CS 599 A1: Assignment 5

Due: Thursday, April 16, 2026

Total: 100 pts

Instructor: Ankush Das

- This assignment is due midnight on the above date and must be submitted electronically on Gradescope.
- This is a programming assignment in Prolog. You must submit a `.pl` file for each problem. The name of the `.pl` file is specified next to the points of each problem. You must use that file name.
- Each problem requires you to define and implement one predicate, but you are welcome to define as many helper predicates as you like.

1 Fun with Prolog

Problem 1 (Mandatory; No LLMs; 10 pts; `dups.pl`) *Implement the following predicate:*

```
remove_duplicates(List, Result)
```

The predicate should satisfy the following requirements:

- `Result` *contains the elements of* `List` *with duplicates removed.*
- `Result` *should preserve the order of first occurrence in* `List`*.*

Example:

```
?- remove_duplicates([a,b,a,c,b,c,a], X).
   X = [a,b,c].
```

Do not use any built-in predicates. Your implementation will be tested as shown above where `Result` *is a variable that should be computed by Prolog.*

Problem 2 (Mandatory; No LLMs; 20 pts; `sublists.pl`) *Implement the following predicate:*

```
sublists(List, Sublists)
```

The predicate should satisfy the following requirements:

- `List` *is a list of distinct elements.*
- *A list m is defined as a sublist of a list ℓ if the elements of m form an* ***in-order*** *subsequence of the elements of ℓ. For e.g.,* `[a, b]` *is a sublist of* `[a, b, c]`*, but* `[b, a]` *is not a sublist of* `[a, b, c]`*.*
- `Sublists` *should be a list containing all possible sublists of* `List`*,* ***in any order****. In other words, every list in* `Sublists` *must be a sublist of* `List`*.*

Example:

```
?- sublists([a,b,c], X).
   X = [[], [c], [b], [b, c], [a], [a, c], [a, b], [a, b, c]].
```

Do not use any built-in predicates. Your implementation will be tested as shown above where `Sublists` *is a variable that should be computed by Prolog.*

2 Inversion Calculus with Proof Terms

Problem 3 (Mandatory; No LLMs; 30 pts; `terms.pl`) *Recall the inversion calculus Prolog program discussed in class. In this problem, you need to extend it with proof terms. Implement the following predicate:*

```
proof_term(Gamma, Prop, Term)
```

The predicate should satisfy the following requirements:

- `Gamma` *is still a list but now has a variable name associated with each proposition in the list. Concretely, each entry in* `Gamma` *is of the form* `var(x, A)` *where* `x` *is a variable name and* `A` *is a proposition.* `Prop` *is a proposition, same as before, and* `Term` *is a proof term.*
- *The query* `proof_term(`Γ`, `A`, T)` *should compute* `T` *if* $\Gamma \longrightarrow A$ *holds, and false otherwise.*
- *All the standard logical connectives must be supported. And they must be implemented as* `and(A, B)`, `or(A, B)`, `impl(A, B)`, `top`, `bot`, *and* `atom(p)`[1].
- *You must use the following terms:*
  - `pair(E1, E2)` *represents the pair* $\langle E_1, E_2 \rangle$.
  - `letpair(A, B, X, E)` *represents the let-expression for pair, namely* $\mathsf{let}\ \langle A, B \rangle = X\ \mathsf{in}\ E$.
  - `inl(E)` *represents* $\mathsf{inl}\ E$, *and* `inr(E)` *represents* $\mathsf{inr}\ E$.
  - `match(X, A, E1, B, E2)` *represents the match-expression* $\mathsf{case}\ X\ \mathsf{of}\ \mathsf{inl}\ A \Rightarrow E_1 \mid \mathsf{inr}\ B \Rightarrow E_2$.
  - `fun(X, E)` *represents the function* $\mathsf{fun}\ X \Rightarrow E$.
  - `letapp(V, X, E1, E2)` *represents the let-expression for application, namely* $\mathsf{let}\ V = X\ E_1\ \mathsf{in}\ E_2$.
  - `unit` *represents* $\langle\rangle$, *the proof term corresponding to* `top`.
  - `abort` *represents* abort*-ing a program.*

Example:

```
?- proof_term([var(x, atom(a))], atom(a), T).
   T = x
?- proof_term([var(x, atom(a))], atom(b), T).
   false.
?- proof_term([var(x, or(atom(a), atom(b))), var(y, or(atom(a), atom(c)))
   ], or(atom(a), and(atom(b), atom(c))), T).
   T = match(x, _A, match(y, _B, inl(_B), _, inl(_A)), _C, match(y, _D,
       inl(_D), _E, inr(pair(_C, _E))))
```

Do not use any built-in predicates. Your implementation will be tested as shown above where both `Gamma` *and* `Prop` *are given, and the predicate must compute* `Term`.

3 Contraction-Free Sequent Calculus with Inversion

Recall the calculus from Assignment 4 that combined the inversion calculus with contraction-free sequent calculus. In this assignment, we will implement this calculus in Prolog.

Problem 4 (Mandatory; No LLMs; 40 pts; `cfic.pl`) *Implement the following predicate:*

```
prove(Gamma, Prop)
```

The predicate must implement rules from the above calculus and satisfy the following requirements:

- `Gamma` *is a list of propositions and* `Prop` *is a proposition.*

[1] In class, we did not cover `top` and `bot`, so you need to add inference rules for these connectives

- *The query* `prove(Γ, A)` *should return true if* $\Gamma \longrightarrow A$*, and false otherwise.*
- *All the standard logical connectives must be supported. And they must be implemented as* `and(A, B)`*,* `or(A, B)`*,* `impl(A, B)`*,* `top`*,* `bot`*, and* `atom(p)`*.*

Example:

```
?- prove([atom(a)], atom(a)).
   true
?- prove([atom(a)], atom(b)).
   false.
?- prove([or(atom(a), atom(b)), or(atom(a), atom(c))],
            or(atom(a), and(atom(b), atom(c)))).
   true
```

Do not use any built-in predicates. Your implementation will be tested as shown above where both `Gamma` *and* `Prop` *are given, and the predicate must return true or false.*

Hint: *The file* `inv_calc.pl` *in the lecture notes can serve as a good starting point for this problem.*

Problem 5 (Optional; LLMs ok; 50 pts; `cfic_terms.pl`**)** *Extend the predicate above with proof terms. Implement the following predicate:*

```
proof_term(Gamma, Prop, Term)
```

The predicate should satisfy the following requirements:

- `Gamma` *is still a list but now has a variable name associated with each proposition in the list. Concretely, each entry in* `Gamma` *is of the form* `var(x, A)` *where* `x` *is a variable name and* `A` *is a proposition.*
- *The query* `proof_term(Γ, A, T)` *should compute the term* `T` *if* $\Gamma \longrightarrow A$*, and false otherwise.*
- *Same as before, you must use the following terms:*
  - `pair(E1, E2)` *represents the pair* $\langle E_1, E_2 \rangle$*.*
  - `letpair(A, B, X, E)` *represents the let-expression for pair, namely* $\mathsf{let}\ \langle A, B \rangle = X\ \mathsf{in}\ E$*.*
  - `inl(E)` *represents* $\mathsf{inl}\ E$*, and* `inr(E)` *represents* $\mathsf{inr}\ E$*.*
  - `match(X, A, E1, B, E2)` *represents the match-expression* $\mathsf{case}\ X\ \mathsf{of}\ \mathsf{inl}\ A \Rightarrow E_1 \mid \mathsf{inr}\ B \Rightarrow E_2$*.*
  - `fun(X, E)` *represents the function* $\mathsf{fun}\ X \Rightarrow E$*.*
  - `letapp(V, X, E1, E2)` *represents the let-expression for application, namely* $\mathsf{let}\ V = X\ E_1\ \mathsf{in}\ E_2$*.*
  - `unit` *represents* $\langle\rangle$*, the proof term corresponding to* `top`*.*
  - `abort` *represents* abort*-ing a program.*

Example:

```
?- proof_term([var(x, atom(a))], atom(a), T).
   T = x
?- proof_term([var(x, atom(a))], atom(b), T).
   false.
?- proof_term([var(x, or(atom(a), atom(b))), var(y, or(atom(a), atom(c)))
   ], or(atom(a), and(atom(b), atom(c))), T).
   T = match(x, _A, match(y, _B, inl(_B), _, inl(_A)), _C, match(y, _D,
       inl(_D), _E, inr(pair(_C, _E))))
```

Do not use any built-in predicates. Your implementation will be tested as shown above where both `Gamma` *and* `Prop` *are given, and the predicate must compute* `Term`*.*

Hint: *Recall the pen-and-paper rules that you used in Assignment 4 for Contraction-Free Sequent Calculus with Inversion. Before solving this problem in Prolog, augment these pen-and-paper rules with proof terms using the (abstract) terms specified above.*