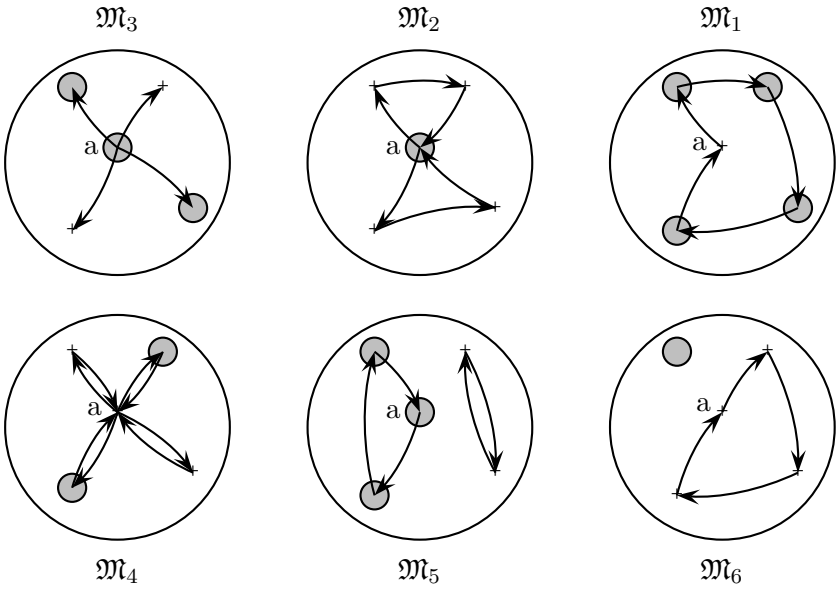
Assignment 3

COMP 2411, Session 1, 2004

Q1 (2 marks): Determine whether each of the following formulas is valid. In case the formula is not valid give a model of the negation of the formula. In case the formula is valid justify your answer.

- $\exists x \forall y \exists z((p(y,z) \rightarrow p(x,z)) \rightarrow p(x,x) \rightarrow p(y,x))$
- $\forall x \forall y(p(x,y) \vee q(y,x)) \rightarrow \forall x(\forall y\, p(x,y) \vee \exists y\, q(y,x))$.

Q2 (2 marks): Consider a vocabulary consisting of a constant a, a unary predicate symbol P, and a binary predicate symbol R, and the three structures $\mathfrak{M}_i$, $1 \leq i \leq 6$, represented below (up to isomorphism). All these structures have a common domain of cardinality 5. The interpretation of a is the same in all structures, and is equal to an individual that is represented by the center point. We render the fact that an individual x has property P with a grey circle on the cross that represents x, and we render the fact that $R(x,y)$ holds for individuals x and y with an arrow from the cross that represents x to the cross that represents y.



Determine for each of the following formulas in which structures it is true.

- $\forall x \forall y(R(x,y) \rightarrow (\neg P(x) \wedge \neg P(y)))$
- $\forall x_1 \ldots \forall x_{11}(R(a,x_1) \wedge R(x_1,x_2) \wedge \ldots \wedge R(x_{10},x_{11}) \rightarrow R(x_{11},a))$
- $\forall x(\exists y(P(y) \wedge R(x,y)) \rightarrow \forall z(R(x,z) \rightarrow \exists u(R(u,z) \wedge P(u))))$

Let your student number end in $a_2a_3a_4a_5a_6$. Find a formula φ without equality such that $\mathfrak{M}_1 \models \varphi$ and for all $i \in \{2, \ldots, 6\}$, $\mathfrak{M}_i \models \varphi$ iff a_i is even. For instance, if your student number ends in 53441 then your formula should be true in $\mathfrak{M}_1$, $\mathfrak{M}_4$ and $\mathfrak{M}_5$ and false in $\mathfrak{M}_2$, $\mathfrak{M}_3$ and $\mathfrak{M}_6$. You should not only write down your formula, but also express in plain English what it represents graphically, so that it is clear from your description that it is true in the structures it is meant to be true in.

Q3 (2 marks): Using Semantic tableaux, show that $\forall x \forall y(p(x, y) \rightarrow p(x, x))$ is a logical consequence of $\{\forall x \forall y(p(x, y) \rightarrow p(y, x)), \forall x \forall y \forall z(p(x, y) \wedge p(y, z) \rightarrow p(x, z))\}$.

Note 1: Due to space constraints, it is probably a good idea to introduce a few abbreviations, like $A = \forall x \forall y(p(x, y) \rightarrow p(y, x))$, $B = \forall x \forall y \forall z(p(x, y) \wedge p(y, z) \rightarrow p(x, z))$, $C = \forall x \forall y(p(x, y) \rightarrow p(x, x))$, etc., and use these abbreviations in the tableau's labels.

Note 2: Your tableau can have as many branches as you like, but you need no more than 4 branches. If you try and use the program `tableau_test.pl`, you will find out that the program is not very helpful because the tree it produces is far from optimal.

4 (2 marks): Write code for `well_formed_polynomial/1`, which tests whether a polynomial is well formed according to the following description. You can assume the argument will be ground (not contain any variable) when this predicate is called.

In this context a well-formed polynomial is one with positive integer coefficients only. It has representation:

```
Monic + Monic + ... + Monic
```

There must be at least one monic in any polynomial expression. A monic polynomial is represented by:

```
Num * Exponent * ... * Exponent
```

where `Num` is in a Peano representation (e.g., a term of the form `s(s(...s(0)...))`), and `Exponent` is an exponent of the form `atom^Num`. The number of exponents can be zero, but there must be exactly one `Num`.

For instance, the polynomial $x^2 + 2xy + y$ is represented by:

```
s(0) * x^s(s(0))  + s(s(0)) * x^s(0) * y^s(0) + s(0) * y^s(0).
```

As a special case, an exception to the above rules, `0` is also a well formed polynomial.

Exponents must be stored in order from greatest degree to least degree, with those of equal degree sorted using the internal `sort/2` predicate or Prolog atom comparison operations.

For instance:

```
?- well_formed_polynomial(s(0) * x^s(0) * y^s(0))
Yes.

?- well_formed_polynomial(s(0) * y^s(0) * x^s(0))
No.
```

Monics must be sorted in order from greatest degree to least degree, with monics of equal degree sorted in the order given by `sort/2` predicate on the exponents, or internal Prolog comparison operations.

For instance:

```
?- well_formed_polynomial(s(0) * x^s(s(0)) +
              s(s(0)) * x^s(0) * y^s(0) + s(0) * y^s(0)).
Yes.

?- well_formed_polynomial(s(0) * x^s(0) +
               s(s(0)) * x^s(0) * y^s(0) + s(0) * y^s(0)).
No.
```

Note that under this definition, all coefficients are positive. Zero coefficients and exponents are not permitted, so `s(0) * x^0` and `0 * x^s(s(0))` are not well formed polynomials. As noted above however, `0` is a special case.

Other examples:

```
?- well_formed_polynomial(x^s(s(0))).
No.

?- well_formed_polynomial(-0 * y^s(0))
No.

?- well_formed_polynomial(4^s(0))
No.
```

Hint: you should use the code for `write_polynomial` as a guide for how to proceed.

5 (2 marks):

a) Write code for `add_polynomial/3` and `multiply_polynomial/3`, working on well formed polynomials so that the result of `add_polynomial(p_1, p_2, p_3)` is that `p_1 + p_2 = p_3`, and the result of `multiply_polynomial(p_1, p_2, p_3)`/3' is that `p_1 * p_2 = p_3`. You can assume that the first two arguments are ground, and the result will be retuned in the third argument. If any argument is not well formed the query should fail (i.e. return `No`). The returned argument must be a well formed polynomial.

Hint: If you're finding the full exercise hard, try first writing code which adds two monics, then a monic to a polynomial, and then two polynomials. Do the same for multiplication.

b) Given some k and l, define a polynomial matrix as a $k \times l$ matrix with (well formed positive integer) polynomial entries.

Write code for `add_polynomial_matrix/3` and `multiply_polynomial_matrix`, which performs matrix addition and multiplication analogously to the previous exercise. A matrix is stored as a list of rows, where rows are all lists of the same length. You can assume that the first two arguments are ground when the predicate is called.

If any of the arguments is not a correctly formatted polynomial matrix (as described) the operation should fail (i.e. answer No). If the number of rows and columns in the matrix do not permit matrix multiplication or addition the operation should fail (i.e. answer No). The returned argument must consist of well formed polynomials.

e.g:

```
?- add_polynomial_matrix([[s(s(0)), s(0) * x^s(0)], [s(0) * x^s(0), s(s(0))]],
                         [[s(0) * x^s(0), s(s(0))], [s(s(0)), s(0) * x^s(0)]],
                         X).

X = [[ s(0) * x^s(0) + s(s(0)), s(0) * x^s(0) + s(s(0)) ],
     [ s(0) * x^s(0) + s(s(0)), s(0) * x^s(0) + s(s(0)) ]]

?- add_polynomial_matrix([[s(s(0)), s(0) * x^s(0)], [s(0) * x^s(0)]],
                         [[s(0) * x^s(0), s(s(0))], [s(s(0)), s(0) * x^s(0)]],
                         X).
No.
```

Hint: You may find E. Kreyszig, "Advanced Engineering Mathematics" helpful if you need to revise linear algebra; any first-year linear algebra text should help you with matrix multiplication and addition. Mostly this task is straightforward list manipulation, and in some ways it's easier than the previous exercises.

Bonus Marks:

An invertible $n \times n$ polynomial matrix is a matrix M such that there exists some N so that $M \cdot N = I$, where I has the standard definition. Write code for `invertible_matrix/1` which tests if an $k \times l$ matrix is invertible or not. You will need to test for the matrix being an $n \times n$ matrix.

For full bonus marks you will need to make your solution work on large matrices and high degree polynomials with both positive and negative integer coefficients, in reasonable time with reasonable memory usage—this means that you should be able to use the standard swi-prolog heap size settings, and not make the marker bored while he/she waits for your code to execute. You may wish to look at Knuth "Art of Computer Programming" volume 2, and Cormen, Leiserson, Rivest, "Introduction to Algorithms" for some hints and tricks for efficient polynomial and matrix multiplication, and any text discussing the Gauss-Jordan decomposition.

Up to 2 bonus marks will be awarded for attempting this, but for a maximum of 10/10 for the assignment. That is, if a student gets 9/10 for the rest of the assignment, and is allocated 2 bonus marks, their final mark will be 10/10. If a student gets 10/10 and doesn't attempt the bonus marks, they will receive 10/10 for the assignment as a whole. Note that this task has the potential to be very time consuming, and is only suggested for talented students bored with the previous activities and interested in arithmetic or algebra. Please note in a comment at the beginning of your submission, "Attempting bonus marks".

Submission:

Submit your assignment as `ass3p2` using the give command. Your submission should all be in the one file. There will be sample code on the web in the file `ass3_sample.pl`.