Recognizing wellformed arithmetic expressions

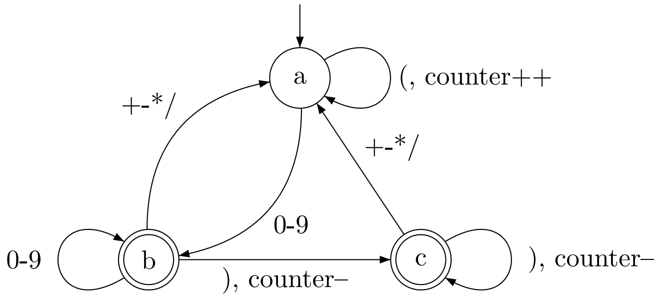
Viktor Engelmann
www.AlgorithMan.de

March 12, 2015

Abstract

This program is (close to) the smalles possible program to check whether a given string is a well-formed arithmetic expression (excluding negative numbers. -5 would have to be given as 0-5 etc.) This program is provided under the terms of the GNU General Public License version 3 (GPL3). See GPL3.txt for details.

This program simulates the following automaton with 1 counter:



The counter (in the program it is called x) stores how many brackets are open, because it is increased iff an opening bracket is read and decreased iff a closing bracket is read. If the counter drops below 0, this means that a prefix of the input has more closing brackets, than opening brackets, which wellformed arithmetic expressions cannot have (in this case the input is rejected - see line 12).
If the whole input has been processed, the program accepts only if the counter has the value 0, because otherwise not every open bracket would have been closed.

Notice how the common transitions of b and c are put in the same code by using

```
b: transition of b that c doesnt have
c: common transitions of b and c
```

also a's transition to b is hacked: s (the pointer to the currently read position of the input) is not incremented to save a line of code. This is later done by b. Only a has to make sure that the transition to b is legal (see line 9), or else you'd get unwanted implied transitions like $a \xrightarrow{+-*/} a$

well-formed arithmetic expressions make the automaton go from state a with counter i to state b or c with counter i.

Proof. Structural induction:

Induction basis

atomic well-formed arithmetic expressions (integers) make the automata go from state a to b, not changing the counter.

Induction Step

- case 1: the expression has the form (exp). The opening bracket makes the automaton go to state a, increasing the counter from i to $i + 1$. By induction exp makes it go to state b or c, with counter $i+1$ and the closing bracket then makes it go to c with counter $(i + 1) - 1 = i$.
- case 2: the expression has the form $exp_1 \; OP \; exp_1$. where OP is +, -, * or /. by induction, exp_1 makes the automaton go from a with counter i to state b or c with counter i, the operator then makes it go to a with counter i, followed by exp_2, which by induction makes it go to state b or c, with counter i.

□