WCS Reasoner

User & Developer Guide

December 2021

Table of Contents

User Guide

Overview

WCS Reasoner is an application for solving datalog programs under the Weak Completion Semantics. It works with Łukasiewicz three-valued logic. The application performs the weak completion of a program, finds its least fixed point using the semantic operator, and performs skeptical abduction. In the simplest terms, this is a glorified calculator for a special kind of logic.

Use this app to quickly crunch a WCS problem for your thesis or paper and paste the solution in your LaTeX document. The app can also be a great educational tool for university students because they can follow each step of the process, check their solutions and play around with modifying the input to see how it affects the results

WCS Reasoner was created for Steffen Hölldobler to accompany his manuscript “Human Reasoning and the Weak Completion Semantics”. Example programs, found in the menu bar, are taken from the manuscript and related works.

∧	⊤	U	⊥
⊤	⊤	U	⊥
U	U	U	⊥
⊥	⊥	⊥	⊥

∨	⊤	U	⊥
⊤	⊤	⊤	⊤
U	⊤	U	U
⊥	⊤	U	⊥

↔	⊤	U	⊥
⊤	⊤	U	⊥
U	U	⊤	U
⊥	⊥	U	⊤

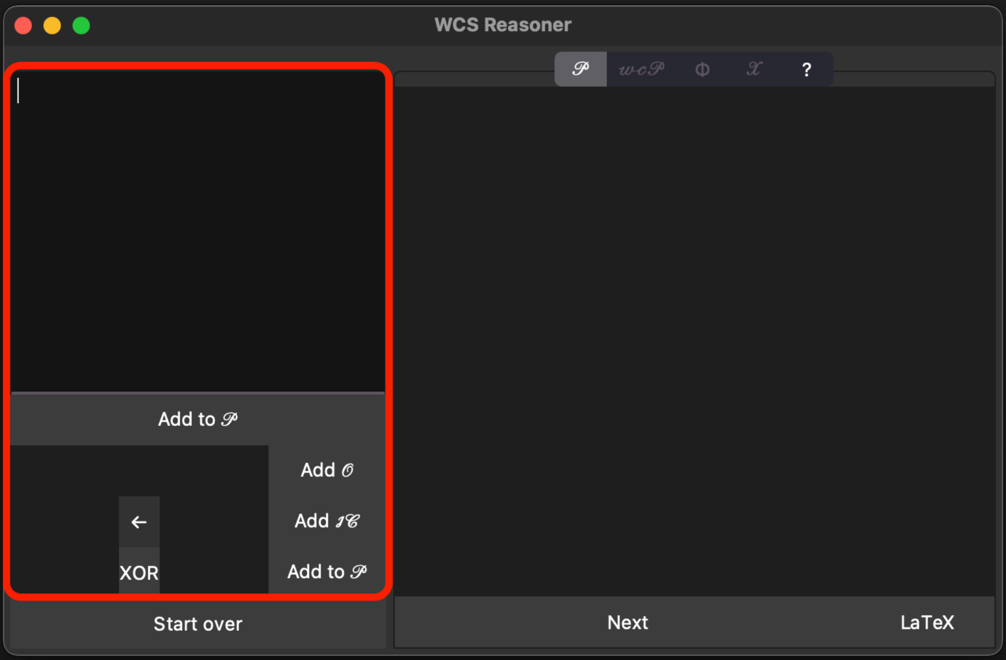
F	⊤	⊥	U
¬F	⊥	⊤	U

F	⊤	⊥	U
ctxtF	⊤	⊥	⊥

Supported Łukasiewicz truth tables

Input

You can input the program, observations, integrity constraints, and disjunctions with the corresponding boxes on the left side of the app window.



Input a datalog program clause by clause or add multiple clauses by separating them with a semicolon ‘;’. Any string beginning with a capital letter is considered a variable. Any predicate beginning with ‘ab’ is considered an abnormality predicate. Each clause must be of the form ‘head if body’.

There are some reserved strings that are recognised interchangeably:

Use the asterisk ‘*’ in the head of the clause to classify it as a non-necessary antecedent.

if	:-	-:	←
and	&	^	∧
not	~	!	¬
or	|	∨	
T	⊤		
F	⊥		
U	∪		

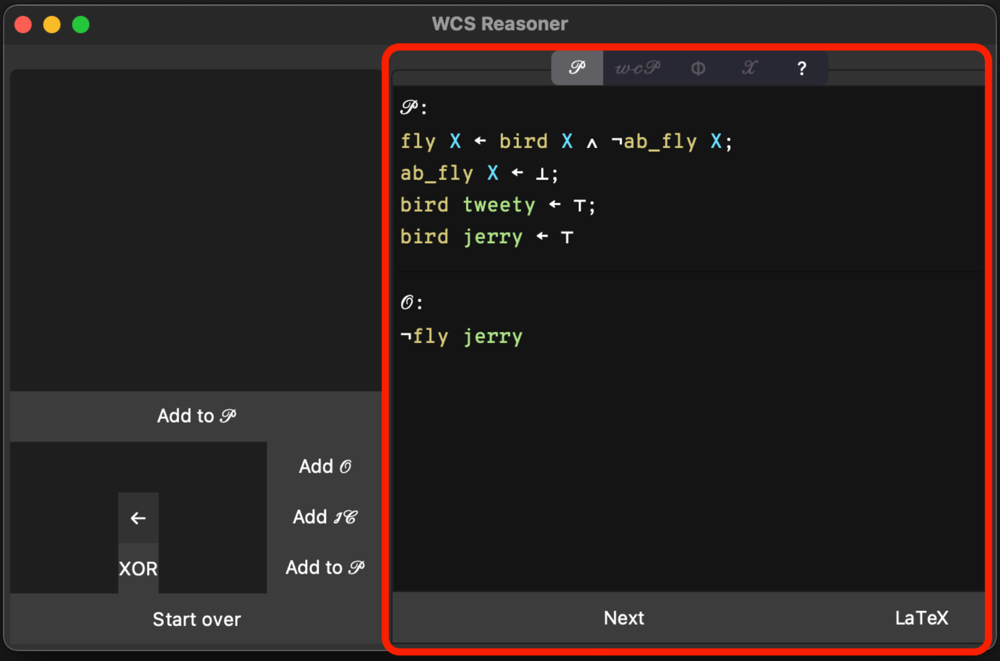
Interchangeable reserved strings

Similarly, use the asterisk in the body of the clause to classify it as a factual conditional.

The context operator can be input with the reserved string ‘ctxt’.

Reasoning Steps

The output of the program is shown on the right. Each tab represents a reasoning step: first you see the given program, then its weak completion, then the least fixed point, and finally, skeptical abduction.

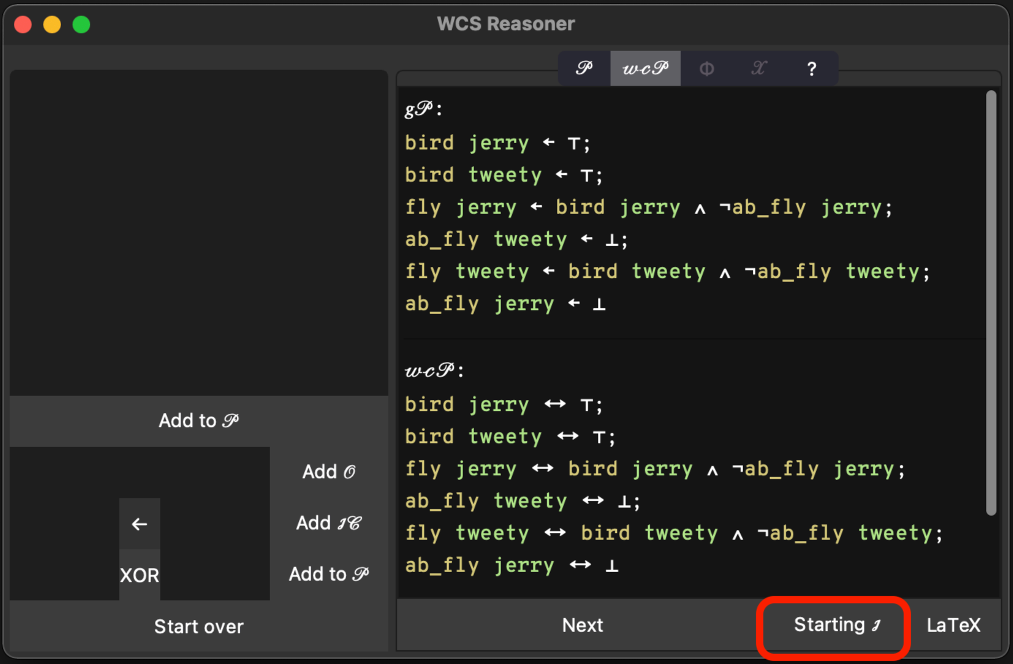


```
𝒫:
fly X ← bird X ∧ ¬ab_fly X;
ab_fly X ← ⊥;
bird tweety ← ⊤;
bird jerry ← ⊤

𝒪:
¬fly jerry
```

After all of your clauses, observations and integrity constraints are entered, click the Next button to ground the program, weakly complete it, and generate the set of abducibles.

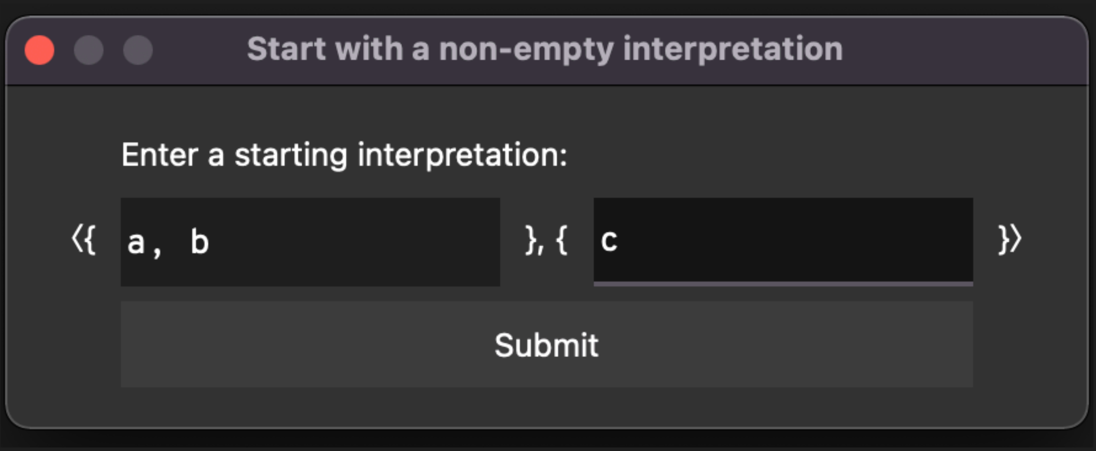
To start the semantic operator with the empty interpretation, click Next. If you need to specify another starting interpretation, click Starting $\mathcal{I}$



```
g𝒫:
bird jerry ← ⊤;
bird tweety ← ⊤;
fly jerry ← bird jerry ∧ ¬ab_fly jerry;
ab_fly tweety ← ⊥;
fly tweety ← bird tweety ∧ ¬ab_fly tweety;
ab_fly jerry ← ⊥

wc𝒫:
bird jerry ↔ ⊤;
bird tweety ↔ ⊤;
fly jerry ↔ bird jerry ∧ ¬ab_fly jerry;
ab_fly tweety ↔ ⊥;
fly tweety ↔ bird tweety ∧ ¬ab_fly tweety;
ab_fly jerry ↔ ⊥
```

Enter the starting interpretation's true terms on the left and false terms on the right



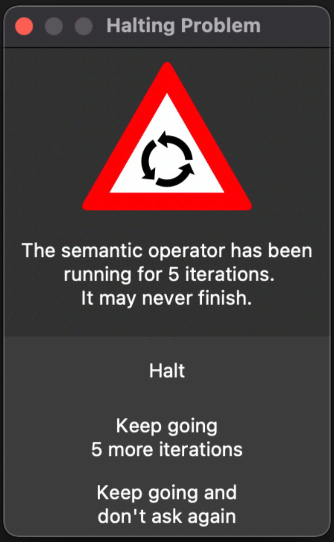
Start with a non-empty interpretation

Enter a starting interpretation:

⟨{ a, b }, { c }⟩

Submit

The semantic operator will iteratively build new interpretations until it gets two identical interpretations in a row, i.e until it finds the fixed point. In some cases, there is no fixed point and the semantic operator will never finish. For example, try the non-monotonic program from the Context menu bar item. After 5 iterations, you will be prompted to halt the semantic operator yourself. If you are not sure that there is a fixed point, you may run Phi for another 5 iterations. If you are confident that Phi will eventually finish, you may disable the prompt. You are taking the risk of getting into an infinite loop and having to have to force quit the app.



Halting Problem

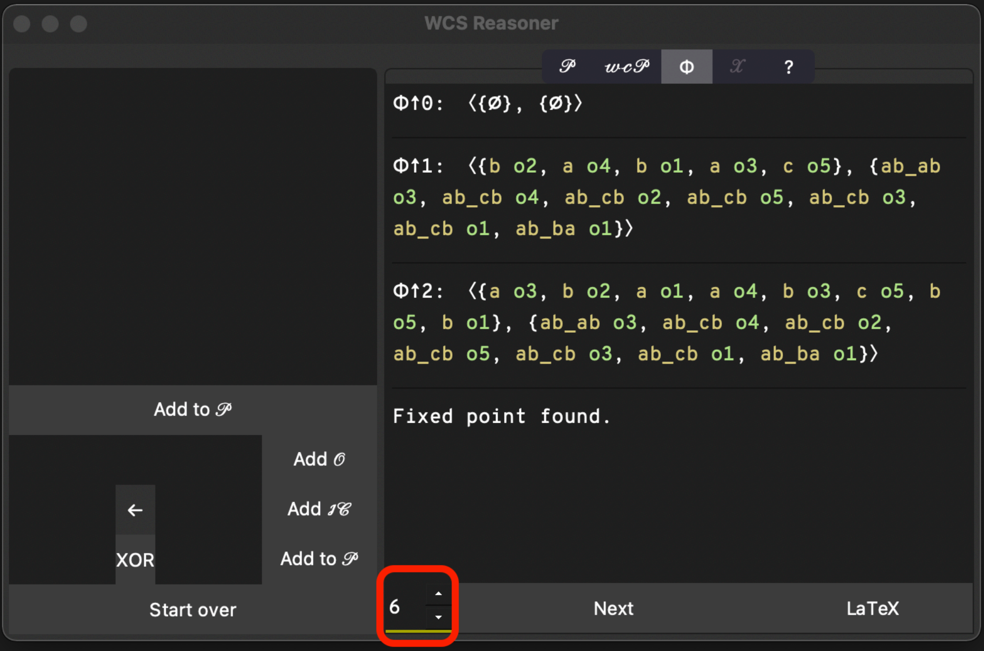
The semantic operator has been running for 5 iterations. It may never finish.

Φ	$I^\top$	$I^\perp$
$\uparrow 0$		
$\uparrow 1$	penguin tweety, bird tweety	ab_wings tweety, ab_fly tweety
$\uparrow 2$	penguin tweety, ab_fly tweety, wings tweety, bird tweety, fly tweety	ab_wings tweety
$\uparrow 3$	penguin tweety, ab_fly tweety, wings tweety, bird tweety	ab_wings tweety, fly tweety

The LaTeX output of the semantic operator is a 'tabular' table that uses the 'array' package. Don't forget to use the package in your LaTeX project.

Each row represents an iteration of the semantic operator. The true terms of an interpretation are shown in the second column, and the false terms are in the third.

To perform skeptical abduction, one adds every combination of abducibles from the set to the program and runs Phi for it. This means, the app runs Phi for each element of the power set of the set of abducibles. A power set can get very large very quickly, so the app limits the length of each explanation to a reasonable number. If you wish to consider explanations with more abducibles in each, you may raise the value in the spin box. Be aware that it may take the app some time to look at many thousands of possible explanations.



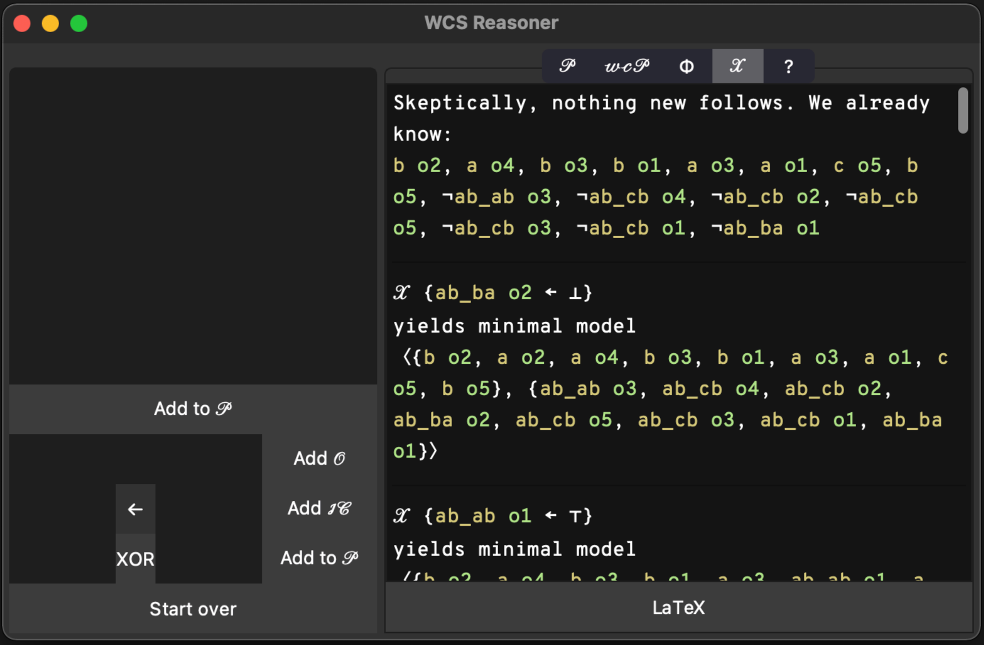
```
Φ↑0:  ⟨{∅}, {∅}⟩

Φ↑1:  ⟨{b o2, a o4, b o1, a o3, c o5}, {ab_ab
o3, ab_cb o4, ab_cb o2, ab_cb o5, ab_cb o3,
ab_cb o1, ab_ba o1}⟩

Φ↑2:  ⟨{a o3, b o2, a o1, a o4, b o3, c o5, b
o5, b o1}, {ab_ab o3, ab_cb o4, ab_cb o2,
ab_cb o5, ab_cb o3, ab_cb o1, ab_ba o1}⟩

Fixed point found.
```

The results of skeptical abduction are summarised in the first paragraph. Then, each valid minimal model is presented with the explanation that yielded it.



```
Skeptically, nothing new follows. We already
know:
b o2, a o4, b o3, b o1, a o3, a o1, c o5, b
o5, ¬ab_ab o3, ¬ab_cb o4, ¬ab_cb o2, ¬ab_cb
o5, ¬ab_cb o3, ¬ab_cb o1, ¬ab_ba o1

𝒳 {ab_ba o2 ← ⊥}
yields minimal model
 ⟨{b o2, a o2, a o4, b o3, b o1, a o3, a o1, c
o5, b o5}, {ab_ab o3, ab_cb o4, ab_cb o2,
ab_ba o2, ab_cb o5, ab_cb o3, ab_cb o1, ab_ba
o1}⟩

𝒳 {ab_ab o1 ← ⊤}
yields minimal model
 ⟨{b o2, a o4, b o3, b o1, a o3, ab_ab o1, a
```

Developer Guide

Overview

WCS Reasoner is written for Python 3.9.2 or newer. The GUI uses PyQt5. Frozen binaries are built with Pyinstaler.

To edit the GUI, for example, to add a button, use Qt Designer and edit .ui files located in the resources folder.

Each button press or similar event has a corresponding function in main.py.

Windows are initialised in GUI.py. To add a new dialog, add a new class there, then add the corresponding variable in main.py.

User input handling, such as parsing infix logical expressions happens in various scripts in the infix folder.

Styling is mainly done by resources/custom.css with a little additional styling for Windows and Linux in resources/non-mac.css.

Freezing Python Code

To freeze the app and distribute the executables:

- Add resource files to r.qrc
- Compile qrc by running `pyrcc5 r.qrc -o resources.py`
- Edit main.spec to include resource files (also, here you can change the target platform)
- Freeze the app via spec by running `pyinstaller main.spec`

Known Issues

The implementation of infix/lexer.py is subpar. Extending this would be challenging. An overhaul is warranted.

First order logic is not supported (that is, the app is limited to datalog programs)

Searching for explanations with observations containing atoms that are not present in the program (i.e. completely unrelated observations) takes a little while even under the recommended explanation length limit.

In case of emergency, contact theShirNick@gmail.com