wfomi

Release 0.1

korecki

May 30, 2020

CONTENTS:

1 Introduction **1**

2 Input files **2**
- 2.1 Weights file 2
- 2.2 Cicuit file 2

3 Code **4**
- 3.1 wfomi 4
- 3.2 term 4
- 3.3 circuit 5
- 3.4 parser 9

Python Module Index **11**

Index **12**

CHAPTER

ONE

INTRODUCTION

This is the documentation for the wfomi solver written in python3. Wfomi stands for weighted first orde model integration. The solver implements a novel algorithm based on the work of Jonathan Feldstein. The solver outperforms the wfomc solver - Forclift as well as wmi solver - pywmi in terms of efficiency and expressiveness.

To run the solver, change directory to the solver folder and call:

```
python3 pywfomi.py [circuit1_filename] [circuit2_filename] [weight_filename]
```

without the brackets and with the intended filenames

circuit 1 represents the partition circuit so the one containing the entire theory
circuit 2 represents the query circuit
weights represent the simple and complex weights corresponding to predicates appearing the circuits of interest

Example call would look like this:

```
python3 pywfomi.py test_input/smokers/theory.txt test_input/smokers/query.txt test_
↪input/smokers/weights_simple.txt
```

The docs folder contains this documentation. The test_input folder contains examples that can be run to test the software.

CHAPTER

TWO

INPUT FILES

The solver requires two kinds of input files. The two files representing the circuits, the theory circuit and the query circuit and the one file representing the weights of the predicates occuring in the two cicuits. In this section we present the syntax of these files. The default parser included in the solver works with the here explained file formats, however it should be easy to extend or introduce new parsers for different file formats.

2.1 Weights file

The weights file contains the weights corresponding to the predicates included in the circuit files. In the weights file there can be 3 types of lines: | the domain line eg. 'person = {Alice}' | the simple weight line eg. 'pre: [1, 1] const[1, 10]', meaning the predicate pre is assigned weight 1*1 and its negation is assigned weight 1*10 | the complex weight line eg. 'bmi(x)fun x**2 + 10 bounds[5, 10] const[1, 10]' Note that for complex weights the negation weight has to be specified seperately eg. 'neg bmi(x)fun x**2 + 10 bounds[10, 20]'. The name of the arguments of the weight functions must correspond to the argument names used in the circuit description. The const[1, 5] indicates the constant multiplier on the weight function and if omitted defaults to 1. This is used for computational speed up.

An example of the weights file follows. Additional examples can be found in test_input folder.

```
person = {Guy, Nima, Wannes, Jesse, Luc}
friends: [0.1, 0.9]
smokes(b)fun -0.001*(b-27)**2+0.3  bounds[35, 45]
neg smokes(b)fun -0.001*(b-27)**2+0.3  bounds[10, 35]
f_1: [7.38905609893065, 1]
```

2.2 Cicuit file

The circuit files contain the theory and query circuits. There are two types of lines in the circuit file, one corresponding to the contents of the given node and the other indicating the connections between nodes. These lines can be intermixed but for readibility it is customary to first write the contents lines and then the connections lines. The lines always begin with the nX eg. n1 where X indicates the unique node number which is used to identify it.

The contens lines are then followed by the description of the node. This can be a connective 'and', 'or', a leaf including the given predicate eg. 'smokes(x)', a quantifier eg. A{x}{persons} or a constant C{x}{persons}. The quantifier and constant lines are of the form Z{x}{persons}, where Z can be A or E indicating a universal or existentiar quantifiers. The first braces store the variable(s) that is quantified over and the second braces store the domain of the variable. If the node quantifies over more than one variable they are listed seperated by commas and so are the domains like: C{x, y}{persons, animals}. Moreover if the domain of the quantifier is constrained not to include a given object we denote it by E{x}{persons/Alice} where persons normally include Alice. Futhermore the type of the domain can be included, eg. if a given quantifier is a descendant of an existential quantifier it must refer to one of the splits of the original domain induced by the existential. Those splits are reffered to as top and bot and are indicated like: A{x}{persons-bot}

The connections lines are of the form nX -> xY, eg. n0 -> n1 indicating n1 is the child of n0.

An example of the circuit file follows. Additional examples can be found in test_input folder.

```
n23 and
n0  C{X}{person} friends(X,X) or neg friends(X,X)
n22 E{x}{person}
n21 and
n1  C{X,Y}{person-bot/Y, person-bot} friends(X,Y) or neg friends(X,Y)
n20 and
n2  C{X,Y}{person/Y, person-top} friends(X,Y) or neg friends(X,Y)
n19 and
n3  C{X}{person-top} smokes(X)
n18 and
n4  C{X,Y}{person, person-top} f_1(X,Y)
n17 and
n5  C{X}{person-bot} neg smokes(X)
n16 and
n6  C{X,Y}{person-bot, person-bot} f_1(X,Y)
n15 A{x}{person-top}
n14 A{y}{person-bot}
n13 or
n9 and
n7  f_1(x,y)
n8  neg friends(x,y)
n12 and
n10  neg f_1(x,y)
n11  friends(x,y)
n23 -> n0;
n23 -> n22;
n22 -> n21;
n21 -> n1;
n21 -> n20;
n20 -> n2;
n20 -> n19;
n19 -> n3;
n19 -> n18;
n18 -> n4;
n18 -> n17;
n17 -> n5;
n17 -> n16;
n16 -> n6;
n16 -> n15;
n15 -> n14;
n14 -> n13;
n13 -> n9;
n13 -> n12;
n9 -> n7;
n9 -> n8;
n12 -> n10;
n12 -> n11;
```

CHAPTER

THREE

CODE

Here follows the documentation of the python code of the solver.

3.1 wfomi

`wfomi.`**`wfomi`**(*partitionFile=None*, *queryFile=None*, *weightFile=None*)

the main function of the solver, takes the theory circuit, query circuit and the weight files as arguments from the command line in that order and returns the probability of the query or as arguments if called as a function from a different module

Parameters

- **`partitionFile`** – the path to the file contiaining the theory circuit which will be used to compute the partition function
- **`queryFile`** – the path to the file containing the query circuit which will be used to compute unnormalised probability of the query
- **`weightFile`** – the path to the weight file containing the weights of the predicates in the theory and query circuits

Returns the probability of the query. In addition the time to compute is printed along with the partition function and unnormalised query

3.2 term

`class` `term.`**`Term`**(*weights=None*, *bounds=None*, *const=None*)

The Term object represent the smallest computational unit over the circuit representation. The Term is used to store the weight functions in symbolic form, the associated bounds and the constant multiplier. The term implements multiplication and addition as well as integration. The weights, bounds and constants are all lists and their elements corresponding to elements of a sum.

`__add__`(*other*)

Implements addition for two terms.

Parameters **`other`** – the right hand side *`Term`* of addition

Returns the *`Term`* representing the sum of two terms

`__init__`(*weights=None*, *bounds=None*, *const=None*)

initialises the term object with specified weights, bounds and constant multiplier

Parameters

- **weights** – the weight function
- **bounds** – the bounds of the weight function containing the integrated variable, the lower and upper bounds in that order
- **const** – the constant multiplier

__mul__(*other*)
Implements multiplication for two terms taking care of bounds of functions of the same variables.

Parameters **other** – the right hand side *Term* of multiplication

Returns the *Term* representing the multiplied terms

__str__()
prints the term

integrate()
Implements efficient integation of a term.

Returns the *Term* with constant multiplier 1, empty bounds and the numerical value of the integrated constituent terms summed

term.**integrateFromDict**(*weight*, *bounds*)
helper function for the integration

Parameters

- **weight** – the weight function
- **bounds** – the bounds of the weight function

Retuns the numeric value of the integrated weight function

term.**symbolicToNumeric**(*weight*, *bounds*)
helper function for the integration

Parameters

- **weight** – the weight function
- **bounds** – the bounds of the weight function

Retuns the numeric value of the integal within the given bounds

3.3 circuit

each node has a compute class which follows the computation step of the algorithm for the given node the maxDomainSize is used to compute the maximum domain size for the existential node .. moduleauthor:: Marcin Korecki

class circuit.**AndNode**(*left=None*, *right=None*)

compute(*domSize=None*, *removed=None*)
computes the symbolic value at the and node by multiplying two terms at its child nodes. the domSize and removed are passed for potential existential and universals that may be the node's descendants

Parameters

- **domSize** – the size of the domain of the ancestor quantifiers
- **removed** – the list of objects removed from the domain

Returns the *Term* representing the multiplication of the values at the child nodes of the or node

maxDomainSize()

used to compute the maximum domain size for the existential node, the maxDomain is the larger domain out of the domains of the left and right children

Returns the maximum size of the domain at the given node and the set of objects removed from it

class circuit.**ConstNode**(*data=None*, *nodeName=None*, *varList=None*, *domData=None*)

__init__(*data=None*, *nodeName=None*, *varList=None*, *domData=None*)

initialises the constant node with the

Parameters

- **data** – represents the type of a constant node (and, or, leaf)
- **nodeName** – used as a key in a dictionary storing the domains of the nodes
- **varList** – the list of variables the constant node deals with
- **domData** – the dictionary containing the data on the domains corresponding to variables, the data stores 3 values for

each variable, the list of objects in the domain, the domain type (top or bot depending on the existential split) and the without set representing the objects removed from the domain

compute(*domSize=None*, *removed=None*)

Computes the symbolic value at the constant node depending on its type

Parameters

- **domSize** – the size of the domain of the ancestor quantifiers
- **removed** – the list of objects removed from the domain

Returns the *Term* representing the value at the constant node which is eithe symbolic if the node is under a universal

node quantifying over the same domain or otherwise numeric corresponding to universal quantification over its domain

maxDomainSize()

used to compute the maximum domain size for the existential node based on the largest domain of the variables of the constant node

Returns the maximum size of the domain at the given node and the set of objects removed from it

class circuit.**ExistsNode**(*var=None*, *domData=None*)

__init__(*var=None*, *domData=None*)

initialises the existential node with the variables it quantifies over and the data on the corresponding domains]

Parameters

- **var** – the name of the variable that the universal quantifies over
- **domData** – the dictionary containing the data on the domains corresponding to variables, the data stores 3 values for

each variable, the list of objects in the domain, the domain type (top or bot depending on the existential split) and the without set representing the objects removed from the domain

compute(*domSize=None*, *removed=None*)
computes the symbolic value at the existential node based on the size of the domain it quantifies over and taking into account the objects removed from it

Parameters

- **domSize** – the size of the domain the existential is quantifying over
- **removed** – the list of objects removed from the domain

Returns the `Term` representing the value at the existential node

maxDomainSize()
used to compute the maximum domain size for the existential node

Returns the maximum size of the domain at the given node and the set of objects removed from it

class circuit.**ForAllNode**(*var=None*, *domData=None*)

__init__(*var=None*, *domData=None*)

initialises the universal node with the variables it quantifies over and the data on the domain that the variables correspond to

Parameters

- **var** – the name of the variable that the universal quantifies over
- **domData** – the dictionary containing the data on the domains corresponding to variables, the data stores 3 values for

each variable, the list of objects in the domain, the domain type (top or bot depending on the existential split) and the without set representing the objects removed from the domain

compute(*domSize=None*, *removed=None*)
computes the numerical value at the universal node based on the size of the domain it quantifies over taking into account the objects that have been removed from it

Parameters

- **domSize** – the size of the domain the universal is quantifying over
- **removed** – the list of objects removed from the domain

Returns the `Term` with a cosntant equal to 1, empty bounds and the result of the integration at the universal node

raised to the power of its domain size

maxDomainSize()
used to compute the maximum domain size for the existential node

Returns the maximum size of the domain at the given node and the set of objects removed from it

class circuit.**LeafNode**(*data=None*, *weights=None*)

__init__(*data=None*, *weights=None*)
initialises the leaf node

Parameters

- **data** – the key for the dictionary containing the weights
- **weights** – a dictionary containing all the data pertaining to the weights of a predicate. In case of simple weights it is a

single value. In case of complex weights it is the symbolic weight function, the bounds, the arguments of the function and the constant multiplier in that order

compute (*domSize=None*, *removed=None*)

computes the symbolic value at the leaves depending on weather the corresponding weight is a float or a function

Parameters

- **domSize** – the size of the domain of the ancestor quantifiers
- **removed** – the list of objects removed from the domain

Retuns the term corresponding to the weight function of the predicate at the node

maxDomainSize ()

used to compute the maximum domain size for the existential node

Returns 0 as the domain of the leaf node is empty as it does not quantify over anything

class circuit.**Node** (*left=None*, *right=None*)

The base class defining a node, all other nodes inherit from it

__init__ (*left=None*, *right=None*)

Initialize self. See help(type(self)) for accurate signature.

maxDomainSize ()

used to compute the maximum domain size for the existential node

class circuit.**OrNode** (*left=None*, *right=None*)

compute (*domSize=None*, *removed=None*)

computes the symbolic value at the or node by adding two terms at its child nodes, the setsize and removed are passed for potential existential and universals that may be the node's descendants

Parameters

- **domSize** – the size of the domain of the ancestor quantifiers
- **removed** – the list of objects removed from the domain

Returns the *Term* representing the sum of the values at the child nodes of the or node

maxDomainSize ()

used to compute the maximum domain size for the existential node, the maxDomain is the larger domain out of the domains of the left and right children

Returns the maximum size of the domain at the given node and the set of objects removed from it

3.4 parser

class parser.**Parser**

the parser for the default circuit and weight files defined by the author. The files descriptions can be found in the docs

__init__()

the parsers stores 3 regex patterns used to detect lines containing node data, which is split into node number and node data, and link data. It also stores the forward and backward connections as well as the created nodes as dictionaries

adjustConstNodes(*constCorrection*)

adjusts the circuit by moving the constant nodes down when they are above a univesal quantifier over the same domain, adjusts the moved nodes to not integrate on themselves during wfomi computation

Parameters **constCorrection** – the list storing the constant nodes for possible adjustment by `adjustConstNodes()`

ancestorIsForAll(*node*)

a helper function used in adjustConstNodes to check if the node has a universal quantifier as an ancestor

Parameters **node** – the name of the node for which we want to check if one of its ancestors is a universal quantifier

Returns None if there are no universal ancestors or the first universal ancestor node object

connectNodes()

connects the nodes in self.nodes dictionary based on data in self.connections

nextMatchingForAll(*node*, *domain*)

a helper function used in adjustConstNodes to detect the next universal quantifier of a given node with matching domain

Parameters

- **node** – the name of the node for which we want to check if one of its ancestors is a universal quantifier
- **domain** – the domain of the node

Returns None if there are no matching universal quantifier or the matchin node name

parseCircuit(*name*, *weights*, *domains*)

parses a circuit file with the given name and creates nodes using data on the weight functions and domains

Parameters

- **name** – the path to the circuit file to be parsed
- **weights** – the weights dictionary as returned by the `parseWeights()`
- **domains** – the domains dictionary as returned by the `parseWeights()`

Returns the root node and the dictionary of all nodes

parseConnections(*content*)

parses the link lines of a circuit file with the given name and stores the connections between nodes in self.connections, and self.reverseConnections

Parameters **content** – the contents of the file as read by `parseCircuit()`

parseConst(*line*, *constCorrection*, *weights*, *domains*, *node*)

parses a line contianing a constant node

Parameters

- **`line`** – the contents of the line containing the quantifier as read by *`parseCircuit()`*
- **`constCorrection`** – the list storing the constant nodes for possible adjustment by *`adjustConstNodes()`*
- **`weights`** – the weights dictionary as returned by the *`parseWeights()`*
- **`domains`** – the domains dictionary as returned by the *`parseWeights()`*
- **`node`** – the constant node name

`parseNodes`(*content*, *constCorrection*, *weights*, *domains*)

parses the node lines of a circuit file with the given name, creates and stores the nodes in the nodes dictionary

Parameters

- **`content`** – the contents of the file as read by *`parseCircuit()`*
- **`constCorrection`** – the list storing the constant nodes for possible adjustment by *`adjustConstNodes()`*
- **`weights`** – the weights dictionary as returned by the *`parseWeights()`*
- **`domains`** – the domains dictionary as returned by the *`parseWeights()`*

`parseQuantifier`(*line*, *domains*)

parses a line contianing a universal or existential quantifier

Parameters

- **`line`** – the contents of the line containing the quantifier as read by *`parseCircuit()`*
- **`domains`** – the domains dictionary as returned by the *`parseWeights()`*

Returns objects contained in the domain and the corresponding variable name

`parseWeights`(*name*)

parses the weight file

Parameters **`name`** – the path to the weight file to be parsed

Retuns dictionaries containing the weights and domains of all the predicates

PYTHON MODULE INDEX

c

circuit, 5

p

parser, 9

t

term, 4

w

wfomi, 4

INDEX

Symbols

__add__() (*term.Term method*), 4
__init__() (*circuit.ConstNode method*), 6
__init__() (*circuit.ExistsNode method*), 6
__init__() (*circuit.ForAllNode method*), 7
__init__() (*circuit.LeafNode method*), 7
__init__() (*circuit.Node method*), 8
__init__() (*parser.Parser method*), 9
__init__() (*term.Term method*), 4
__mul__() (*term.Term method*), 5
__str__() (*term.Term method*), 5

A

adjustConstNodes() (*parser.Parser method*), 9
ancestorIsForAll() (*parser.Parser method*), 9
AndNode (*class in circuit*), 5

C

circuit
    module, 5
compute() (*circuit.AndNode method*), 5
compute() (*circuit.ConstNode method*), 6
compute() (*circuit.ExistsNode method*), 7
compute() (*circuit.ForAllNode method*), 7
compute() (*circuit.LeafNode method*), 8
compute() (*circuit.OrNode method*), 8
connectNodes() (*parser.Parser method*), 9
ConstNode (*class in circuit*), 6

E

ExistsNode (*class in circuit*), 6

F

ForAllNode (*class in circuit*), 7

I

integrate() (*term.Term method*), 5
integrateFromDict() (*in module term*), 5

L

LeafNode (*class in circuit*), 7

M

maxDomainSize() (*circuit.AndNode method*), 6
maxDomainSize() (*circuit.ConstNode method*), 6
maxDomainSize() (*circuit.ExistsNode method*), 7
maxDomainSize() (*circuit.ForAllNode method*), 7
maxDomainSize() (*circuit.LeafNode method*), 8
maxDomainSize() (*circuit.Node method*), 8
maxDomainSize() (*circuit.OrNode method*), 8
module
    circuit, 5
    parser, 9
    term, 4
    wfomi, 4

N

nextMatchingForAll() (*parser.Parser method*), 9
Node (*class in circuit*), 8

O

OrNode (*class in circuit*), 8

P

parseCircuit() (*parser.Parser method*), 9
parseConnections() (*parser.Parser method*), 9
parseConst() (*parser.Parser method*), 9
parseNodes() (*parser.Parser method*), 10
parseQuantifier() (*parser.Parser method*), 10
parser
    module, 9
Parser (*class in parser*), 9
parseWeights() (*parser.Parser method*), 10

S

symbolicToNumeric() (*in module term*), 5

T

term
    module, 4
Term (*class in term*), 4

W

wfomi

module, 4
wfomi() (*in module wfomi*), 4