Walnut 6

Walnut is an automated theorem prover for automatic words.

Looking for older Walnut? The previous version of Walnut (Walnut 5) is available <u>here</u>.

To run Walnut, first run build.sh to build Walnut, then run the walnut.sh file.

Please read the Manual.pdf file in Help Documentation/ to learn what Walnut is and how one would work with it.

Use the help; command to view documentation for all available commands.

To run Walnut tests, run build.sh with the -t flag.

Walnut 6 Additional Documentation

This new version of Walnut has new capabilities and changes added by Anatoly Zavyalov (<u>anatoly.zavyalov@mail.utoronto.ca</u>), with direction from Jeffrey Shallit (<u>shallit@uwaterloo.ca</u>). This version also features major performance improvements by John Nicol.

The new capabilities are as follows:

- Help documentation
- Automata operations

- alphabet command
- Fixing leading and trailing zeroes
- Delimiters for word automata
- Drawing automata and word automata
- Reversing automata
- Bug fixes and performance improvements

Help documentation

Walnut now provides built-in documentation for all commands using the help command. To view all commands for which documentation exists, one writes:

help;

To view documentation for a specific command, one writes:

help <command>;

Automata operations

One can now perform basic automata operations. The operations, along with their commands, are:

- Union of automata, using the union command
- Intersection of automata, using the intersect command
- Kleene star of automata, using the star command
- Concatenation of automata, using the concat command
- Left quotient of automata, using the leftquo command
- Right quotient of automata, using the rightquo command

Union of automata

The syntax for the union command is as follows:

union <new> <old1> [old2] [old3] ... [oldN]

The union command requires at least one input automaton. All automata must have the same input alphabet.

For example, to take the union res of automata named a1 and a2 both saved in Automata Library/, one uses the following command:

union res a1 a2;

The resulting automaton res is saved in Automata Library/, and accepts the union of the inputs accepted by a1 and a2.

Intersection of automata

The syntax for the intersect command is as follows:

intersect <new> <old1> [old2] [old3] ... [oldN]

The intersect command requires at least one input automaton. All automata must have the same input alphabet.

For example, to take the intersection res of automata named a1 and a2 both saved in Automata Library/, one uses the following command:

intersect res a1 a2;

The resulting automaton res is saved in Automata Library/, and accepts the intersection of the inputs accepted by a1 and a2.

Kleene star of automata

The syntax for the star command is as follows:

star <new> <old>

For example, to take the Kleene star res of the automaton aut saved in Automata Library/, one uses the following command:

star res aut;

The resulting automaton res is saved in Automata Library/, and accepts the Kleene star of the inputs accepted by aut.

NOTE: The alphabet of the resulting automaton res will be changed if one of the input alphabets of aut is not a set alphabet (i.e. {0, 1}) or of the form msd_k or lsd_k. Use the alphabet command to force an alphabet on the resulting automaton. For example, if aut is an msd_fib automaton, res will be an msd_2 automaton.

Concatenation of automata

The syntax for the concat command is as follows:

concat <new> <old1> <old2> [old3] ... [oldN]

The concat command requires at least two input automata. All automata must have the same input alphabet.

For example, to take the concatenation res of automata named a1, a2, a3 and a4, all saved in Automata Library/, one uses the following command:

concat res a1 a2 a3 a4;

The resulting automaton res is saved in Automata Library/, and

accepts the concatenation of the inputs accepted by a1, a2, a3, and a4.

NOTE: The alphabet of the resulting automaton res will be changed if one of the input alphabets of the input automata is not a set alphabet (i.e. {0, 1}) or of the form msd_k or lsd_k. Use the alphabet command to force an alphabet on the resulting automaton. For example, if a1, a2, a3, a4 above are msd_fib automata, res will be an msd_2 automaton.

Left quotient of automata

The left quotient of two automata M1 and M2 is defined to be the automaton that accepts the left quotient $L2\L1 = \{w \mid exists x in L2 : xw is in L1\}$, where L1 and L2 are the languages accepted by M1 and M2 respectively.

The syntax for the leftquo command is as follows:

```
leftquo <new> <old1> <old2>
```

For example, to take the left quotient res of automata named a1 and a2 all saved in Automata Library/, one uses the following command:

leftquo res a1 a2;

The resulting automaton res is saved in Automata Library/.

Right quotient of automata

The right quotient of two automata M1 and M2 is defined to be the automaton that accepts the right quotient $L1/L2 = \{w \mid exists x \text{ in } L2 : wx \text{ is in } L1\}$, where L1 and L2 are the languages

accepted by M1 and M2 respectively.

The syntax for the rightquo command is as follows:

rightquo <new> <old1> <old2>

For example, to take the right quotient res of automata named a1 and a2 all saved in Automata Library/, one uses the following command:

rightquo res a1 a2;

The resulting automaton res is saved in Automata Library/.

alphabet command: set the number system of Word Automata and Automata

One may change the alphabet of a Word Automaton using the "alphabet" command as follows:

alphabet <new> <alphabet1> [alphabet2] ... [alphabetN] <old>

Results saved in: Result/, Word Automata Library/.

The alphabet of the resulting automaton is set to the input alphabets, and all invalid transitions are removed from the new automaton.

The number of alphabets in the command must equal to the number of input alphabets of the automaton.

For example, if "AUT" (saved in "Word Automata Library/") has the alphabets "msd_2 msd_2 msd_fib", to set its alphabets to "msd_fib msd_fib msd_4", one writes

alphabet RES msd_fib msd_fib msd_4 AUT

To apply the "alphabet" command to regular automata (that is, not Word Automata), one prepends the "\$" symbol (without the quotation marks) to the old automaton's name. The result will be saved in the "Automata Library/" directory.

For example, if "foo" (saved in "Automata Library/") has alphabet "msd_fib", to set its alphabet to "msd_2" one writes

alphabet bar msd_2 \$foo

The resulting automaton "bar" will be saved in "Automata Library/".

Fixing leading and trailing zeroes

One can now "fix" leading and trailing zeroes for Automata (not Word Automata) using the "fixleadzero" and "fixtrailzero" commands. The syntax is as follows: for an automaton "foo" saved in "Automata Library/", one writes

fixleadzero bar foo;

The resulting automaton bar accepts an input $0^* x'$ if and only if foo accepts an input x, where x' is x with its leading zeroes removed.

Similarly, for trailing zeroes, one writes

fixtrailzero bar foo;

The resulting automaton bar accepts an input $x' 0^*$ if and only if foo accepts an input x, where x' is x with its trailing zeroes removed.

For both cases, the resulting automaton "bar" will be saved in the "Automata Library/" directory.

Delimiters for Word Automata

In previous versions of Walnut, word automata names could not begin with A, E, or I. This restriction has now been lifted using a new delimiter for word automata: putting "." (without quotation marks) before the name of a word automaton now signals that the following string of characters is the name of the word automaton.

If there is a word automaton named AUTOMATON, you can write ".AUTOMATON" (without quotation marks) to refer to it in eval/ def commands. For example, the following is now valid:

```
def test ".AUTOMATON[n] = @1";
```

Drawing automata and word automata

The new draw command creates a .gv file from the .txt definition of a Word Automaton saved in Word Automata Library/, or of an ordinary automaton saved in Automata

Library/.

The syntax for the draw command for Word Automata is as follows:

draw <name>

For ordinary automata, prepend the \$ symbol to the automaton name:

draw \$<name>

For example, to draw a Word Automaton named AUT saved in Word Automata Library/, one writes

draw AUT;

This will save the file AUT.gv in Result/.

To draw an automaton named aut saved in Automata Library/, one writes

draw \$aut;

This will save the file aut.gv in Result/.

Reversing automata

One can now use the "reverse" command to reverse ordinary automata saved in the "Automata Library/" directory.

To reverse Automata, one prepends the "\$" symbol (without the quotation marks) to the old Automaton's name. The result will be saved in the "Automata Library/" directory.

For example, to reverse an Automaton named "foo" saved in

"Automata Library/", one writes

reverse bar \$foo;

The resulting automaton bar will be saved in "Automata Library/".

Bug fixes

- Fixed a bug where the resulting Word Automaton after running the "combine" command was not totalized
- Fixed a bug where reversing an automaton that does not have a number system (i.e. uses {0, 1} as a number system) will throw an error
- Fixed a bug where whitespace and new lines in regular expressions could result in differing automata
- Fixed a bug where using the combine command on one automaton would not totalize the resulting Word Automaton and apply the valid representations

Performance improvements

- Significant memory and time improvements; thanks to John Nicol for his contributions!
- Multiplication has been drastically sped up