StackNN Documentation

Release 1.0

William Merrill, Yiding Hao, Robert Frank, Dana Angluin, Noah Am

Feb 09, 2019

Contents:

CHAPTER 1

formalisms package

1.1 Submodules

1.2 formalisms.buta_example module

1.3 formalisms.cfg module

Utility module defining various context-free grammars.

Example usage: CFGTask(**formalisms.cfg.dyck_task_parameters).run_experiment()

formalisms.cfg.**unambig_agreement_grammar = <MagicMock name='mock.CFG.fromstring()' id='139**
XOR Grammars

1.4 formalisms.depth_generate module

Generate a list of random sentences of a given derivation depth from a context-free grammar. Code by Dana Angluin. The main function is random_sentences.

formalisms.depth_generate.**all_lhs_from_grammar**(*gr*)
Finds the left-hand sides of all productions in a context-free grammar. The return value may contain duplicates.

Parameters **gr** (*CFG*) – A context-free grammar

Return type list

Returns All the nonterminals appearing in the left-hand side of a production of gr

formalisms.depth_generate.**all_not_in**(*lst1*, *lst2*)
Determines whether a list contains an element of another list.

Parameters

- **lst1** (*list*) – A list
- **lst2** (*list*) – Another list

Return type bool

Returns True if no element of lst1 is in list2, False otherwise

formalisms.depth_generate.**choose_production**(*nt*, *depth*, *table*, *gr*)

Randomly chooses a production with a given left-hand side nt. The probability of each production p is the proportion of strings generable from nt with derivations of at most a given depth that have derivations invoking p in the first step.

Parameters

- **nt** (*nltk.grammar.Nonterminal*) – The left-hand side of the production chosen
- **depth** (*int*) – The maximum depth of derivations considered
- **table** (*dict*) – A table computed by make_table (see make_table and count_production_depth)
- **gr** (*CFG*) – The context-free grammar from which the productions are drawn

Return type nltk.grammar.Production

Returns The chosen production

formalisms.depth_generate.**count_nonterminal_depth**(*nonterminal*, *depth*, *table*, *gr*)

A helper function for make_table. This function computes the number of terminal strings generable from a nonterminal using a derivation of at most a given depth. This function assumes that the number for the previous depth has already been computed.

Parameters

- **nonterminal** (*nltk.grammar.Nonterminal*) – The nonterminal from which generable strings are considered
- **depth** (*int*) – The maximum depth of derivations considered
- **table** (*dict*) – A table containing the results obtained from this function for the previous depth. The format of this table is the same as the return value of make_table
- **gr** (*CFG*) – The grammar whose generable strings are being considered

Return type int

Returns The number of strings generable by gr from nonterminal using derivations at most depth-many layers deep

formalisms.depth_generate.**count_production_depth**(*prod*, *depth*, *table*, *gr*)

A helper function for make_table. This function computes the number of terminal strings generable using a derivation of at most a given depth that invokes a given production as its first step. This function assumes that the number for the previous depth has already been computed.

Parameters

- **prod** (*nltk.grammar.Production*) – The production invoked during the first step of the derivations considered
- **depth** (*int*) – The maximum depth of derivations considered
- **table** (*dict*) – A table containing the results obtained from this function for the previous depth. The format of this table is the same as the return value of make_table
- **gr** (*CFG*) – The grammar whose generable strings are being considered

Return type int

Returns The number of strings generable by gr using derivations at most depth-many layers deep using prod as their first step

formalisms.depth_generate.**is_terminal_production**(*prod*, *gr*)

Determins whether or not a rule contains nonterminals in its right- hand side.

Parameters

- **prod** (*nltk.grammar.Production*) – A production
- **gr** (*CFG*) – A CFG

Return type bool

Returns True if there are no nonterminals in the right-hand side of prod, False otherwise

formalisms.depth_generate.**make_table**(*depth*, *gr*)

For each production p of a context-free grammar and each number k, this function computes the number of terminal strings whose derivations

- invoke the production p as the first step and
- have depth at most k.

Parameters

- **depth** (*int*) – The maximum possible value of k (see above)
- **gr** (*CFG*) – A context-free grammar

Return type dict

Returns For each production p and number k, the return dict maps the tuple (p, k) to the number described above

formalisms.depth_generate.**nonterminals_from_grammar**(*gr*)

Finds the left-hand sides of all productions in a context-free grammar. The return value does not contain duplicates.

Parameters **gr** (*CFG*) – A context-free grammar

Return type list

Returns All the nonterminals appearing in the left-hand side of a production of gr

formalisms.depth_generate.**random_from_form**(*form*, *depth*, *table*, *nonterminals*, *gr*)

Generates a random terminal string generable from a list of terminals and nonterminals using a derivation of at most a given depth.

Parameters

- **form** (*list*) – A list of terminals and nonterminals, from which the return value is derived
- **depth** (*int*) – The maximum depth of derivations considered
- **table** (*dict*) – A table computed by make_table (see make_table and count_production_depth)
- **nonterminals** (*list*) – Only nonterminals appearing in this list will be expanded
- **gr** (*CFG*) – A context-free grammar

Return type list

Returns The generated terminal string, in sentence format

formalisms.depth_generate.**random_sentences**(*count*, *depth*, *gr*)
Generates a number of random sentences using derivations of at most a given depth.

Parameters

- **count** (*int*) – The number of sentences to generate
- **depth** (*int*) – The maximum derivation depth of a generated sentence
- **gr** (*CFG*) – A context-free grammar to generate from

Type list

Returns The generated sentences

formalisms.depth_generate.**remove_duplicates**(*lst*)
Removes duplicates from a given list.

Parameters **lst** (*list*) – A list

Return type list

Returns lst, but with duplicates removed

formalisms.depth_generate.**select_from_dist**(*prob*)
This function chooses a random number according to a given probability distribution.

Parameters **prob** (*list*) – A probability distribution represented as a list. For each number i, prob[i] is the probability that this function returns i. For example, if [.2, .3, .5] is passed to this parameter, then there is a 20% chance of returning 0, a 30% chance of returning 1, and a 50% chance of returning 2.

Return type int

Returns The number chosen

1.5 formalisms.generate_tests module

Generate a list of random sentences of a given derivation depth from a context-free grammar. Code by Dana Angluin. The main function is random_sentences.

formalisms.generate_tests.**random_cfg_test**(*count*, *depth*, *gr*, *savepath*)
Calls random_sentences to generate a number of random sentences with derivations of depth at most the given amount from the grammar gr, and saves them as a test file in savepath. Format: each line is input,output where input is the generated string and output is the generated string with first symbol removed.

1.6 formalisms.tree_automata module

Classes for various kinds of tree automata. A tree automaton consists of a list of transitions, a set of final states, and in the case of a top- down tree automaton, an initial state. Throughout this module, automaton states are represented as Nonterminal objects from nltk.grammar, while tree labels are represented as unicode strings. State transitions are represented as Production objects from nltk.grammar; see check_is_transition and BUTA for more details.

For more information about tree automata in general, please see the TATA book by Comon et al.: http://tata.gforge.inria.fr/

class formalisms.tree_automata.**BUTA**(*transitions*, *finals*)
Bases: object

A non-deterministic bottom-up tree automaton (BUTA). A BUTA reads a tree from bottom to top. Each node of the tree is assigned a state based on its label and the states assigned to its children. The transitions of a BUTA are represented as Production objects of the following form:

Q -> "a" Q1 Q2 ... Qn.

The interpretation of a transition is that a node labelled "a" is assigned state Q if its n-many children are assigned states Q1, Q2, ..., Qn, respectively.

static fromstring(*transitions*, **finals*)

Constructs a BUTA from a string representation of the transitions.

Parameters

- **transitions** (*str*) – The transitions of the tree automaton, in string representation
- **finals** (*unicode*) – The accept states of the tree automaton

Return type *BUTA*

Returns The BUTA described by the parameters

generate(*states=None*, *depth=10*, *n=None*)

Generates all trees up to a certain depth or number that are assigned one or more given states by this BUTA.

Parameters

- **states** (*set*) – The root node of every tree generated is assigned a state from this set by this BUTA. If states is not specified, it will be the set of final states of this BUTA by default
- **depth** (*int*) – The maximum height of a generated tree
- **n** (*int*) – The maximum number of trees to generate

Return type generator

Returns A generator that produces every tree satisfying the above description

parse(*tree*)

Given a tree, this function computes the states assigned to each node of the tree (i.e., the "parse" of the tree). If this BUTA is nondeterministic, a tree may have more than one parse.

Parameters **tree** (*Tree*) – A tree

Return type generator

Returns The possible parses of tree according to this BUTA. Each parse is represented as a tree in which each node is labelled with its state according to this BUTA

recognize(*tree*)

Checks whether or not a tree is accepted by this BUTA.

Parameters **tree** (*Tree*) – A tree

Return type bool

Returns True if this BUTA accepts tree; False otherwise

transitions(*lhs=None*, *label=None*)

Public accessor for self._transitions.

Parameters

- **lhs** (*gr.Nonterminal*) – If lhs is not set to None, then only transitions with lhs on the left-hand side will be returned

- **label** (*unicode*) – If label is set to none, then only transitions whose right-hand sides begin with label will be returned

Return type set

Returns The set of transitions with the specified lhs and label

formalisms.tree_automata.**check_is_nonterminal**(**nts*)
Asserts that all of one or more objects are Nonterminals.

Parameters **nts** – An object, which may or may not be a Nonterminal

Returns None

formalisms.tree_automata.**check_is_transition**(**ps*)
Asserts that all of one or more Productions are transitions.

Parameters **ps** (*Production*) – One or more Productions

Returns None

formalisms.tree_automata.**check_type**(*obj*, *t*)
Asserts that an object has a certain type.

Parameters

- **obj** – An object
- **t** (*Type*) – A type

Returns None

formalisms.tree_automata.**is_transition**(*p*)
Checks to see if a Production object is a transition. A transition is a Production in which the right-hand side must begin with a terminal. See BUTA for the interpretation of a transition.

Parameters **p** (*gr.Production*) – A production

Return type bool

Returns True if p is a transition, False otherwise

1.7 formalisms.trees module

Helper functions for working with trees.

formalisms.trees.**get_root_label**(*tree*)
Finds the label of the root node of a tree.

Parameters **tree** – A tree

Returns The label of the root node of tree

formalisms.trees.**polish**(*tree*)
Computes the Polish representation of a tree.

Parameters **tree** (*Tree*) – A tree

Return type list

Returns The Polish representation of tree

formalisms.trees.**reverse_polish**(*tree*)
Computes the reverse-Polish representation of a tree.

Parameters **tree** (*Tree*) – A tree

Return type list

Returns The reverse-Polish representation of tree

1.8 Module contents

CHAPTER 2

models package

CHAPTER 3

structs package

CHAPTER 4

tasks package

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index