

Semantic and Declarative Technologies. Sample mid-course test 2.

Time: 80 minutes. Total score: 200 points

For tasks 1 and 2 this is a “closed book” test. You are not allowed to use a computer during this phase. However, you may ask the instructors for clarification or hints, if needed. After submitting solutions for tasks 1 and 2 you can use your laptop for writing the programs for tasks 3 and 4. However you should use internet only for reading the Prolog manuals, slides, and sample programs. Please do not do searches on the internet.

For tasks 1 and 2 you should preferably write the solutions on this task sheet, in the space provided. Otherwise please refer to the (sub)questions using their numeric (and alphabetic) label, e.g., 2.b.

In tasks 3 and 4 you may use all built-in predicates as well as those found in the slides.

When writing down solutions, please refer to the (sub)questions using their numeric (and alphabetic) label, e.g., 2.b.

1. Determine whether the execution of the following Prolog queries results in an *error* (no need to name a specific error type), *failure*, or *success*. In case of success, specify the variable substitutions of named (non-void, i.e. non `_`) variables. All queries are fed to Prolog independently, i.e. typed on their own after the `| ?-` prompt. (6*5 = 30 p)

- (a) `\+ X = 6, X = 2.`
- (b) `[a|[b,c,d]] = [X,Y|L].`
- (c) `A*B = 2*5*(7+2).`
- (d) `2+3 is U+V.`
- (e) `N is 2*4, M = N+1.`
- (f) `append([2|R], S, [3,4,5]).`

2. Assume that the following program is loaded into the Prolog system.

```
p([_|Xs], A, Y) :-
    A1 is A+1,
    p(Xs, A1, Y).
p([X|Xs], X, X).
```

Determine the values that `X` will take as a result of the following (independent) queries. Write down *all* solutions separated by semicolons, in the order Prolog enumerates them. If there are no solutions, write `no`. (4*10+10 = 50 p)

- (a) `p([], 5, Y).`
- (b) `p([3,2,1,3], 1, Y).`
- (c) `p([0,3,2,1], 0, Y).`
- (d) `p([1,3,9,5,6,4,8], 2, Y).`

Consider the following predicate, which builds on the predicate `p/3` defined above:

```
% p(L, Z): Z is an element of L such that...
p(L, Z) :- p(L, 0, Z).
```

- (e) Provide a declarative spec for the predicate `p/2`, i.e. expand the head comment above to a full sentence. Furthermore, describe in what order are the solutions enumerated.

3. Consider a linear expression of the form $a_1x_1 + \dots + a_nx_n$, where a_i are numbers and x_i are variables. Let us represent such an expression with a Prolog list of the form $[A_1 * X_1, \dots, A_n * X_n]$, where A_i are (Prolog) numbers and X_i are atoms (variable names). For example, the Prolog list $[1 * x, 2 * y, 3 * z]$ represents the algebraic formula $x + 2y + 3z$, while the $[]$ empty list represents the formula 0. We assume that no two elements of the list contain the same X_i and no A_i is equal to 0.

Write a Prolog predicate `lin_sum(+E0, +A, +X, ?E)`, where `E0` is a list representing a linear expression, `A` is a number, `X` is an atom, and `E` is a list representing the sum of `E0` and `A*X`, satisfying the assumptions discussed above. The order of the elements of the list `E` is arbitrary.

```
% lin_sum(+E0, +A, +X, -E): E is the list format representation of the sum
% of E0 (in list format) with A*X.
```

(80 p)

Examples:

```
| ?- lin_sum([], 5, x, K).           ==> K = [5*x] ? ; no
| ?- lin_sum([1*u,3*y], 3, u, K).   ==> K = [4*u,3*y] ? ; no
| ?- lin_sum([-1*u,3*y], 1, u, K).  ==> K = [3*y] ? ; no
| ?- lin_sum([1*u,3*y,7*x,8*q], 3, x, K). ==> K = [1*u,3*y,10*x,8*q] ? ; no
| ?- lin_sum([1*u,3*y,8*q], 3, x, K). ==> K = [1*u,3*y,8*q,3*x] ? ; no
```

Hints: To handle the first three test cases, you should write non-recursive clauses. The fourth and fifth test cases should be covered by a single recursive clause.

4. Consider a Prolog term that takes one of the following forms: (a) `Atom*Number`, (b) `Number*Atom`, (c) `Atom`, or (d) a compound built from cases (a)–(c) through repeated use of the operator `+`.

Write a Prolog predicate that takes a term in the above operator-based form and returns its equivalent in list-based form, as described in task 3.

```
% list_form_of(+OpExpr, -E): The list based form of the operator-based
% expression OpExpr is E.
```

(40 p)

Examples:

```
| ?- list_form_of(y*3, K).           ==> K = [3*y] ? ; no
| ?- list_form_of(6*x, K).           ==> K = [6*x] ? ; no
| ?- list_form_of(x, K).              ==> K = [1*x] ? ; no
| ?- list_form_of(-1*x+x, K).         ==> K = [] ? ; no
| ?- list_form_of(x*5+x, K).          ==> K = [6*x] ? ; no
| ?- list_form_of(x*5+ -1*x, K).      ==> K = [4*x] ? ; no
| ?- list_form_of(5*x+(3*x+y*2), K). ==> K = [8*x,2*y] ? ; no
| ?- list_form_of((x+2*y)+(x+3*y), K). ==> K = [4*x,3*y] ? ; no
```

Hint: use the following helper predicate

```
% list_form_of(+OpExpr, +E0, -E): The sum of the operator-based OpExpr and
% list-based E0 is the list-based expression E.
```

The above cases (a)–(c) can be handled using a non-recursive clause, while case (d) can be covered by a doubly recursive clause.