iglsynth Documentation

Release 0.2.1

Abhishek N. Kulkarni

Nov 20, 2019

IGLSynth Documentation

IGLSynth is a high-level Python API for solving Infinite Games and Logic-based strategy Synthesis. It provides an easy interface to

1. Define two-player games-on-graphs.
2. Assign tasks to players using formal logic.
3. Write solvers to compute winning strategies in the game.

IGLSynth consists of 4 modules,

1. `game`: Defines classes representing deterministic/stochastic and concurrent/turn-based games as well as hyper-games.
2. *`logic`*: Defines classes representing formal logic such as Propositional Logic, Linear Temporal Logic etc.
3. `solver`: Defines solvers for different games such as ZielonkaSolver etc.
4. *`util`*: Defines commonly used classes such as Graph.

CHAPTER 1

Indices and tables

1.1 IGLSynth Examples

1.1.1 Game Graph

Hello

1.2 IGLSynth API

1.2.1 Game Module API

Overview

Will be written..

Game Module

Global Variables

`iglsynth.game.core.`**`TURN_BASED = 'Turn-based'`**
Macro to define concurrent transition system and game.

Action

class iglsynth.game.core.**Action**(*name=None*, *func=None*)
Bases: object

Represents an action. An action acts on a state (of TSys or Game etc.) to produce a new state.

Parameters

- **name** – (str) Name of the action.
- **func** – (function) An implementation of action.

Note: Acceptable function templates are,

- st <- func(st)
- st <- func(st, *args)
- st <- func(st, **kwargs)
- st <- func(st, *args, **kwargs)

Game Module

Two-Player Deterministic Game

1.2.2 Logic Module API

Logic Module

in progress...

Logic Module

Global Variables

iglsynth.logic.core.**TRUE = AP(name=true)**

iglsynth.logic.core.**FALSE = AP(name=false)**

ILogic (Interface Class)

SyntaxTree

Atomic Propositions

Alphabet

Propositional Logic Formulas

Linear Temporal Logic

1.2.3 Utilities API

Utilities

in proigress..

Utilities

Graph Class

class iglsynth.util.graph.**Graph**(*vtype=None*, *etype=None*, *graph=None*, *file=None*)

Base class to represent graph-based objects in IGLSynth.

- *Graph* may represent a Digraph or a Multi-Digraph.
- *Graph.Edge* may represent a self-loop, i.e. *source* = *target*.
- *Graph* stores objects of *Graph.Vertex* and *Graph.Edge* classes or their sub-classes, which users may define.
- *Graph.Vertex* and *Graph.Edge* may have attributes, which represent the vertex and edge properties of the graph.

Parameters

- **vtype** – (class) Class representing vertex objects.
- **etype** – (class) Class representing edge objects.
- **graph** – (*Graph*) Copy constructor. Copies the input graph into new Graph object.
- **file** – (str) Name of file (with absolute path) from which to load the graph.

Todo: The copy-constructor and load-from-file functionality.

---

class Edge(*u: iglsynth.util.graph.Graph.Vertex*, *v: iglsynth.util.graph.Graph.Vertex*)

Base class for representing a edge of graph.

- *Graph.Edge* represents a directed edge.
- Two edges are equal, if the two *Graph.Edge* objects are same.

Parameters

- **u** – (*Graph.Vertex* or its sub-class) Source vertex of edge.
- **v** – (*Graph.Vertex* or its sub-class) Target vertex of edge.

source
Returns the source vertex of edge.

target
Returns the target vertex of edge.

class Vertex
Base class for representing a vertex of graph.

- *Graph.Vertex* constructor takes no arguments.
- Two vertices are equal, if the two *Graph.Vertex* objects are same.

add_edge(*e: iglsynth.util.graph.Graph.Edge*)
Adds an edge to the graph. Both the vertices must be present in the graph.

Raises

- **AttributeError** – When at least one of the vertex is not in the graph.
- **AssertionError** – When argument *e* is not an *Graph.Edge* object.

add_edges(*ebunch: Iterable[Graph.Edge]*)
Adds a bunch of edges to the graph. Both the vertices of all edges must be present in the graph.

Raises

- **AttributeError** – When at least one of the vertex is not in the graph.
- **AssertionError** – When argument *e* is not an *Graph.Edge* object.

add_vertex(*v: iglsynth.util.graph.Graph.Vertex*)
Adds a new vertex to graph. An attempt to add existing vertex will be ignored, with a warning.

Parameters **v** – (*Graph.Vertex*) Vertex to be added to graph.

add_vertices(*vbunch: Iterable[Graph.Vertex]*)
Adds a bunch of vertices to graph. An attempt to add existing vertex will be ignored, with a warning.

Parameters **vbunch** – (Iterable over *Graph.Vertex*) Vertices to be added to graph.

edges
Returns an iterator over edges in graph.

in_edges(*v: Union[Graph.Vertex, Iterable[Graph.Vertex]]*)
Returns an iterator over incoming edges to given vertex or vertices. In case of vertices, the iterator is defined over the union of set of incoming edges of individual vertices.

Parameters **v** – (*Graph.Vertex*) Vertex of graph.

Raises **AssertionError** – When *v* is neither a *Graph.Vertex* object nor an iterable of *Graph.Vertex* objects.

in_neighbors(*v: Union[Graph.Vertex, Iterable[Graph.Vertex]]*)
Returns an iterator over sources of incoming edges to given vertex or vertices. In case of vertices, the iterator is defined over the union of set of incoming edges of individual vertices.

Parameters **v** – (*Graph.Vertex*) Vertex of graph.

Raises **AssertionError** – When *v* is neither a *Graph.Vertex* object nor an iterable of *Graph.Vertex* objects.

num_edges
Returns the number of edges in graph.

num_vertices
Returns the number of vertices in graph.

out_edges (*v: Union[Graph.Vertex, Iterable[Graph.Vertex]]*)
Returns an iterator over outgoing edges to given vertex or vertices. In case of vertices, the iterator is defined over the union of set of incoming edges of individual vertices.

Parameters **v** – (*Graph.Vertex*) Vertex of graph.

Raises **AssertionError** – When *v* is neither a *Graph.Vertex* object nor an iterable of *Graph.Vertex* objects.

out_neighbors (*v: Union[Graph.Vertex, Iterable[Graph.Vertex]]*)
Returns an iterator over targets of incoming edges to given vertex or vertices. In case of vertices, the iterator is defined over the union of set of incoming edges of individual vertices.

Parameters **v** – (*Graph.Vertex*) Vertex of graph.

Raises **AssertionError** – When *v* is neither a *Graph.Vertex* object nor an iterable of *Graph.Vertex* objects.

rm_edge (*e: iglsynth.util.graph.Graph.Edge*)
Removes an edge from the graph. An attempt to remove a non-existing edge will be ignored, with a warning.

Parameters **e** – (*Graph.Edge*) Edge to be removed.

rm_edges (*ebunch: Iterable[Graph.Edge]*)
Removes a bunch of edges from the graph. An attempt to remove a non-existing edge will be ignored, with a warning.

Parameters **ebunch** – (Iterable over *Graph.Edge*) Edges to be removed.

rm_vertex (*v: iglsynth.util.graph.Graph.Vertex*)
Removes a vertex from the graph. An attempt to remove a non-existing vertex will be ignored, with a warning.

Parameters **v** – (*Graph.Vertex*) Vertex to be removed.

rm_vertices (*vbunch: Iterable[Graph.Vertex]*)
Removes a bunch of vertices from the graph. An attempt to remove a non-existing vertex will be ignored, with a warning.

Parameters **vbunch** – (Iterable over *Graph.Vertex*) Vertices to be removed.

vertices
Returns an iterator over vertices in graph.

Utilities

Spot Formulas

Current release and documentation update date:

- genindex
- modindex
- search

Python Module Index

e

i

Index