

StableMotifs Documentation

Contents

Module StableMotifs	3
Sub-modules	3
Module StableMotifs.Attractor	3
Classes	3
Class Attractor	3
Parameters	3
Attributes	3
Methods	4
Module StableMotifs.AttractorRepertoire	4
Classes	4
Class AttractorRepertoire	4
Attributes	4
Static methods	5
Methods	5
Module StableMotifs.DomainOfInfluence	7
Functions	7
Function GRASP	7
Function GRASP_default_scores	7
Function all_drivers_of_size	7
Function construct_GRASP_solution	8
Function fixed_excludes_implicant	8
Function fixed_implies_implicant	9
Function initial_GRASP_candidates	9
Function internal_drivers	9
Function knock_to_partial_state	10
Function local_GRASP_reduction	10
Function logical_domain_of_influence	10
Function minimal_drivers	11
Function single_drivers	11
Module StableMotifs.Export	12
Functions	12
Function attractor_dataframe	12
Function expanded_network	12
Function format_reduction_label	12
Function networkx_succession_diagram	12
Function networkx_succession_diagram_motif_based	13
Function networkx_succession_diagram_reduced_network_based	13
Function plot_nx_succession_diagram	14
Function save_to_graphml	14
Module StableMotifs.Format	15
Functions	15

Function bnet2sympy	15
Function bnetDNF2list	15
Function booleannet2bnet	15
Function build_rule_using_bnetDNFs	15
Function implicant2bnet	16
Function import_primes	16
Function longbnet2primes	16
Function pretty_print_prime_rules	17
Function pretty_print_primes	17
Function pretty_print_rspace	17
Function primes2bnet	17
Function remove_comment_lines	18
Function rule2bnet	18
Function statedict2str	18
Function statelist2dict	18
Function statestring2dict	19
Function sympy2bnet	19
Module StableMotifs.RandomBooleanNetworks	19
Functions	19
Function Binary_Rule_From_Decimal	19
Function Binary_Rules_From_Decimal	20
Function Random_Boolean_Network_Ensemble_Kauffman	20
Function String_Rule_From_Binary	20
Function String_Rules_From_Binary	21
Function get_criticality_K_Kauffman	21
Function get_criticality_p_Kauffman	21
Function read_Boolean_network_decimal	21
Function write_Boolean_network_decimal	22
Classes	22
Class RandomBooleanNetworks	22
Attributes	22
Methods	22
Module StableMotifs.Reduction	23
Functions	23
Function delete_node	23
Function deletion_reduction	23
Function mediator_reduction	24
Function reduce_primes	24
Function remove_outdag	24
Function simplify_primes	25
Function simplify_using_expression_and_negation	25
Classes	25
Class MotifReduction	25
Parameters	25
Attributes	26
Methods	27
Module StableMotifs.RestrictSpace	30
Functions	30
Function attractor_space_candidates	30
Function fixed_rspace_nodes	30
Function partial_state_contradicts_rspace	30
Function reduce_rspace	31
Function reduce_rspace_string	31
Function rspace	31
Function state_in_rspace	32

Module <code>StableMotifs.Succession</code>	32
Functions	32
Function <code>build_succession_diagram</code>	32
Classes	33
Class <code>SuccessionDiagram</code>	33
Attributes	33
Methods	33
Module <code>StableMotifs.TimeReversal</code>	37
Functions	37
Function <code>time_reverse_primes</code>	37

Module `StableMotifs`

Sub-modules

- [StableMotifs.Attractor](#)
- [StableMotifs.AttractorRepertoire](#)
- [StableMotifs.DomainOfInfluence](#)
- [StableMotifs.Export](#)
- [StableMotifs.Format](#)
- [StableMotifs.RandomBooleanNetworks](#)
- [StableMotifs.Reduction](#)
- [StableMotifs.RestrictSpace](#)
- [StableMotifs.Succession](#)
- [StableMotifs.TimeReversal](#)

Module `StableMotifs.Attractor`

Classes

Class `Attractor`

```
class Attractor(
    reduction,
    reduction_attractor_id
)
```

Stores attractor data for a reduced network. Automatically initialized by the `AttractorRepertoire` class.

Parameters

reduction : `Reduction.MotifReduction` Motif reduction to use as the representative (see attributes).
reduction_attractor_id : `int` Reduction id to use for the representative (see attributes)

Attributes

logically_fixed_nodes : `partial state dictionary` The nodes that are fixed by percolation on the expanded network (i.e., not by up-stream oscillations)
representative : `Reduction.MotifReduction, int tuple` Entry 0 is a maximally reduced `Reduction.MotifReduction` object that contains the attractor. In general, other such objects contain the attractor, but they will correspond to equivalent reduced networks. Entry 1 is a unique identifier number (integer) for the attractor within the reduced network; this is necessary in cases when a fully reduced network contains multiple (complex) attractors.

reductions : list of Reduction.MotifReduction Maximally reduced MotifReductions that contain the attractor.

attractor_dict : dictionary a dictionary describing the node states in the attractor according to the following key - 1 variable is "ON" 0 variable is "OFF" X variable is known to oscillate ? at least one such variable must oscillate ! the attractor may be false; if it is genuine, at least one such variable must oscillate

stg : networkx.DiGraph The state transition graph corresponding to the attractor (if computed)

fixed_nodes : partial state dictionary All node states that are known to be fixed in the attractor.

oscillation_fixed_nodes : partial state dictionary Node states that are fixed in the attractor, but that are not fixed by percolation in the expanded network. These states are instead fixed by up-stream oscillation.

reduced_primes : PyBoolNet primes dictionary Update rules for the maximally reduced network that contains the attractor.

n_unfixed : int Number of nodes that are not logically fixed.

size_lower_bound : int Lower bound on number of states in attractor.

size_upper_bound : int Upper bound on number of states in attractor.

explored : bool True if all attractor states and transitions are explicitly computed.

guaranteed : bool True if and only if the attractor is known to be genuine. If False, the attractor may not actually be stable.

Methods

Method add_reduction

```
def add_reduction(
    self,
    reduction
)
```

Add a reduction to the attractor. Does not check for compatibility.

Parameters

reduction : Reduction.MotifReduction Motif reduction that also contains the attractor.

Module StableMotifs.AttractorRepertoire

Classes

Class AttractorRepertoire

```
class AttractorRepertoire
```

The class that stores information about attractors. Initialize using either `from_primes` or `from_succession_diagram`.

Attributes

succession_diagram : Succession.SuccessionDiagram Succession diagram summarizing the stable motif structure of the model.

attractors : list of Attractor.Attractor List of (possible) attractors in the model.

reduction_attractors : dictionary A dictionary with integer keys that correspond to the `succession_diagram.digraph` nodes. The dictionary values are lists of `Attractor.Attractor` objects that correspond to attractors that exist in the region of statespace corresponding to the reduced network represented by the key in the succession diagram.

fewest_attractors : int A lower bound on the number of attractors in the model.

most_attractors : **int** An upper bound on the number of attractors in the model.

primes : **PyBoolNet primes dictionary** The model rules.

succession_digraph : **networkx digraph** Networkx digraph representation of the succession_diagram object. If `AttractorRepertoire.simplify_diagram`, it is equivalent to `AttractorRepertoire.succession_digraph`. Otherwise, several of its nodes may be contracted (depending on input parameters).

attractor_equivalence_classes : **list** List of attractor equivalence classes. Each item is a dictionary with keys 'states', 'attractors', and 'reductions'. The 'states' value is a dictionary of variable values that all attractors in the class share. The 'attractors' value is a list of `Attractor` objects (i.e., a sublist of `self.attractors`); all attractors in this list have all relevant nodes equivalently characterized. The 'reductions' value is a list of `reduction_attractor` keys that collectively contain all the attractors in the class (and therefore cannot differ in any relevant node).

relevant_nodes : **list** List of nodes that are "relevant", i.e., if trap spaces differ in the values of these variables, then the corresponding succession diagram nodes and attractors will not be merged.

Static methods

Method `from_primes`

```
def from_primes(
    primes,
    max_simulate_size=20,
    max_stable_motifs=10000
)
```

Build the succession diagram and attractor repertoire from PyBoolNet formatted update rules rules.

Parameters

primes : **PyBoolNet primes dictionary** The model rules.

max_simulate_size : **int** Maximum number of variables for which to brute-force build a state transition graph (the default is 20).

max_stable_motifs : **int** Maximum number of output lines for PyBoolNet to process from the `AspSolver` (the default is 10000).

Returns

AttractorRepertoire `AttractorRepertoire` object for the input primes.

Method `from_succession_diagram`

```
def from_succession_diagram(
    succession_diagram
)
```

Build the succession diagram and attractor repertoire from a precomputed succession diagram.

Parameters

succession_diagram : **Succession.SuccessionDiagram** Succession diagram summarizing the stable motif structure of the model.

Returns

AttractorRepertoire `AttractorRepertoire` object for the input succession diagram.

Methods

Method `analyze_system`

```
def analyze_system(  
    self,  
    primes,  
    max_simulate_size=20,  
    max_stable_motifs=10000  
)
```

Build and process the succession diagram for the model.

Parameters

primes : PyBoolNet primes dictionary The model rules.

max_simulate_size : int Maximum number of variables for which to brute-force build a state transition graph (the default is 20).

max_stable_motifs : int Maximum number of output lines for PyBoolNet to process from the AspSolver (the default is 10000).

Method `simplify_diagram`

```
def simplify_diagram(  
    self,  
    projection_nodes,  
    merge_equivalent_reductions=True,  
    keep_only_projection_nodes=False,  
    condense_simple_paths=False  
)
```

Simplify the succession diagram for the model. This is done in two ways. First, variables can be designated ignorable using the `projection_nodes` parameter. If `keep_only_projection_nodes` is False, these variables are ignorable, otherwise, all other nodes are ignorable. When `merge_equivalent_reductions` is True, all nodes of the succession diagram that correspond to trap spaces whose fixed variables differ only in ignorable variables are contracted (in the graph theory sense). After this process, if `condense_simple_paths` is True, then all succession diagram nodes with in-degree equal to one are contracted with their parent node. This function constructs the `succession_digraph` and `attractor_equivalence_classes` attributes, which are described in the class documentation.

Parameters

projection_nodes : list of variable names These nodes will be ignored if `keep_only_projection_nodes` is False (default); otherwise, all nodes except these will be ignored.

merge_equivalent_reductions : bool Whether to contract succession diagram nodes whose reductions differ only in ignorable nodes.

keep_only_projection_nodes : bool Whether `projection_nodes` specifies non-ignorable nodes.

condense_simple_paths : bool Whether to contract nodes with in-degree one.

Method `summary`

```
def summary(  
    self  
)
```

Prints a summary of the attractors to standard output.

Module StableMotifs.DomainOfInfluence

Functions

Function GRASP

```
def GRASP(
    target,
    primes,
    GRASP_iterations,
    forbidden=None,
    GRASP_scores=<function GRASP_default_scores>
)
```

Search for drivers of target in primes using the method of Yang et al. 2018.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

GRASP_iterations : int The number of times to run the GRASP method.

forbidden : set of str variable names Variables to be considered uncontrollable (the default is None).

GRASP_scores : function Function to score candidates (the default is GRASP_default_scores; see that function for required inputs and outputs of the scoring function).

Returns

solutions : list of partial state dictionaries Each partial set dictionary represents a driver set whose LDOI contains the target.

Function GRASP_default_scores

```
def GRASP_default_scores(
    target,
    primes,
    candidates
)
```

Helper function for GRASP driver search. Scores candidate driver nodes.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

candidates : list of partial state dictionaries List of variable states that can potentially lead to the target.

Returns

scores : list of ints Logical domain of influence sizes for individual node states. If the node leads to a contradiction, the score will become the negative of the largest LDOI size. Scores are ordered in the same order as the candidates list.

Function all_drivers_of_size

```
def all_drivers_of_size(
    driver_set_size,
    target,
    primes,
    external_search_vars=None,
```

```

        internal_search_vars=None
    )

```

Short summary.

Parameters

driver_set_size : int The number of driver nodes to try to find.

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

external_search_vars : set of str variable names Node set not in target to consider as potential drivers. If None, then all nodes not fixed in target (the default is None).

internal_search_vars : set of str variable names Node set in target to consider as potential drivers. If None, all nodes in partial state (the default is None).

Returns

driver_sets : list of partial state dictionaries Each state dictionary in the list drives target.

Function `construct_GRASP_solution`

```

def construct_GRASP_solution(
    target,
    primes,
    candidates,
    scores
)

```

Helper function for GRASP driver search. Constructs individual driver set using the GRASP search method.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

candidates : list of partial state dictionaries List of variable states that can potentially lead to the target.

scores : list of ints Logical domain of influence sizes for individual node states. If the node leads to a contradiction, the score will become the negative of the largest LDOI size. Scores are ordered in the same order as the candidates list.

Returns

partial state dictionary A partial state that contains the target in its LDOI. If no such partial state is found, returns an empty dictionary instead.

Function `fixed_excludes_implicant`

```

def fixed_excludes_implicant(
    fixed,
    implicant
)

```

Returns True if and only if the (possibly partial) state “fixed” contradicts the implicant.

Parameters

fixed : partial state dictionary State (or partial state) representing fixed variable states.

implicant : partial state dictionary State (or partial state) representing the target implicant.

Returns

bool True if and only if the implicant contradicts the logical domain of influence of the fixed (partial) state.

Function `fixed_implies_implicant`

```
def fixed_implies_implicant(  
    fixed,  
    implicant  
)
```

Returns True if and only if the (possibly partial) state “fixed” implies the implicant.

Parameters

fixed : partial state dictionary State (or partial state) representing fixed variable states.
implicant : partial state dictionary State (or partial state) representing the target implicant.

Returns

bool True if and only if the implicant is in the logical domain of influence of the fixed (partial) state.

Function `initial_GRASP_candidates`

```
def initial_GRASP_candidates(  
    target,  
    primes,  
    forbidden  
)
```

Helper function for GRASP driver search. Constructs initial candidates for driver nodes.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.
primes : PyBoolNet primes dictionary Update rules.
forbidden : set of str variable names Variables to be considered uncontrollable (the default is None).

Returns

candidates : list of partial state dictionaries List of variable states that can potentially lead to the target.

Function `internal_drivers`

```
def internal_drivers(  
    target,  
    primes,  
    max_drivers=None  
)
```

Find internal (logical) driver nodes of target through brute-force.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.
primes : PyBoolNet primes dictionary Update rules.
max_drivers : int Maximum size of driver set to consider. If None, is set to the size of the partial state (as this is always sufficient to achieve the target) (the default is None).

Returns

driver_sets : list of partial state dictionaries Each state dictionary in the list drives target. These are sorted by length (smallest first).

Function knock_to_partial_state

```
def knock_to_partial_state(
    target,
    primes,
    min_drivers=1,
    max_drivers=None,
    forbidden=None
)
```

Find all partial states in primes that drive the target. Do not consider nodes in the forbidden list.

Parameters

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

min_drivers : int Minimum size of driver set to consider. (the default is 1).

max_drivers : int Maximum size of driver set to consider. If None, is set to the size of the partial state (as this is always sufficient to achieve the target) (the default is None).

forbidden : set of str variable names Variables to be considered uncontrollable (the default is None).

Returns

knocked_nodes : list of partial state dictionaries Each state dictionary in the list drives target. Supersets of previously considered dictionaries are excluded.

Function local_GRASP_reduction

```
def local_GRASP_reduction(
    solution,
    target,
    primes
)
```

A helper function for GRASP driver search. Reduces valid solutions to attempt to remove redundancies.

Parameters

solution : partial state dictionary Solution to be reduced; must contain the target in its LDOI.

target : partial state dictionary PyBoolNet implicant that defines target fixed node states.

primes : PyBoolNet primes dictionary Update rules.

Returns

partial state dictionary Reduced solution that also contains target in its LDOI.

Function logical_domain_of_influence

```
def logical_domain_of_influence(
    partial_state,
    primes,
    implied_hint=None,
    contradicted_hint=None
)
```

Computes the logical domain of influence (LDOI) (see Yang et al. 2018)

Parameters

partial_state : **partial state dictionary** PyBoolNet implicant that defines fixed nodes.
primes : **PyBoolNet primes dictionary** Update rules.
implied_hint : **partial state dictionary** Known subset of the LDOI; used during optimization.
contradicted_hint : **partial state dictionary** Known subset of the contradiction boundary; used during optimization.

Returns

implied : **partial state dictionary** The logical domain of influence.
contradicted : **partial state dictionary** The contradiction boundary.

Function `minimal_drivers`

```
def minimal_drivers(  
    target,  
    primes,  
    max_drivers=None  
)
```

Finds smallest set(s) of (logical) driver nodes of target through brute-force. Unlike `minimal_drivers`, we are not limited to internal drivers nodes.

Parameters

target : **partial state dictionary** PyBoolNet implicant that defines target fixed node states.
primes : **PyBoolNet primes dictionary** Update rules.
max_drivers : **int** Maximum size of driver set to consider. If None, is set to the size of the partial state (as this is always sufficient to achieve the target) (the default is None).

Returns

driver_sets : **list of partial state dictionaries** Each state dictionary in the list drives target. These are sorted by length (smallest first).

Function `single_drivers`

```
def single_drivers(  
    target,  
    primes  
)
```

Finds all 1-node (logical) drivers of target under the rules given by primes.

Parameters

target : **partial state dictionary** PyBoolNet implicant that defines target fixed node states.
primes : **PyBoolNet primes dictionary** Update rules.

Returns

list of length-1 dictionaries Each dictionary describes a single node state that contains the target in its logical domain of influence.

Module `StableMotifs.Export`

Functions

Function `attractor_dataframe`

```
def attractor_dataframe(
    ar
)
```

Summarize the input attractor repertoire in a pandas DataFrame (requires pandas).

Parameters

ar : **AttractorRepertoire** Attractor repertoire to summarize.

Returns

pandas.DataFrame Summary of the attractors.

Function `expanded_network`

```
def expanded_network(
    primes,
    single_parent_composites=False
)
```

Produce the expanded network for given input update rules.

Parameters

primes : **PyBoolNet primes dictionary** The update rules for which to construct the expanded network.

single_parent_composites : **bool** Whether to insert composite nodes between virtual nodes when one is a prime implicant of the other. If False, the number of nodes is decreased; if True, then the expanded network is bipartite (the default is False).

Returns

networkx.DiGraph Digraph representing the expanded network. Nodes have a 'type' attribute that can be either 'virtual' or 'composite'.

Function `format_reduction_label`

```
def format_reduction_label(
    s
)
```

Helper function to make graph labels more readable. Removes single quotes and square brackets from the input string.

Parameters

s : **str** Input label.

Returns

str Label with characters ' , [, and] removed.

Function `networkx_succession_diagram`

```
def networkx_succession_diagram(
    ar,
    include_attractors_in_diagram=True,
    use_compressed_diagram=True
)
```

Label the succession diagram and (optionally) attractors of the input attractor repertoire according to the conventions of Rozum et al. (2021). This is an alias for the function `Export.networkx_succession_diagram_reduced_network_based`.

Parameters

ar : AttractorRepertoire Attractor repertoire object for which to build the diagram.
include_attractors_in_diagram : bool Whether attractors should be represented as nodes in the diagram (the default is True).
use_compressed_diagram : bool Whether to use the (potentially compressed) succession diagram stored in `ar.succession_digraph` instead of the complete one `ar.succession_diagram.digraph`. These are equivalent unless `ar.simplify_diagram` is called. See `AttractorRepertoire.py` for additional details. The default is True.

Returns

networkx.DiGraph A labeled digraph that represents the succession diagram.

Function `networkx_succession_diagram_motif_based`

```
def networkx_succession_diagram_motif_based(
    ar,
    include_attractors_in_diagram=True
)
```

Label the succession diagram and (optionally) attractors of the input attractor repertoire according to the conventions of Zanudo and Albert (2015). If attractors are not included, this is the line graph of the succession diagram defined in Rozum et al. (2021). Does not support compression.

Parameters

ar : AttractorRepertoire Attractor repertoire object for which to build the diagram.
include_attractors_in_diagram : bool Whether attractors should be represented as nodes in the diagram (the default is True).

Returns

networkx.DiGraph A labeled digraph that represents the succession diagram.

Function `networkx_succession_diagram_reduced_network_based`

```
def networkx_succession_diagram_reduced_network_based(
    ar,
    include_attractors_in_diagram=True,
    use_compressed_diagram=True
)
```

Label the succession diagram and (optionally) attractors of the input attractor repertoire according to the conventions of Rozum et al. (2021).

Parameters

ar : AttractorRepertoire Attractor repertoire object for which to build the diagram.
include_attractors_in_diagram : bool Whether attractors should be represented as nodes in the diagram (the default is True).
use_compressed_diagram : bool Whether to use the (potentially compressed) succession diagram stored in `ar.succession_digraph` instead of the complete one `ar.succession_diagram.digraph`. These are equivalent unless `ar.simplify_diagram` is called. See `AttractorRepertoire.py` for additional details. The default is True.

Returns

networkx.DiGraph A labeled digraph that represents the succession diagram.

Function `plot_nx_succession_diagram`

```
def plot_nx_succession_diagram(
    G,
    pos=None,
    fig_dimensions=(None, None),
    nx_node_kwargs=None,
    nx_edge_kwargs=None,
    draw_node_labels=True,
    draw_edge_labels=False,
    nx_node_label_kwargs=None,
    nx_edge_label_kwargs=None
)
```

Plot the input succession diagram. Requires matplotlib. For finer control over plot appearance, it is recommended to plot `g` directly.

Parameters

`g : networkx.DiGraph` Labeled succession diagram, e.g., as is output from `Export.networkx_succession_diagram_reduced_network_based()`.

`fig_dimensions : (int,int)` Dimensions of the output figure. If `(None, None)`, then the dimensions are calculated based on the number of nodes in `g` (the default is `(None, None)`).

`pos : str or graphviz_layout` Layout for the nodes; A dictionary with nodes as keys and positions as values. Positions should be sequences of length 2. If none, we attempt to use `pydot/graphviz` to construct a layout, otherwise we fall back to the `networkx` `planar_layout` function (succession diagrams are always planar).

`draw_node_labels : bool` Whether node labels should be drawn (`True`) or left as metadata (`False`) (the default is `True`).

`draw_edge_labels : bool` Whether edge labels should be drawn (`True`) or left as metadata (`False`) (the default is `True`).

`nx_node_kwargs : dictionary` Keyword arguments passed to `nx.draw_networkx_nodes` (in addition to `G` and `pos`). If `None`, we pass `{'node_size':50*G.number_of_nodes()}` by default.

`nx_edge_kwargs : dictionary` Keyword arguments passed to `nx.draw_networkx_edges` (in addition to `G` and `pos`). If `None`, we pass `{'arrowstyle':'->', 'width':2, 'arrowsize':30}` by default.

`nx_node_label_kwargs : dictionary` Keyword arguments passed to `nx.draw_networkx_labels` (in addition to `G` and `pos`). If `None`, we pass `{'font_size':16}` by default.

`nx_edge_label_kwargs : dictionary` Keyword arguments passed to `nx.draw_networkx_edge_labels` (in addition to `G` and `pos`). If `None`, we pass `{'font_size':16}` by default.

Function `save_to_graphml`

```
def save_to_graphml(
    G,
    model_name
)
```

Export a labeled succession diagram to graphml format.

Parameters

`G : networkx.DiGraph` Labeled succession diagram to export.

`model_name : str` Name of file to save to (.graphml extension will be appended).

Module StableMotifs.Format

Functions

Function bnet2sympy

```
def bnet2sympy(  
    rule  
)
```

Converts a BNet string expression to a sympy string expression.

Parameters

rule : **str** Boolean expression in BNET format.

Returns

str Expression in sympy format.

Function bnetDNF2list

```
def bnetDNF2list(  
    bnet  
)
```

Converts a BNet string expression to a list of prime implicant dictionaries.

Parameters

bnet : **str** BNET formatted rules.

Returns

list of partial state dictionaries Variable states specified by each dictionary are to be thought of as “AND”- connected, and the dictionaries as “OR”-connected.

Function booleannet2bnet

```
def booleannet2bnet(  
    rules  
)
```

Converts BooleanNet rules to BNet format. e.g., an input of “A*=B or C and not D” returns A, B | C & !D

Also replaces ~ with !

Parameters

rules : **str** BooleanNet formatted rules.

Returns

str BNET formatted rules.

Function build_rule_using_bnetDNFs

```
def build_rule_using_bnetDNFs(  
    expr0,  
    expr1  
)
```

Converts a BNet string expression (expr1) and its negation (expr0) to a PyBoolNet rule list. Note that this function does not test for consistency between expr0 and expr1.

Parameters

expr0 : str Rule, in BNET format, for the “OFF” state of a variable.

expr1 : str Rule, in BNET format, for the “ON” state of a variable.

Returns

PyBoolNet rule list The complementary expressions as they would appear in a PyBoolNet primes dictionary for a variable whose update rule is given by expr1.

Function `implicant2bnet`

```
def implicant2bnet(
    partial_state
)
```

Converts a partial state dictionary to a BNet string e.g., {'A':1,'B':0} returns 'A & !B'

Parameters

partial_state : partial state dictionary Partial state to convert.

Returns

str BNET representation of the partial state.

Function `import_primes`

```
def import_primes(
    fname,
    format='BooleanNet',
    remove_constants=False
)
```

Import boolean rules from file and return PyBoolNet formatted primes list.

Parameters

fname : str Path to (plaintext) file containing Boolean rules in format specified by the 'format' option.

format : str Boolean rule format; options are 'BooleanNet' or 'BNet' (the default is 'BooleanNet').

remove_constants : bool If True, variables that are constant are removed and their influence is percolated. Otherwise, they remain and we consider initial conditions in opposition to their values (the default is False).

Returns

PyBoolNet primes dictionary Update rules in PyBoolNet format.

Function `longbnet2primes`

```
def longbnet2primes(
    BNET,
    remove_constants=False
)
```

A modified version of PyBoolNet's `bnet2primes` that does not do path-checking, as this can cause errors if the bnet rules are very long. Assumes BNET is a bnet string, not a file.

Parameters

BNET : str BNET formatted rules to convert.

remove_constants : bool Whether or not to remove and percolate constant input values (the default is False).

Returns

primes : PyBoolNet primes dictionary Update rules in PyBoolNet format.

Function pretty_print_prime_rules

```
def pretty_print_prime_rules(  
    primes  
)
```

Prints PyBoolNet a prime dictionary as Boolean rules The output format is of the form: $A^* = B \ \& \ C \mid !D$, for example.

Parameters

primes : PyBoolNet primes dictionary Update rules to print.

Function pretty_print_primes

```
def pretty_print_primes(  
    primes  
)
```

Prints PyBoolNet a prime dictionary in a more readable format. Prints both state updates (1 and 0).

Parameters

primes : PyBoolNet primes dictionary Update rules to print.

Function pretty_print_rspace

```
def pretty_print_rspace(  
    L,  
    simplify=True,  
    silent=True  
)
```

Produces string representation of the Boolean rule describing the input rspace L (see RestrictSpace.rspace).

Parameters

L : rspace list Restrict space list (see RestrictSpace.rspace for details).

simplify : bool Whether to simplify the rule (the default is True).

silent : bool Whether to suppress output of the rule (the default is True).

Returns

str BNET expression that is true in and only in the rspace specified by L.

Function primes2bnet

```
def primes2bnet(  
    primes  
)
```

A simpler version of PyBoolNet's FileExchange.primes2bnet function with fewer options and less organized output. Should handle prime rules with tautologies better than the PyBoolNet version though.

Parameters

primes : PyBoolNet primes dictionary Update rules to convert.

Returns

str BNET representation of update rules.

Function `remove_comment_lines`

```
def remove_comment_lines(
    stream,
    comment_char='#'
)
```

Removes commented out lines from stream, e.g., those starting with '#'.

Parameters

stream : **iterable of str** Lines from which comments should be excluded.
comment_char : **str** Lines beginning with this character will be excluded.

Returns

list of str Lines that do not begin with comment_char.

Function `rule2bnet`

```
def rule2bnet(
    rule
)
```

Converts a PyBoolNet prime rule into a BNet string. e.g., [{'A':1,'B':0},{'C':0}] returns 'A & !B | !C'

Parameters

rule : **list of PyBoolNet partial states** Update rule to convert.

Returns

str BNET representation of Boolean expression.

Function `statedict2str`

```
def statedict2str(
    statedict
)
```

Converts a state dictionary to a statestring using alphabetical sorting.

Parameters

statedict : **partial state dictionary** State to convert to a binary string representation.

Returns

str A binary string, with each position corresponding to the variable name at the same position in the alphabetized keys in statedict.

Function `statelist2dict`

```
def statelist2dict(
    names,
    statestrings
)
```

Converts a collection of statestrings to a dictionary.

Parameters

names : **list of str** An ordered list of variable names; (alphabetical order is PyBoolNet's default, e.g. sorted(primes)).

c : **iterable of str** Each element should be a binary string, with each position corresponding to the variable name at the same position in names.

Returns

dictionary Dictionary summarizing c. If a node takes the same value in every state, the corresponding dictionary value matches its fixed value; otherwise, the dictionary value is 'X'.

Function `statestring2dict`

```
def statestring2dict(
    statestring,
    names
)
```

Converts a state string, which specifies a node in an STG, to the corresponding dictionary representation.

Parameters

statestring : **str** A binary string, e.g., '01101'.

names : **list of str** An ordered list of variable names; (alphabetical order is PyBoolNet's default, e.g. `sorted(primes)`).

Returns

dictionary The keys are the elements of names and the values are the corresponding value in statestring.

Function `sympy2bnet`

```
def sympy2bnet(
    rule
)
```

Converts a sympy string expression to a BNET string expression.

Parameters

rule : **str** Boolean expression in sympy format.

Returns

str Expression in BNET format.

Module `StableMotifs.RandomBooleanNetworks`

Functions

Function `Binary_Rule_From_Decimal`

```
def Binary_Rule_From_Decimal(
    node_rule_decimal,
    node_input_list
)
```

Convert single decimal rule to its binary form.

Parameters

node_rule_decimal : **int** Decimal form of a truth table's output column.

node_input_list : **list of str** Variable names that correspond to each column of the truth table.

Returns

list of int Binary rule list corresponding to an output column of a truth table.

Function Binary_Rules_From_Decimal

```
def Binary_Rules_From_Decimal(  
    node_rules_decimal_dictionary  
)
```

Construct Binary format rules from decimal format rules.

Parameters

node_rules_decimal_dictionary : **dictionary** Rules in decimal format to convert.

Returns

dictionary Binary rules dictionary.

Function Random_Boolean_Network_Ensemble_Kauffman

```
def Random_Boolean_Network_Ensemble_Kauffman(  
    N,  
    K,  
    p,  
    N_ensemble,  
    seed=1000,  
    write_Boolean_network=False  
)
```

Generate a sample from the Kauffman NK RBN ensemble.

Parameters

N : **int** Number of nodes of RBN.

K : **int** Number of inputs of each node in the RBN.

p : **float** Probability that each entry in each truth table output column is equal to 1.

N_ensemble : **int** Number of networks to generate.

seed : **int** Random seed for generating the RBN ensemble (the default is 1000).

write_Boolean_network : **bool** Whether to write each network in the ensemble as a CSV file in a new directory (the default is False).

Returns

RBN_ensemble_rules : **list of str** Each string are the Boolean rules of an ensemble in boolean-net format. Each element in **RBN_ensemble_rules** can be used as an input for the `Format.booleannet2bnet` function.

Function String_Rule_From_Binary

```
def String_Rule_From_Binary(  
    node_rule_binary,  
    node_input_list  
)
```

Convert binary rule to BooleanNet format.

Parameters

node_rule_binary : **list of int** Binary rule list corresponding to an output column of a truth table.

node_input_list : **list of str** Variable names that correspond to each column of the truth table.

Returns

str BooleanNet representation of rule.

Function String_Rules_From_Binary

```
def String_Rules_From_Binary(
    node_rules_binary_dictionary
)
```

Convert from binary dictionary rule format to BooleanNet format.

Parameters

node_rules_binary_dictionary : dictionary Binary dictionary representation of rules.

Returns

str BooleanNet representation of rules.

Function get_criticality_K_Kauffman

```
def get_criticality_K_Kauffman(
    p
)
```

The Kauffman RBN is at criticality when $K = 2/(p(1-p))$.

Parameters

p : float Probability that each entry in each truth table output column is equal to 1.

Returns

K_criticality : int Number of inputs of each node in the RBN.

Function get_criticality_p_Kauffman

```
def get_criticality_p_Kauffman(
    K
)
```

The Kauffman RBN is at criticality when $K = 2/(p(1-p))$.

Parameters

K : int Number of inputs of each node in the RBN.

Returns

p_criticality : float Probability that each entry in each truth table output column is equal to 1.

Function read_Boolean_network_decimal

```
def read_Boolean_network_decimal(
    filename
)
```

Imports rules from csv in decimal format.

Parameters

filename : str Path to csv from which to import decimal-formatted rules.

Returns

str Rules in BooleanNet format.

Function write_Boolean_network_decimal

```
def write_Boolean_network_decimal(
    node_rules_decimal_dictionary,
    filename
)
```

Write the decimal format of the Boolean rules to file.

Parameters

node_rules_decimal_dictionary : **dictionary** Update rule truth table in decimal format.
filename : **str** Path to file for csv output of the truth table.

Classes

Class RandomBooleanNetworks

```
class RandomBooleanNetworks
```

Generator of random Boolean networks (RBN) and ensembles of RBN. The RandomBooleanNetworks class object is a Boolean model and stores information of how the Boolean model was generated. It has functions that generate ensembles of RBN by generating multiple RandomBooleanNetworks objects.

Attributes

node_names : **list of str** List of variable names.
node_inputs_dictionary : **dictionary** Each value is a (fixed order) list of the names of the nodes whose values are inputs into the key variable's update function.
node_rules_binary_dictionary : **dictionary** Each value is a list of outputs for the key variable's update function, stored as a list in ascending order of the numerical representation of the input row.
node_rules_decimal_dictionary : **dictionary** Decimal conversion of node_rules_binary_dictionary.
node_rules_string_dictionary : **dictionary** BooleanNet (str) conversion of node_rules_binary_dictionary values.
node_rules_string : **str** BooleanNet representation of update rules.
random_Boolean_type : **str** Description of generative process. Currently only "Kauffman NK" is implemented.
N : **int** Number of nodes in the Boolean network.
random_Boolean_Network_parameters : **list** For Kauffman NK generation - [K,p], where K is the in-degree and p is the bias. K is a positive integer less than or equal to N, and p is a float between 0 and 1 (inclusive).
random_seed : **int** Seed for random functions.
filename : **str** Path to file where network data are stored. If None, no files are written.

Methods

Method Random_Boolean_Network

```
def Random_Boolean_Network(
    self,
    random_Boolean_type,
    N,
    rbn_parameters,
    seed=None,
    filename=None
)
```

Construct network using specified generative process.

Parameters

random_Boolean_type : **str** Description of generative process. Currently only “Kauffman NK” is implemented.

N : **int** Number of nodes in the Boolean network.

random_Boolean_Network_parameters : **list** For Kauffman NK generation - [K,p], where K is the in-degree and p is the bias. K is a positive integer less than or equal to N, and p is a float between 0 and 1 (inclusive).

random_seed : **int** Seed for random functions.

filename : **str** Path to file where network data are stored. If None, no files are written.

Method `Random_Boolean_Network_Rules`

```
def Random_Boolean_Network_Rules(  
    self  
)
```

Generate various conversions of the `node_rules_binary_dictionary` attribute.

Module `StableMotifs.Reduction`

Functions

Function `delete_node`

```
def delete_node(  
    primes,  
    node  
)
```

Reduces Boolean rules given by primes by deleting the variable specified by node. The deleted node may not appear in its own update function. Any update rules depending on the deleted node will have that dependence replaced by the update function of the deleted node. The rules are simplified after node deletion.

Parameters

primes : **PyBoolNet primes dictionary** Update rules.

node : **str** Name of the node to delete.

Returns

new_primes : **PyBoolNet primes dictionary** The reduced primes.

constants : **partial state dictionary** Node states that became logically fixed during simplification.

Function `deletion_reduction`

```
def deletion_reduction(  
    primes,  
    max_in_degree=inf  
)
```

Implements the reduction method of Veliz-Cuba (2011). Deletion order is such that nodes with low in-degree are prioritized for removal. Deletion proceeds until all remaining nodes have self-loops.

Parameters

primes : **PyBoolNet primes dictionary** Update rules.

max_in_degree : int or float Will not try to delete nodes with in-degree larger than this. Deleting nodes with large in-degree can be computationally expensive (the default is `float('inf')`).

Returns

reduced : PyBoolNet primes dictionary The reduced primes.

constants : partial state dictionary Node states that became logically fixed during reduction.

Function mediator_reduction

```
def mediator_reduction(  
    primes  
)
```

Network reduction method of Saadadtpour, Albert, Reluga (2013) Preserves fixed points. Number of complex attractors is often, but not always conserved (despite initial claims). Can be viewed as a more restrictive version of the deletion reduction method of Veliz-Cuba (2011).

Parameters

primes : PyBoolNet primes dictionary Update rules.

Returns

reduced : PyBoolNet primes dictionary The reduced primes.

constants : partial state dictionary Node states that became logically fixed during reduction.

Function reduce_primes

```
def reduce_primes(  
    fixed,  
    primes  
)
```

Simplifies boolean rules when some nodes are held fixed

Parameters

fixed : partial state dictionary Node states to be held fixed.

primes : PyBoolNet primes dictionary Update rules.

Returns

reduced_primes : PyBoolNet primes dictionary Simplified update rules

percolated_states : partial state dictionary Fixed node states (including inputs) that were simplified and removed.

Function remove_outdag

```
def remove_outdag(  
    primes  
)
```

Removes the terminal directed acyclic part of the regulatory network. This part of the network does not influence the attractor repertoire.

Parameters

primes : PyBoolNet primes dictionary Update rules.

Returns

reduced : **PyBoolNet primes dictionary** The reduced primes.
constants : **partial state dictionary** Node states that became logically fixed during reduction.

Function `simplify_primes`

```
def simplify_primes(  
    primes  
)
```

Simplifies PyBoolNet primes (e.g., $A \mid A \ \& \ B$ becomes A)

Parameters

primes : **PyBoolNet primes dictionary** Rules to simplify.

Returns

PyBoolNet primes dictionary Simplified rules.

Function `simplify_using_expression_and_negation`

```
def simplify_using_expression_and_negation(  
    node,  
    expr0,  
    expr1,  
    bnet  
)
```

Simplify the expression `bnet` by substituting the value for `node` given by `node = expr1 = !expr0` (does not check that `expr1 != expr0`)

Parameters

node : **str** Name of node to substitute

expr0 : **str** Expression to substitute for `!node`

expr1 : **str** Expression to substitute for `node`

bnet : **str** BNET expression in which to perform the substitutions.

Returns

str Simplified BNET expression after substitutions are performed.

Classes

Class `MotifReduction`

```
class MotifReduction(  
    motif_history,  
    fixed,  
    reduced_primes,  
    max_simulate_size=20,  
    prioritize_source_motifs=True,  
    max_stable_motifs=10000  
)
```

Class to generate and store data about a network reduction that arises during the stable motif succession diagram construction algorithm.

Parameters

motif_history : **list of partial state dictionaries** Stable motifs that can lock in to give the reduced network (in order).

fixed : **partial state dictionary** Nodes values that have been fixed and reduced by stable motifs and their logical domain of influence.

reduced_primes : **PyBoolNet primes dictionary** Update rules for the reduced network.

max_simulate_size : **int** Maximum number of variables for which to brute-force build a state transition graph (the default is 20).

prioritize_source_motifs : **bool** Whether source nodes should be considered first (the default is True).

max_stable_motifs : **int** Maximum number of output lines for PyBoolNet to process from the AspSolver (the default is 10000).

Attributes

merged_history_permutations : **list of lists of int** Permutations of motif_history (by index) that also yield this reduction.

logically_fixed_nodes : **partial state dictionary** Nodes values that have been fixed and reduced by stable motifs and their logical domain of influence.

time_reverse_primes : **PyBoolNet primes dictionary** Update rules of the time reversed reduced system.

stable_motifs : **list of partial state dictionaries** Stable motifs of the reduced system.

time_reverse_stable_motifs : **list of partial state dictionaries** Stable motifs of the time reversed system.

merged_source_motifs : **list of partial state dictionaries** List of source-like stable motifs that have been merged into a single motif to avoid redundancy.

source_independent_motifs : **list of partial state dictionaries** Stable motifs that exist independent of the values of the source nodes

merge_source_motifs : **list of partial state dictionaries** Stable motifs generated by merging the stable motifs corresponding to source nodes.

rspace : **rspace list** The rspace, or “restrict space” of the reduced network, describing a necessary condition for the system to avoid activating additional stable motifs (see RestrictSpace.py for further details).

motif_history : **list of partial state dictionaries** Stable motifs that can lock in to give the reduced network (in order)

reduced_primes : **PyBoolNet primes dictionary** Update rules for the reduced network.

fixed_rspace_nodes : **partial state dictionary** Nodes values that are fixed in the rspace.

rspace_constraint : **str** BNET expression that is true in and only in the rspace.

reduced_rspace_constraint : **str** S simplification of the rspace_constraint given the fixed_rspace_nodes states are satisfied

rspace_update_primes : **PyBoolNet primes dictionary** The update rules obtained from simplifying under the assumption that the fixed_rspace_nodes are fixed

conserved_functions : **list of PyBoolNet expressions** Boolean functions that are constant within every attractor, in PyBoolNet update rule format

rspace_attractor_candidates : **list of str** Attractors (lists of statestrings) in the rspace_update_primes that satisfy the reduced_rspace_constraint

partial_STG : **networkx.DiGraph** Subgraph of the state transition graph of the reduced network that contains any and all attractors that do not lie in any of the reduced network’s stable motifs.

no_motif_attractors : **list of str** Complex attractors that do not “lock in” any additional stable motifs, stored as collections of state strings.

attractor_dict_list : **list of dictionaries** Dictionaries corresponding to attractors that are in this reductions, but not in any of its subreductions (if it has any). Each describes the node states in the attractors according to the following 1 variable is “ON” 0 variable is “OFF” X variable is known to oscillate ? at least one such variable must oscillate ! the attractor may be false; if it is genuine, at least one such variable must oscillate

terminal : **str** One of “yes”, “no”, or “possible”, indicating whether the reduction contains attractors that are not in any of its subreductions.

delprimes : **PyBoolNet prime dictionary** Update rules for the system’s deletion projection. Steady states and stable motif activation are preserved. These rules may yield

additional, spurious complex attractors.
deletion_STG : **networkx.DiGraph** Portion of the deletion projection's STG that contains all motif-avoidant attractors.
deletion_no_motif_attractors : **list of str** Motif avoidant attractors of the deletion projection. The number of these is an upper bound on the number of motif avoidant attractors in the reduction.

Methods

Method `build_K0`

```
def build_K0(
    self
)
```

Helper function for smart STG building. Builds initial set of nodes that are not part of any motif-avoidant attractor.

Returns

set of str Statestrings that do not belong to any motif-avoidant attractor.

Method `build_deletion_STG`

```
def build_deletion_STG(
    self,
    max_stable_motifs=10000
)
```

Build a piece of the STG that is guaranteed to contain all motif-avoidant attractors of the deletion projection. Complex attractors found here may be spurious.

Parameters

max_stable_motifs : **int** Maximum number of output lines for PyBoolNet to process from the AspSolver (the default is 10000).

Method `build_inspace`

```
def build_inspace(
    self,
    ss,
    names,
    tr_stable_motifs=None
)
```

Helper function for smart STG building. List all time reversal stable motifs to which (partial) state `ss` belongs.

Parameters

ss : **str** Statestring (possibly on a subspace).

names : **list of str** Variable names ordered to correspond to the positions of `ss`.

tr_stable_motifs : **list of partial state dictionaries** Time reverse stable motifs. If `None`, use all time reverse stable motifs in the reduced system (the default is `None`).

Returns

list of partial state dictionaries Time reverse stable motifs that are active in the state `ss`.

Method build_partial_STG

```
def build_partial_STG(
    self
)
```

Build a piece of the STG that is guaranteed to contain all motif-avoidant attractors of the reduction.

Method find_constants_in_complex_attractor

```
def find_constants_in_complex_attractor(
    self,
    c
)
```

Given a set of strings representing the states of a complex attractor the function finds the nodes that are constant in the full complex attractor.

Parameters

c : a set of binary strings Set of statestrings, e.g. set(['000', '010', '100']).

Returns

list of str An array consisting of 0s, 1s, and Xs. X represents an oscillating node, and the 0s and 1s represent nodes stabilized to those states.

Method find_deletion_no_motif_attractors

```
def find_deletion_no_motif_attractors(
    self,
    max_stable_motifs=10000
)
```

Identify motif-avoidant attractors in the deletion projection.

Parameters

max_stable_motifs : int Maximum number of output lines for PyBoolNet to process from the AspSolver (the default is 10000).

Method find_no_motif_attractors

```
def find_no_motif_attractors(
    self
)
```

Find attractors of the reduction that are not present in any of its subreductions.

Method generate_attr_dict

```
def generate_attr_dict(
    self
)
```

Generate a list of attractors that are present in the reduction, but not in any of its subreductions.

Returns

list of dictionaries Dictionaries corresponding to attractors that are in this reductions, but not in any of its subreductions (if it has any). Each describes the node states in the attractors according to the following 1 variable is "ON" 0 variable is "OFF" X variable

is known to oscillate ? at least one such variable must oscillate ! the attractor may be false; if it is genuine, at least one such variable must oscillate

Method `in_motif`

```
def in_motif(
    self,
    ss,
    names
)
```

Tests whether the (partial) state `ss` is in any stable motifs

Parameters

`ss` : **str** Statestring (possibly on a subspace).

`names` : **list of str** Variable names ordered to correspond to the positions of `ss`.

Returns

bool Whether `ss` is in any stable motif of the reduced system.

Method `merge_source_motifs`

```
def merge_source_motifs(
    self
)
```

Merges stable motifs (and time-reversal stable motifs) that correspond to source nodes, e.g. $A=A$, *into combined motifs to avoid combinatorial explosion*. For example, $A=A$, $B=B$, $C=C$ produces six motifs that can stabilize in 8 ways; without merging, these 8 combinations lead to $8 \cdot 3! = 48$ successions because they can be considered in any order. This is silly because source nodes all stabilize simultaneously.

We will assume that stable motifs and time reverse stable motifs have already been computed.

Note that a source node in the forward time system is a source node in the time reverse system as well. This follows from $A^* = A \Rightarrow A^- = \sim(A^*(A=\sim A)) = {}_A A = A$.

If $A^* = A$ or X (i.e., $A=1$ is a stable motif), then $A^- = {}_A A \mid X = A \& \sim X$, so $A=0$ is a time-reverse stable motif. A similar argument applies for the $A=0$ stable motif. Thus, a motif is only a source motif if it is also a time-reverse motif.

Method `summary`

```
def summary(
    self,
    show_original_rules=True,
    hide_rules=False,
    show_explicit_permutations=False
)
```

Print a summary of the reduction.

Parameters

`show_original_rules` : **bool** Show rules of the unreduced system (the default is True)?

`hide_rules` : **bool** Hide rules of the reduced system (the default is False)?

`show_explicit_permutations` : **bool** Show motif permutations explicitly, instead of by index (the default is False)?

Module `StableMotifs.RestrictSpace`

Functions

Function `attractor_space_candidates`

```
def attractor_space_candidates(
    maxts,
    trmaxts
)
```

Merge the maximum trap spaces `maxts` and time-reverse maximum trap spaces to obtain a list of attractor-conserved quantities. Note that any Boolean function of these is also conserved in attractors.

Parameters

maxts : **list of partial state dictionaries** Stable motifs, i.e., maximum trap spaces for the system.

trmaxts : **list of partial state dictionaries** Stable motifs, i.e., maximum trap spaces for the time-reversed system.

Returns

rspace() **list** Restrict space list (see `RestrictSpace.rspace` for details).

Function `fixed_rspace_nodes`

```
def fixed_rspace_nodes(
    L,
    primes
)
```

Finds the nodes that must have a fixed value in order for the rspace constraint `L` to be satisfied in the system given by `primes`.

Parameters

L : **rspace()** **list** Restrict space list (see `RestrictSpace.rspace` for details).

primes : **PyBoolNet primes dictionary** Update rule for the system.

Returns

dictionary Nodes that are fixed everywhere in the rspace `L`. Returns `{'0':1}` if `L` is a self-contradictory.

Function `partial_state_contradicts_rspace`

```
def partial_state_contradicts_rspace(
    state,
    L
)
```

Tests to see if state lies entirely outside the rspace `L`.

Parameters

state : **partial state dictionary** State, or partial state to test.

L : **rspace()** **list** Restrict space list (see `RestrictSpace.rspace` for details).

Returns

bool True if and only if state is not in `L`.

Function `reduce_rspace`

```
def reduce_rspace(  
    L,  
    primes  
)
```

Reduce the rspace L for the system given by primes so that trivially fixed nodes are factored out. The first element of the returned rspace (L2) will specify these trivially fixed nodes (i.e., they are factored on the left).

Parameters

L : **rspace()** list Restrict space list (see `RestrictSpace.rspace` for details).

primes : **PyBoolNet primes dictionary** Update rule for the system.

Returns

L2 : **rspace()** list Reduced restrict space list (see `RestrictSpace.rspace` for details).

Function `reduce_rspace_string`

```
def reduce_rspace_string(  
    s,  
    fd,  
    simplify=True  
)
```

Replaces variables in the string s with the fixed values given by the dictionary fd.

Parameters

s : **str** Boolean expression in BNET format.

fd : **partial state dictionary** Node values that are to be considered fixed.

simplify : **bool** Whether to simplify the expression using espresso (the default is True).

Returns

str String with substitutions made according to fd.

Function `rspace`

```
def rspace(  
    maxts,  
    trmaxts,  
    primes  
)
```

In order for none of the trap spaces to “lock in”, we would require that their single-node drivers are all sustained in a negated state. We can use this idea to hone the exclusion space. `rspace` will return the region that

- 1) has the negations of 1-node drivers of each `maxts` active and . . .
- 2) has the update rules of these 1-node drivers taking the appropriate value

In addition, a time-reverse trap space (`trmaxts`) describes a region that, once exited, cannot be reentered. Thus, if the LDOI of the region contains any contradiction, the region cannot contain any attractor. Therefore, we include a third criterion for the `rspace`:

- 3) is not in a state belonging to an attractor-free time-reversed trap space

The return value is a list L of lists of prime implicants. Each element of L is to be interpreted as a list of OR-separated prime implicants; L is to be interpreted as AND-separated. e.g., `L=[[{'A':0,'B':1},{C:0}],[{B':0,'D':1},{A':1}]]` should be read as `L = (!A&B | !C) & (!B & D | A)`

Parameters

mxts : list of partial state dictionaries Stable motifs, i.e., maximum trap spaces for the system.

trmxts : list of partial state dictionaries Stable motifs, i.e., maximum trap spaces for the time-reversed system.

primes : PyBoolNet primes dictionary Update rule for the system.

Returns

L : [rspace\(\)](#) list Description of rspace in list form (see summary above for details).

Function state_in_rspace

```
def state_in_rspace(
    state,
    L
)
```

Tests to see if state is in the rspace L.

Parameters

state : partial state dictionary State, or partial state to test.

L : [rspace\(\)](#) list Restrict space list (see RestrictSpace.rspace for details).

Returns

bool True if and only if state is in L.

Module StableMotifs.Succession

Functions

Function build_succession_diagram

```
def build_succession_diagram(
    primes,
    fixed=None,
    motif_history=None,
    diagram=None,
    merge_equivalent_motifs=True,
    max_simulate_size=20,
    prioritize_source_motifs=True,
    max_stable_motifs=10000
)
```

Recursively construct a succession diagram from the input update rules. Generally, it is preferable to construct this from within the AttractorRepertoire class (using, e.g., AttractorRepertoire.from_primes).

Parameters

primes : PyBoolNet primes dictionary Update rules.

fixed : partial state dictionary Used only for recursion. Specifies nodes to be fixed in the next reduced network to be added to the diagram.

motif_history : list of partial state dictionaries Used only for recursion. Specifies stable motif history for the next reduced network to be added to the diagram.

diagram : SuccessionDiagram Used only for recursion. The SuccessionDiagram object that is under construction.

merge_equivalent_motifs : bool If False, equivalent reduced networks have their data re-computed and copied. Making this False is only recommended if the succession diagram must be represented in a form that has no feedforward loops; making this True

provides large computational advantages, both in terms of speed and memory usage (the default is True).

max_simulate_size : int Maximum number of variables for which to brute-force build a state transition graph (the default is 20).

prioritize_source_motifs : bool Whether source nodes should be considered first (the default is True).

max_stable_motifs : int Maximum number of output lines for PyBoolNet to process from the AspSolver (the default is 10000).

Returns

SuccessionDiagram The succession diagram for the input update rules.

Classes

Class SuccessionDiagram

```
class SuccessionDiagram
```

Class describing the succession diagram of a Boolean system. See, e.g., Zanudo and Albert (2015) or Rozum et al. (2021).

Attributes

motif_reduction_dict : dictionary MotifReduction-valued dictionary with integer (index) keys (see Reduction.py).

digraph : networkx.DiGraph Topological structure of the succession diagram. Nodes are integers that align with the entries of motif_reduction_dict.

Methods

Method add_motif_permutation

```
def add_motif_permutation(
    self,
    reduction_index,
    permutation
)
```

Adds a permutation of a preexisting stable motif history to a precomputed MotifReduction object.

Parameters

reduction_index : int Index of the preexisting reduced network.

permutation : list of int Permutation that maps the preexisting history to the input history.

Method add_motif_reduction

```
def add_motif_reduction(
    self,
    motif_reduction
)
```

Inserts a given MotifReduction into the succession diagram. Does not check for consistency, but will insert a properly constructed MotifReduction into the correct place in the diagram.

Parameters

motif_reduction : MotifReduction Reduced network to be appended to the succession diagram.

Method `find_equivalent_reduction`

```
def find_equivalent_reduction(
    self,
    fixed
)
```

Extracts the **MotifReduction** object that has the frozen node values specified by `fixed`, if such an object exists (returns `None` otherwise).

Parameters

fixed : partial state dictionary Nodes values that have been fixed and reduced by stable motifs and their logical domain of influence.

Returns

MotifReduction Reduced network that has the frozen node values specified by `fixed`, if such an object exists (returns `None` otherwise).

Method `find_motif_permutation`

```
def find_motif_permutation(
    self,
    motif_history
)
```

Check whether some permutation of the input `motif_history` is already represented in the succession diagram. If so, return the preexisting reduction's index and the permutation that maps between the two histories.

Parameters

motif_history : list of partial state dictionaries Stable motifs that can lock in to give a given reduced network (in order).

Returns

reduction_index : int Index of the preexisting reduced network. This value is `None` if no such reduced network exists.

permutation : list of int Permutation that maps the preexisting history to the input history. This value is `None` if no such history exists.

Method `get_motifs`

```
def get_motifs(
    self
)
```

Extract the stable motifs of a system and its reduced networks from its attractor repertoire. Notably, these include both the system's primary stable motifs and conditionally stable motifs (see, e.g., Deritei et al. 2019).

Returns

list of dictionaries Stable motifs that appear in the system or during reduction (in no particular order).

Method `reduction_drivers`

```
def reduction_drivers(
    self,
    target_index,
    method='internal',
    max_drivers=None,
```

```

        GRASP_iterations=None
    )

```

Find control strategies that lead to the reduced network specified by the target index. Several control strategies are implemented. See `Succession.SuccessionDiagram.reprogram_to_trap_spaces` for a detailed description of control methods available. Generally, this method should not be used directly. Instead, use `reprogram_to_trap_spaces`.

Parameters

target_index : **int** Index of the target reduced network.
method : **str** One of 'internal', 'minimal', or 'GRASP'. See `Succession.SuccessionDiagram.reprogram_to_trap_spaces` for details.
max_drivers : **int** Maximum number of driver nodes to consider (not used in GRASP methods). If none, the upper limit is given by the number of free variables (the default is None).
GRASP_iterations : **int** Number of times to construct GRASP driver sets; only used in GRASP methods. If none, the number of iterations is chosen based on the network size (the default is None).

Returns

list Control strategies found; interpretation depends on method selected See `Succession.SuccessionDiagram.reprogram_to_trap_spaces` for details.

Method `reductions_indices_with_states`

```

def reductions_indices_with_states(
    self,
    logically_fixed,
    optimize=True
)

```

Find all reductions (by index) that have the nodes states specified logically fixed.

Parameters

logically_fixed : **partial state dictionary** Nodes states that should be fixed in all returned network reductions.
optimize : **bool** Whether to remove reduced networks that are subnetworks of valid reductions. This is generally recommended so as to obtain the most parsimonious control strategies (the default is True).

Returns

list of int Indices of reduced networks that have the appropriate fixed states.

Method `reprogram_to_trap_spaces`

```

def reprogram_to_trap_spaces(
    self,
    logically_fixed,
    target_method='history',
    driver_method='internal',
    max_drivers=None,
    GRASP_iterations=None,
    GRASP_score_override=None
)

```

Find driver sets that lead to fixing the node states specified.

Parameters

logically_fixed : **partial state dictionary** Targeted fixed nodes.
target_method : **str** Either 'history' or 'merge'; see Notes below for details.

driver_method : **str** Either 'internal', 'minimal', or 'GRASP' see Notes below for details.
max_drivers : **int** Maximum number of driver nodes to consider (not used in GRASP methods). If none, the upper limit is given by the number of free variables (the default is None).
GRASP_iterations : **int** Number of times to construct GRASP driver sets; only used in GRASP methods. If none, the number of iterations is chosen based on the network size (the default is None).
GRASP_score_override : **function** Optional heuristic score function override (see Domain-OfInfluence.GRASP for details). Only used in GRASP methods (the default is None).

Returns

list Control strategies found; interpretation depends on method selected See Notes below for details.

Notes

The various combinations of **target_method** and **driver_method** options result in different control strategies, which are outlined below.

target_method = history, **driver_method** = internal: Finds all shortest stable motif histories that result in the target node states being logically fixed. Each stable motif is searched for internal driver nodes. The resulting internal drivers are combined into a single control set. The return value consists of all such control sets for all stable motif histories. Each control set eventually becomes self-sustaining.

target_method = history, **driver_method** = minimal: Similar to the history method, except the search for stable motif drivers includes external driver nodes for the motif and does not extend to driver sets of larger size once one driver set has been found for a motif. Because the search includes external driver nodes, special care must be taken in interpreting the effect of the drivers, as their influence may impact the effect of motifs stabilizing. Thus, the control is only guaranteed to work if the interventions are temporary and implemented in the order specified by the motif history.

For this reason, the output consists of lists of ordered interventions. Each element of the return value is a list of lists of dictionaries. Each element of the return value represents a control strategy. To implement such a strategy, select a dictionary from the first element of the strategy and fix the node states it specifies until their influence has propagated through the system. Then repeat this process iteratively for each element of the strategy list, in order. For example, if `nonredundant_drivers = [[{ 'xD':1, 'xE':1 }], [{ 'xA':1 }, { 'xB':1 }], [{ 'xC':1 }]]` then there are two control strategies available: 1) fix `xD=xE=1` temporarily and 2) first fix either `xA=1` or `xB=1` temporarily, then fix `xC=1` temporarily.

target_method = history, **driver_method** = GRASP: The same as history, minimal, except external driver nodes are searched for using the GRASP algorithm using **GRASP_iterations** iterations.

target_method = merge, **driver_method** = internal: Finds all shortest stable motif histories that result in the target node states being logically fixed. All node states in the motifs in the history are merged into a stable module dictionary. This is then searched for internal driver nodes. Each element of the return value is a dictionary corresponding to a control set. Each control set eventually becomes self-sustaining.

target_method = merge, **driver_method** = minimal: Similar to the merge method, except the search for drivers is conducted over all nodes, not just those internal to the merged stable module. Furthermore, the search is truncated when a control set is found such that the search does not proceed to driver sets larger than the smallest found. Each element of the return value is a dictionary corresponding to a control set. The control sets are only guaranteed to result in activation of the target if they are temporary interventions.

target_method = merge, **driver_method** = GRASP: The same as merge, minimal, except external driver nodes are searched for using the GRASP algorithm using **GRASP_iterations** iterations.

Module `StableMotifs.TimeReversal`

Functions

Function `time_reverse_primes`

```
def time_reverse_primes(  
    primes  
)
```

Computes the time reversal of the input system (under general asynchronous update). The time reverse system has the same STG as the input system, but with each edge reversed.

Parameters

`primes` : **PyBoolNet prime dictionary** System update rules.

Returns

`trprimes` : **PyBoolNet prime dictionary** Time-reversed system update rules.

Generated by *pdoc* 0.9.1 (<https://pdoc3.github.io>).