thA<root=honA/tense=past/g=m/n=sg>

OUTPUT:
BEGC<cat=NG> merA<root=mEz/cat=pron/n=sg/p=1/case=poss>
choTA<root=choTA/cat=adj/g=m> beTA<root=beTA/cat=noun/g=m/n=sg/p=3>
ENDC<cat=NG> BEGC<cat=NG> kala<root=kala/cat=noun> ENDC<cat=NG>
skUla<root=skUla/cat=noun/n=sg> se<root=se/cat=prep> dhire<root=dhire/cat=adv>
se<root=se/cat=adv.p>
BEGC<cat=VG> calA<root=calanA/cat=verb/aspect=perf/g=m/n=sg>
AyA<root=AnA/cat=verb/aspect=perf/g=m/n=sg>
thA<root=honA/tense=past/g=m/n=sg> ENDC<cat=VG>.