Abstract dependency trees

Table of Contents

Introduction 1
ADT 1
- Introduction 1
- Category 3
- Lexical nodes 4
- Index nodes 5
- Reference nodes 6
- Punctuation 6
- Separable Verb Prefixes 6

ADT formats 6
- Prolog 6
- XML 7

How to obtain ADT structures 8
- ADT as Alpino output format 8
- ADT from DT 8

Bibliography 8

Introduction

This document provides a description of the format of dependency structures that are used as the input for the Alpino chart generator. Dependency structures describe grammatical dependency relations between lexical items, and the constituents dominating over lexical items. Since dependency structures for generation can contain less information than dependency trees that are produced as a side effect of parsing, we call them abstract dependency trees (ADTs).

While different input formalisms have been proposed for sentence realization in the past, such as minimal recursion semantics (MRS), we have chosen to use a different input format that describes the grammatical dependencies of the to-be generated sentence. The rationale for this format is:

- ADTs can be derived from (Alpino) dependency trees with ease.
- Most other input formalisms would require rather extensive changes to the lexicon and grammar.
- Prior work with Alpino dependency trees has shown that (abstract) dependency trees provide sufficient abstraction for tasks where a generation component is desired.

This document describes the format of ADTs, including the representation of ADTs as Prolog terms and XML documents. The procedure for deriving an ADT from a normal Alpino dependency tree is also described.

ADT is an abstraction of the dependency structure format of Alpino (and the Lassy corpora). The Lassy Annotation Manual is therefore required in combination with the present document in order to understand the representation in full detail.

ADT

Introduction

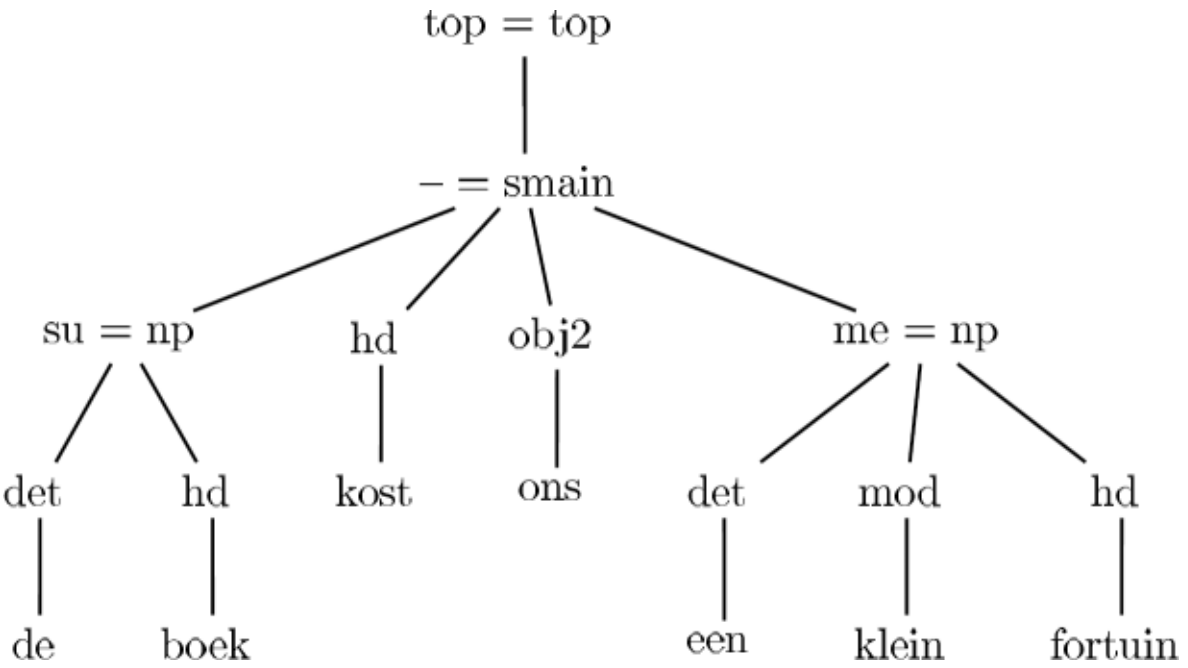
An abstract dependency is a directed acyclic graph that models the grammatical relations between lexical items and categories built from lexical items. The generator creates realizations for abstract

dependency trees that describe at least one possible sentence. An abstract dependency tree can consist of four node types:

- Category (interior) nodes.
- Lexical (leaf) nodes.
- Indexed nodes.
- Reference nodes.

Edges between nodes have a dependency label, where *hd* indicates the head of a grammatical relation. Figure Figure 1, “Dependency tree for "De boeken kosten ons een klein fortuin"” shows a dependency tree for the sentence "De boeken kosten ons een klein fortuin". For interior nodes, the dependency label is shown as the first element, and the category as the second.

Figure 1. Dependency tree for "De boeken kosten ons een klein fortuin"



Possible dependency labels are listed in Table 1, “Dependency labels”. In the following sections, the node types will be described in more detail.

Table 1. Dependency labels

Dependency label	Description
app	apposition
body	body (with complementizer)
cmp	complementizer
cnj	conjunct
crd	coordinator
det	determiner
dlink	discourse-link
dp	discourse-part
hd	head

Dependency label	Description
hdf	closing element of a circumposition
ld	locative or directional complement
me	measure phrase complement
mod	modifier
mwp	part of a multi-word-unit
nucl	nucleus discourse unit
obcomp	object of comparative
obj1	direct object
obj2	secondary object (indirect object, . . .)
pc	prepositional complement
pobj1	provisional direct object
predc	predicative complement
predm	predicative modifier
rhd	head of a relative clause
sat	satellite discourse unit
se	inherently reflexive complement
su	subject
sup	provisional subject
svp	separable verb particle
tag	discourse tag
vc	verbal complement
whd	head of wh-question

Category

All nodes are associated with a category. All possible categories are listed in table Table 2, “Category labels”. Note that this is different from Alpino dependency structures, in which only interior nodes are associated with a category.

Table 2. Category labels

Category label	Description
ap	adjective phrase
advp	adverb phrase
ahi	*aan het*-infinitive group
conj	conjunction
cp	phrase started by a subordinating conjunction
detp	word group with a determiner as the head
du	discourse unit
inf	bare infinitive group
np	noun phrase
oti	*om te*-infinitive-group
ppart	passive/perfect participle
pp	prepositional phrase

Category label	Description
ppres	present participle group
rel	relative clause
smain	declarative sentence (verb at the second position)
ssub	Subordinate clause (verb final)
svan	*van*-sentence
sv1	verb-initial sentence (yes/no question, imperatives)
ti	*te*-infinitive group
whrel	relative clause with embedded antecedent
whsub	embedded question
whq	WH-question

Lexical nodes

Lexical nodes are leaf nodes that provide an abstracted representation for (surface) words. As a minimum a lexical node should specify:

- The word sense. The sense of a word is the root of a word, possibly with additional information to select for a specific reading.
- An Alpino part of speech tag.
- A set of attribute/value pairs.

Part of speech tags and possible attributes are discussed in more detail below.

Alpino part of speech tags

Table Table 3, “POS tags” lists all possible Alpino part of speech tags for lexical items.

Table 3. POS tags

Tag	Meaning
adj	Adjective
adv	Adverb
comp	Complementizer
comparative	Comparative
det	Determiner
fixed	Fixed part of a fixed expression
name	Name
noun	Noun
num	Number
part	Particle
pron	Pronoun
prep	Preposition
punct	Punctuation
verb	Verb
vg	Conjunction

Attributes

In addition to the Alpino part of speech tag, some additional information is normally required to determine the semantics embodied by an ADT. In particular, the number for nouns, and the tense and inflection for verbs should be known.

- The number of a noun may be specified with the *rnum* attribute, which can have one of the following values: *sg*, *pl*. If the attribute is not specified, the generator will attempt to produce both singular and plural variants for this noun.
- The tense of the (finite) verb should be specified with the *tense* attribute, which can have one of the following values: *present*, *past*, *subjunctive*. If the attribute is not specified, the generator will attempt to produce both present and past tense, but not subjunctive.
- The sentence type can be specified using the attribute *stype* associated with the head of the sentence (the finite verb). Possible values are *ynquestion*, *whquestion*, *declarative*, *imparative*, *topic_drop*. If the attribute is not specified, the generator will attempt to generate any sentence type.
- Pronominals can be associated with the attributes *per*, *def* and *refl* to indicate person, reduction, definiteness and reflexiveness. The attribute *def* has values *def*, *indef*. The attribute *per* has values *fir*, *je*, *thi*, *u*, *u_thi*. The attribute *refl* has a single possible value *refl*. The attribute *wk* has a single attribute *yes* to indicate that the pronominal is in reduced form (as in *we* versus *wij*, *me* versus *mij*, etc.). If the attributes are not given, the generator will attempt to generate all consistent pronominals.
- Adjectives can be associated with the attribute *aform* with values *base*, *compar* and *super* to differentiate neutral, comparative and superlative adjectives. If the attribute is not given, the generator will attempt to generate with any of the forms.
- The attribute *pron=true* can be associated with determiners to distinguish, for example, *iedere* and *ieders*. If the attribute is not specified, the generator will only use lexical entries which do not have *pron=true*.
- Names can be associated with the attribute *neclass* with values 'LOC', 'PER', 'ORG', and 'MISC' . Unfortunately, the single quotes are currently part of the value.
- The attribute *personalized=true* is used to distinguish nominalized adjectives as in "de snelste" versus "de snelsten". If the attribute is not given, then only lexical entries are used by the generator which do not have *personalized=true*.
- The attribute *iets=true* is associated with adjectives to distinguish "iets lekkers" from "lekkere iets". If the attribute is not given, then only lexical entries are used by the generator which do not have *iets=true*.

Currently, the category, relation name, and postag must be specified for any category.

Some attributes need to be specified in the input, in order that Alpino will allow corresponding lexical entries to be considered for generation. These attributes are:

```
iets
personalized
pron
```

TODO: table of attributes with allowed values

TODO: table of attributes with default values

Index nodes

Indexed nodes are nodes associated with a specific index so that the node can be referred to by a reference node. For instance, in the sentence *Ik heb de trein gemist* both *ik-heb* and *ik-gemist* have a

subject-head relation, while *gemist* is the head of a *vc* of *heb*. To allow for such representations, co-indexation is required.

Reference nodes

Reference nodes only have an index, and no additional content nor sub-structure. The additional content can be found at the Index node with the same index.

Punctuation

The current assumption is, that no punctuation is specified in the ADT. The generator will add a minimum amount of obligatory punctuation for a given ADT.

Separable Verb Prefixes

To allow for an abstract description of verbs that have a separable particle, it is allowed to omit particles with the *svp* relation in an ADT. If such particles are not included as nodes in the ADT, the generator will attempt both to generate sentences with a separate particle as in *ik bel hem op* and a sentences in which the particle is not separated as in *omdat ik hem opbel*.

ADT formats

Prolog

An ADT can be stored as a Prolog term that consists of recursive *tree* terms. The basic format is:

```
tree(Relation,Daughters)
```

Where *Relation* is a relation term, and Daughters is a list of daughter nodes, or the empty list for leaf nodes. A relation has the following form:

```
r(Type,Label)
```

Type indicates the type of relation, such as *su*, *obj1*, or *mod*. The label comes in four types: interior nodes, leaf nodes, index nodes and reference nodes. An interior node uses a *p/1* term, for instance,

```
tree(r(vc,p(ppart)),[...])
```

is a node of the category *ppart* with a *vc* relation.

Lexical nodes use a *adt_lex/5* term as their label:

```
adt_lex(Cat,Root,Sense,PosTag,Attributes)
```

Here the category, root, sense, POS tag, and attributes of a lexical item are noted. The sense of a lexical item can contain additional information about a lexical item to select for a specific reading. For instance, for words with fixed parts, the fixed parts are often listed in the sense. E.g. the sense of *rood aanlopen* is *rood-loop_aan*. The sense can be omitted by replacing it with a variable (e.g., _).

This is an example of a lexical node:

```
tree(r(hd,adt_lex(np,trein,trein,noun,[rnum=sg])),[])
```

This describes a noun with the root *trein* and an attribute (*rnum=sg*) with the head (*hd*) relation.

An index node is represented as:

```
i(Number,Label)
```

where Label is an interior node or a leaf node.

Reference nodes are represented by

```
i(Number)
```

For instance, we can refer to

```
i(1,adt_lex(ik,ik,pron,[per=fir,rnum=sg,def=def]))
```

with:

```
i(1)
```

Combined, we can construct ADTs as Prolog terms for all grammatical sentences. E.g., the ADT term for the sample discussed above is:

```
tree(r(top,p(top)),[
 tree(r(--,p(smain)),[
  tree(r(su,i(1,adt_lex(ik,ik,pron,
    [per=fir,rnum=sg,def=def]))),[]),
  tree(r(hd,adt_lex(heb,heb,verb,
    [stype=declarative,tense=present])),[]),
  tree(r(vc,p(ppart)),[
   tree(r(su,i(1)),[]),tree(r(obj1,p(np)),[
    tree(r(det,adt_lex(de,de,det,[])),[]),
    tree(r(hd,adt_lex(trein,trein,noun,[rnum=sg])),[])
   ]),
   tree(r(hd,adt_lex(mis,mis,verb,[])),[])
  ])
 ])
])
```

XML

ADTs can also be represented as XML documents to allow for easy querying and manipulation outside the Alpino environment.

The dependency tree is represented in XML as a recursive structure of *node* elements. Each node has an identifier (*id*) and relation (*rel*). Category nodes have a *cat* attribute that specifies the category. For instance:

```
<node cat="ppart" id="4" rel="vc">
 ...
</node>
```

Every lexical node has the *root*, *sense*, and *postag* attributes for respectively describing the word root, sense, and part of speech tag. E.g.:

```
<node gen="de" id="8" num="sg" cat="np" pos="noun" rel="hd" root="trein"
  sense="trein" type="adt_lex"/>
```

Lexical nodes have other attributes, as described in the previous sections. For instace, here the *num* and *gen* attributes are also listed for the noun number and gender.

A node can be co-indexed by adding the *index* attribute to a lexical node:

```
<node id="2" index="1" root="ik" sense="ik" [...] />
```

The referring node also contains an *index* attribute, but no other information specific to a lexical node. For example:

```
<node id="5" index="1" rel="su"/>
```

As an example of a full ADT, we include an ADT for the same sentence used in the Prolog ADT:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_adt version="1.3">
  <node cat="top" id="0" rel="top">
    <node cat="smain" id="1" rel="--">
      <node def="def" id="2" index="1" rnum="sg" cat="np"
        per="fir" pos="pron" rel="su" root="ik" sense="ik"/>
      <node id="3" pos="verb" rel="hd" root="heb" sense="heb" tense="present"/>
      <node cat="ppart" id="4" rel="vc">
        <node id="5" index="1" rel="su"/>
        <node cat="np" id="6" rel="obj1">
          <node id="7" pos="det" rel="det" root="de"
            sense="de"/>
          <node id="8" rnum="sg" pos="noun" rel="hd" root="trein" sense="trein"/>
        </node>
        <node id="9" pos="verb" rel="hd" root="mis" sense="mis"/>
      </node>
    </node>
  </node>
</alpino_adt>
```

How to obtain ADT structures

There are two approaches. First, you can take a dependency structure, as produced by Alpino, and convert it to a ADT (using Alpino). Second, you can parse sentences with Alpino, and request an ADT as output. The format of the ADT is either Prolog or XML - as explained in a previous section.

ADT as Alpino output format

The Alpino option *end_hook=adt_prolog* will generate an ADT in Prolog format for every parse.

Likewise, the Alpino option *end_hook=adt_xml* will generate an ADT in XML format.

As a special case, the option *end_hook=gen_suite(best_score)* can be used to generate an ADT for the parse which resembles most the gold standard parse for the input sentence (obviously this assumes that we are parsing sentences for which the gold standard parse is available). The ADT uses the Prolog format in this case.

ADT from DT

This is currently not functioning properly. Perhaps this is not such a good idea anyway.

Bibliography

[lassyann] Gertjan van Noord, Ineke Schuurman, and Gosse Bouma, Lassy Syntactische Annotatie, http://www.let.rug.nl/vannoord/Lassy/sa-man_lassy.pdf