**Homework 1. Fixpoints and grammar filters**

[[131 home](http://web.cs.ucla.edu/classes/spring19/cs131/index.html) > [Homework](http://web.cs.ucla.edu/classes/spring19/cs131/homework.html)]

**Introduction**

You are a reader for Computer Science 181, which asks students to submit grammars that solve various problems. However, many of the submitted grammars are trivially wrong, in several ways. Here is one. Some grammars contain unreachable rules, that is, rules that can never be reached from the start symbol by applying zero or more rules. Unreachable rules do not affect the language or parse trees generated by a grammar, so in some sense they don't make the answers wrong, but they're noise and they make grading harder. You'd like to filter out the noise, and just grade the useful parts of each grammar.

You've heard that OCaml is a good language for writing compilers and whatnot, so you decide to give it a try for this application. While you're at it, you have a background in [fixed point](https://en.wikipedia.org/wiki/Fixed_point_%28mathematics%29) and [periodic point](https://en.wikipedia.org/wiki/Periodic_point) theory, so you decide to give it a try too.

**Definitions**

*fixed point*

(of a function f) A point x such that f x = x. In this description we are using OCaml notation, in which functions always have one argument and parentheses are not needed around arguments.

*computed fixed point*

(of a function f with respect to an initial point x) A fixed point of f computed by calculating x, f x, f (f x), f (f (f x)), etc., stopping when a fixed point is found for f. If no fixed point is ever found by this procedure, the computed fixed point is not defined for f and x.

*periodic point*

(of a function f with period p) A point x such that f (f ... (f x)) = x, where there are p occurrences of f in the call. That is, a periodic point is like a fixed point, except the function returns to the point after piterations instead of 1 iteration. Every point is a periodic point for p=0. A fixed point is a periodic point for p=1.

*computed periodic point*

(of a function f with respect to a period p and an initial point x) A periodic point of f with period p, computed by calculating x, f x, f (f x), f (f (f x)), etc., stopping when a periodic point with period p is found for f. The computed periodic point need not be equal to x. If no periodic point is ever found by this procedure, the computed periodic point is not defined for f, p, and x.

*symbol*

A symbol used in a grammar. It can be either a nonterminal symbol or a terminal symbol; each kind of symbol has a value, whose type is arbitrary. A symbol has the following OCaml type:

type ('nonterminal, 'terminal) symbol =

| N of 'nonterminal

| T of 'terminal

*right hand side*

A list of symbols. It corresponds to the right hand side of a single grammar rule. A right hand side can be empty.

*rule*

A pair, consisting of (1) a nonterminal value (the left hand side of the grammar rule) and (2) a right hand side.

*grammar*

A pair, consisting of a start symbol and a list of rules. The start symbol is a nonterminal value.

**Assignment**

Let's warm up by modeling sets using OCaml lists. The empty list represents the empty set, and if the list t represents the set T, then the list h::t represents the set {h}∪T. Although sets by definition do not contain duplicates, the lists that represent sets can contain duplicates. Another set of warmup exercises will compute fixed points. Finally, you can write a function that filters unreachable rules.

1. Write a function subset a b that returns true iff *a*⊆*b*, i.e., if the set represented by the list a is a subset of the set represented by the list b. Every set is a subset of itself. This function should be generic to lists of any type: that is, the type of subset should be a generalization of 'a list -> 'a list -> bool.
2. Write a function equal\_sets a b that returns true iff the represented sets are equal.
3. Write a function set\_union a b that returns a list representing a∪b.
4. Write a function set\_intersection a b that returns a list representing a∩b.
5. Write a function set\_diff a b that returns a list representing *a*−*b*, that is, the set of all members of *a* that are not also members of *b*.
6. Write a function computed\_fixed\_point eq f x that returns the computed fixed point for f with respect to x, assuming that eq is the equality predicate for f's domain. A common case is that eq will be [(=)](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL%28=%29), that is, the builtin equality predicate of OCaml; but any predicate can be used. If there is no computed fixed point, your implementation can do whatever it wants: for example, it can print a diagnostic, or go into a loop, or send nasty email messages to the user's relatives.
7. OK, now for the real work. Write a function filter\_reachable g that returns a copy of the grammar gwith all unreachable rules removed. This function should preserve the order of rules: that is, all rules that are returned should be in the same order as the rules in g.
8. Supply at least one test case for each of the above functions in the style shown in the sample test cases below. When testing the function F call the test cases my\_F\_test0, my\_F\_test1, etc. For example, for subsetyour first test case should be called my\_subset\_test0. Your test cases should exercise all the above functions, even though the sample test cases do not.

Your code should follow these guidelines:

1. Your code may use the [Pervasives](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html) and [List](https://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html) modules, but it should use no other modules other than your own code.
2. It is OK (and indeed encouraged) for your solutions to be based on one another; for example, it is fine for filter\_reachable to use equal\_sets and computed\_fixed\_point.
3. Your code should prefer pattern matching to conditionals when pattern matching is natural.
4. Your code should be free of [side effects](https://en.wikipedia.org/wiki/Side_effect_%28computer_science%29) such as loops, assignment, input/output, incr, and decr. Use recursion instead of loops.
5. Simplicity is more important than efficiency, but your code should avoid using unnecessary time and space when it is easy to do so. For example, instead of repeating a expression, compute its value once and reuse the computed value.
6. The test cases below should work with your program. You are unlikely to get credit for it otherwise.

Assess your work by writing a brief after-action report that summarizes why you solved the problem the way you did, other approaches that you considered and rejected (and why you rejected them), and any weaknesses in your solution in the context of its intended application. This report should be a [plain text](http://en.wikipedia.org/wiki/Plain_text) file that is no more than 2000 bytes long. See [Resources for oral presentations and written reports](http://web.cs.ucla.edu/classes/spring19/cs131/comm.html) for advice on how to write assessments; admittedly much of the advice there is overkill for the simple kind of report we're looking for here.

**Submit**

Submit three files via CourseWeb. The file hw1.ml should implement the abovementioned functions, along with any auxiliary types and functions; in particular, it should define the symbol type as shown above. The file hw1test.ml should contain your test cases. The file hw1.txt should hold your assessment. Please do not put your name, student ID, or other personally identifying information in your files.

**Sample test cases**

See [hw1sample.ml](http://web.cs.ucla.edu/classes/spring19/cs131/hw/hw1sample.ml) for a copy of these tests.

let subset\_test0 = subset [] [1;2;3]

let subset\_test1 = subset [3;1;3] [1;2;3]

let subset\_test2 = [not](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALnot) (subset [1;3;7] [4;1;3])

let equal\_sets\_test0 = equal\_sets [1;3] [3;1;3]

let equal\_sets\_test1 = not (equal\_sets [1;3;4] [3;1;3])

let set\_union\_test0 = equal\_sets (set\_union [] [1;2;3]) [1;2;3]

let set\_union\_test1 = equal\_sets (set\_union [3;1;3] [1;2;3]) [1;2;3]

let set\_union\_test2 = equal\_sets (set\_union [] []) []

let set\_intersection\_test0 =

equal\_sets (set\_intersection [] [1;2;3]) []

let set\_intersection\_test1 =

equal\_sets (set\_intersection [3;1;3] [1;2;3]) [1;3]

let set\_intersection\_test2 =

equal\_sets (set\_intersection [1;2;3;4] [3;1;2;4]) [4;3;2;1]

let set\_diff\_test0 = equal\_sets (set\_diff [1;3] [1;4;3;1]) []

let set\_diff\_test1 = equal\_sets (set\_diff [4;3;1;1;3] [1;3]) [4]

let set\_diff\_test2 = equal\_sets (set\_diff [4;3;1] []) [1;3;4]

let set\_diff\_test3 = equal\_sets (set\_diff [] [4;3;1]) []

let computed\_fixed\_point\_test0 =

computed\_fixed\_point (=) (fun x -> x [/](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL%28/%29) 2) 1000000000 = 0

let computed\_fixed\_point\_test1 =

computed\_fixed\_point (=) (fun x -> x [\*.](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL%28%20*.%20%29) 2.) 1. = [infinity](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALinfinity)

let computed\_fixed\_point\_test2 =

computed\_fixed\_point (=) [sqrt](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALsqrt) 10. = 1.

let computed\_fixed\_point\_test3 =

((computed\_fixed\_point (fun x y -> [abs\_float](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALabs_float) (x [-.](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL%28-.%29) y) < 1.)

(fun x -> x [/.](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VAL%28/.%29) 2.)

10.)

= 1.25)

(\* An example grammar for a small subset of Awk. \*)

type awksub\_nonterminals =

| Expr | Lvalue | Incrop | Binop | Num

let awksub\_rules =

[Expr, [T"("; N Expr; T")"];

Expr, [N Num];

Expr, [N Expr; N Binop; N Expr];

Expr, [N Lvalue];

Expr, [N Incrop; N Lvalue];

Expr, [N Lvalue; N Incrop];

Lvalue, [T"$"; N Expr];

Incrop, [T"++"];

Incrop, [T"--"];

Binop, [T"+"];

Binop, [T"-"];

Num, [T"0"];

Num, [T"1"];

Num, [T"2"];

Num, [T"3"];

Num, [T"4"];

Num, [T"5"];

Num, [T"6"];

Num, [T"7"];

Num, [T"8"];

Num, [T"9"]]

let awksub\_grammar = Expr, awksub\_rules

let awksub\_test0 =

filter\_reachable awksub\_grammar = awksub\_grammar

let awksub\_test1 =

filter\_reachable (Expr, [List.tl](https://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html#VALtl) awksub\_rules) = (Expr, List.tl awksub\_rules)

let awksub\_test2 =

filter\_reachable (Lvalue, awksub\_rules) = (Lvalue, awksub\_rules)

let awksub\_test3 =

filter\_reachable (Expr, List.tl (List.tl awksub\_rules)) =

(Expr,

[Expr, [N Expr; N Binop; N Expr];

Expr, [N Lvalue];

Expr, [N Incrop; N Lvalue];

Expr, [N Lvalue; N Incrop];

Lvalue, [T "$"; N Expr];

Incrop, [T "++"];

Incrop, [T "--"];

Binop, [T "+"];

Binop, [T "-"]])

let awksub\_test4 =

filter\_reachable (Expr, List.tl (List.tl (List.tl awksub\_rules))) =

(Expr,

[Expr, [N Lvalue];

Expr, [N Incrop; N Lvalue];

Expr, [N Lvalue; N Incrop];

Lvalue, [T "$"; N Expr];

Incrop, [T "++"];

Incrop, [T "--"]])

type giant\_nonterminals =

| Conversation | Sentence | Grunt | Snore | Shout | Quiet

let giant\_grammar =

Conversation,

[Snore, [T"ZZZ"];

Quiet, [];

Grunt, [T"khrgh"];

Shout, [T"aooogah!"];

Sentence, [N Quiet];

Sentence, [N Grunt];

Sentence, [N Shout];

Conversation, [N Snore];

Conversation, [N Sentence; T","; N Conversation]]

let giant\_test0 =

filter\_reachable giant\_grammar = giant\_grammar

let giant\_test1 =

filter\_reachable (Sentence, List.tl ([snd](https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALsnd) giant\_grammar)) =

(Sentence,

[Quiet, []; Grunt, [T "khrgh"]; Shout, [T "aooogah!"];

Sentence, [N Quiet]; Sentence, [N Grunt]; Sentence, [N Shout]])

let giant\_test2 =

filter\_reachable (Quiet, snd giant\_grammar) = (Quiet, [Quiet, []])

**Sample use of test cases**

When testing on SEASnet, use one of the machines lnxsrv06.seas.ucla.edu,lnxsrv07.seas.ucla.edu, lnxsrv09.seas.ucla.edu, and lnxsrv10.seas.ucla.edu. Make sure /usr/local/cs/bin is at the start of your path, so that you get the proper version of OCaml. To do this, append the following lines to your $HOME/.profile file if you use [bash](https://www.gnu.org/software/bash/) or [ksh](http://www.kornshell.com/):

export PATH=/usr/local/cs/bin:$PATH

or the following line to your $HOME/.login file if you use [tcsh](https://en.wikipedia.org/wiki/Tcsh) or [csh](https://en.wikipedia.org/wiki/C_shell):

set path=(/usr/local/cs/bin $path)

The command ocaml should output the version number 4.07.1.

If you put the [sample test cases](http://web.cs.ucla.edu/classes/spring19/cs131/hw/hw1sample.ml) into a file hw1sample.ml, you should be able to use it as follows to test your hw1.ml solution on the SEASnet implementation of OCaml. Similarly, the command #use "hw1test.ml";;should run your own test cases on your solution.

$ ocaml

Objective Caml version 4.07.1

# #use "hw1.ml";;

type ('a, 'b) symbol = N of 'a | T of 'b

...

# #use "[hw1sample.ml](http://web.cs.ucla.edu/classes/spring19/cs131/hw/hw1sample.ml)";;

val subset\_test0 : bool = true

val subset\_test1 : bool = true

val subset\_test2 : bool = true

val equal\_sets\_test0 : bool = true

val equal\_sets\_test1 : bool = true

val set\_union\_test0 : bool = true

val set\_union\_test1 : bool = true

val set\_union\_test2 : bool = true

val set\_intersection\_test0 : bool = true

val set\_intersection\_test1 : bool = true

val set\_intersection\_test2 : bool = true

val computed\_fixed\_point\_test0 : bool = true

val computed\_fixed\_point\_test1 : bool = true

val computed\_fixed\_point\_test2 : bool = true

val computed\_fixed\_point\_test3 : bool = true

type awksub\_nonterminals = Expr | Lvalue | Incrop | Binop | Num

val awksub\_rules :

(awksub\_nonterminals \* (awksub\_nonterminals, string) symbol list) list =

[(Expr, [T "("; N Expr; T ")"]); (Expr, [N Num]);

(Expr, [N Expr; N Binop; N Expr]); (Expr, [N Lvalue]);

(Expr, [N Incrop; N Lvalue]); (Expr, [N Lvalue; N Incrop]);

(Lvalue, [T "$"; N Expr]); (Incrop, [T "++"]); (Incrop, [T "--"]);

(Binop, [T "+"]); (Binop, [T "-"]); (Num, [T "0"]); (Num, [T "1"]);

(Num, [T "2"]); (Num, [T "3"]); (Num, [T "4"]); (Num, [T "5"]);

(Num, [T "6"]); (Num, [T "7"]); (Num, [T "8"]); (Num, [T "9"])]

val awksub\_grammar :

awksub\_nonterminals \*

(awksub\_nonterminals \* (awksub\_nonterminals, string) symbol list) list =

(Expr,

[(Expr, [T "("; N Expr; T ")"]); (Expr, [N Num]);

(Expr, [N Expr; N Binop; N Expr]); (Expr, [N Lvalue]);

(Expr, [N Incrop; N Lvalue]); (Expr, [N Lvalue; N Incrop]);

(Lvalue, [T "$"; N Expr]); (Incrop, [T "++"]); (Incrop, [T "--"]);

(Binop, [T "+"]); (Binop, [T "-"]); (Num, [T "0"]); (Num, [T "1"]);

(Num, [T "2"]); (Num, [T "3"]); (Num, [T "4"]); (Num, [T "5"]);

(Num, [T "6"]); (Num, [T "7"]); (Num, [T "8"]); (Num, [T "9"])])

val awksub\_test0 : bool = true

val awksub\_test1 : bool = true

val awksub\_test2 : bool = true

val awksub\_test3 : bool = true

val awksub\_test4 : bool = true

type giant\_nonterminals =

Conversation

| Sentence

| Grunt

| Snore

| Shout

| Quiet

val giant\_grammar :

giant\_nonterminals \*

(giant\_nonterminals \* (giant\_nonterminals, string) symbol list) list =

(Conversation,

[(Snore, [T "ZZZ"]); (Quiet, []); (Grunt, [T "khrgh"]);

(Shout, [T "aooogah!"]); (Sentence, [N Quiet]); (Sentence, [N Grunt]);

(Sentence, [N Shout]); (Conversation, [N Snore]);

(Conversation, [N Sentence; T ","; N Conversation])])

val giant\_test0 : bool = true

val giant\_test1 : bool = true

val giant\_test2 : bool = true

#

*© 2006–2011, 2013–2019*[*Paul Eggert*](http://web.cs.ucla.edu/classes/spring19/cs131/mail-eggert.html)*. See*[*copying rules*](http://web.cs.ucla.edu/classes/spring19/cs131/copyright.html)*.  
$Id: hw1.html,v 1.76 2019/04/02 22:13:55 eggert Exp $*