Acta Cybernetica **18** (2008) 595–613.

Computer-Based Intelligent Educational Program for Teaching Chemistry

Róbert Pántya* and László Zsakó†

Abstract

Improving problem-solving skills is a basic requirement in present education. Logic programming languages, often used in the area of artificial intelligence and expert systems, are very much suitable for developing problem-solving skills.

This paper presents a computer-based tool for solving different chemical problems (calculation of quantum numbers, description of electron configurations of atoms, determination of oxidation numbers and electronegativity), which uses the logic programming language Prolog. The authors argue that this intelligent educational material does not only improve chemistry education but it also inspires students to create rule-based systems when they face various problems of everyday life.

Keywords: expert systems, rule-based systems, intelligent educational programs, artificial intelligence

1 Introduction

Computer-assisted problem solving is an important field of informatics education. There are countless occasions in everyday life in which people are confronted with complicated tasks. Students have to decide if such problems can be solved using the tools of informatics, or not. If solving a problem requires the use of a computer, they either have to select an existing tool adequate for the task, or have to build a new one [15].

Computers and programs are frequently used in education. However, such educational programs cannot work without intelligent educational modules. Beyond its basics functions, educational software has to provide advice to students. In the centre of an intelligent tutoring system there is an expert system which encapsulates a brief summary of the knowledge of a given topic [3].

Expert systems are used to solve problems that require natural intelligence. Knowledge is stored in expert systems in a way that makes it possible for the system

*Károly Róbert College, Department of Business Mathematics and Informatics, E-mail: rpantya@karolyrobert.hu

†ELU Department of Teacher Training in Computer Science, E-mail: zsako@ludens.elte.hu

to provide a solution for a problem, making decisions when necessary, and justifying its choices using an appropriate argumentation. Besides transforming knowledge into the most appropriate form, the building of an expert systems also involves further elements, such as strategies for knowledge representation and consistency checking, and the use of heuristic search. Such tasks can be implemented efficiently by using artificial intelligence techniques [12].

There are a lot of existing expert systems (Dendral, MetabolExpert, Mycin, Internist, Guidon, MProlog Shell, Hearsey etc. [12]). Expert systems have attracted considerable attention in the field of chemistry over the past three to four decades. Dendral (Dendritic Algorithm), the first successful expert system in chemistry, was constructed in the 1970's. Its primary aim was to identify unknown organic molecules by analyzing their mass spectra and using the knowledge of chemistry. Originally it was developed for being deployed in a satellite in NASA's Mars program [6].

Further famous chemical expert systems include the Metabol Expert (used in chemical, medical and biological predictions), SELEX (which can be used for the evaluation and quantitative rating of data published on selenium content of foods), and others, such as expert systems for structure elucidation, relying on spectroscopy knowledge bases, data property expert systems, etc. [8].

Another interesting expert system, which can be used in teaching chemistry, is PIRExS (Predicting Inorganic Reactivity Expert System) [2], which predicts the products of inorganic reactions.

Rule-based expert systems seem to be most appropriate for solving chemical problems. In relation to this, it may be of interest to study the origins of the periodic table. In the 19^{th} century, D. I. Mendeleyev arranged the chemical elements in a table according to their atomic mass. At that time the atomic structure was unknown, and thus the most important feature of the atoms was their atomic mass. Mendeleyev realised that every eighth element is similar, and therefore he placed such similar elements in the same column.

There were blank places in his table but he trusted that appropriate new elements, which were not yet discovered, would be found later. He could not explain this similarity of chemical properties, as he did not know the atomic structure. But his conclusions were correct. He discovered a wonderful natural law. The way Mendeleyev reasoned is characteristic of rule-based expert systems. 60 years later, after the discovery of protons and electrons, his system could be explained: the order of the elements is determined by the number of protons, while the chemical properties depend on the characteristics of the electron cloud [5].

In a general chemistry course the studies of electron configurations, quantum numbers and the periodic table are basic topics. There are several papers describing techniques for improving the process of learning chemistry. Iza and Gil [10] provide a mnemonic method for assigning the electron configurations to atoms. According to Mabrouk [11] the periodic table can be used as a mnemonic device for remembering electron configurations.

Many students have difficulties in learning the oxidation number model. Holder et al. [9] have developed an intelligent computer tutoring tool for assigning oxidation

numbers. Solving quantum number problems has been examined by Ardac [1]. Birk [4] has also developed a computer program (OXRULES) to test the effect of various rules on the assignment of oxidation numbers.

In our study we present an intelligent educational material, which can be used in teaching chemistry. The program was built using the Prolog programming language, in Win-Prolog version 4.600 [13]. We assume that the reader is familiar with the Prolog language.

Our educational material uses a rule-based system for systematic presentation of the characteristics of chemical elements, such as the electron configuration, the oxidation numbers, and the electronegativity. Currently, it consists of 50 rules and 150 facts only. However, the development of the program is still in progress. We believe that our innovative method of using a logic programming language not only improves the process of teaching chemistry, but it can also inspire students to construct other rule-based systems, when they are confronted with a new problem. Thus we can reach our primary objective, namely the improvement of student problem-solving skills.

Figure 1 shows the main window of the intelligent educational material. When the *Periodic table* button is pressed in the main window, a new window appears, containing the periodic table with 111 elements. This is shown in Figure 2. The other buttons lead to various tutorials, as explained later.

Figure 1: The main window of the intelligent educational material

The rest of the paper is structured as follows. In Section 2 we give a short overview of the basics in chemistry, required for understanding the paper. Next, in Section 3, we discuss the three tutorials developed. The paper is concluded with a brief summary in Section 4.

2 A short overview of chemical basics

An atom is composed of a positively charged nucleus and negatively charged electrons. The nucleus consists of positively charged protons and electrically neutral neutrons. An atom is electrically neutral when the number of protons matches the number of electrons. The number of protons determines the atomic number in the periodic table. The electron cloud consists of electron shells and subshells.

An electron orbital is associated with a region in the space where an electron is likely to be found. The electron configuration of atoms can be described using quantum numbers [7]. There are four kinds of quantum numbers:

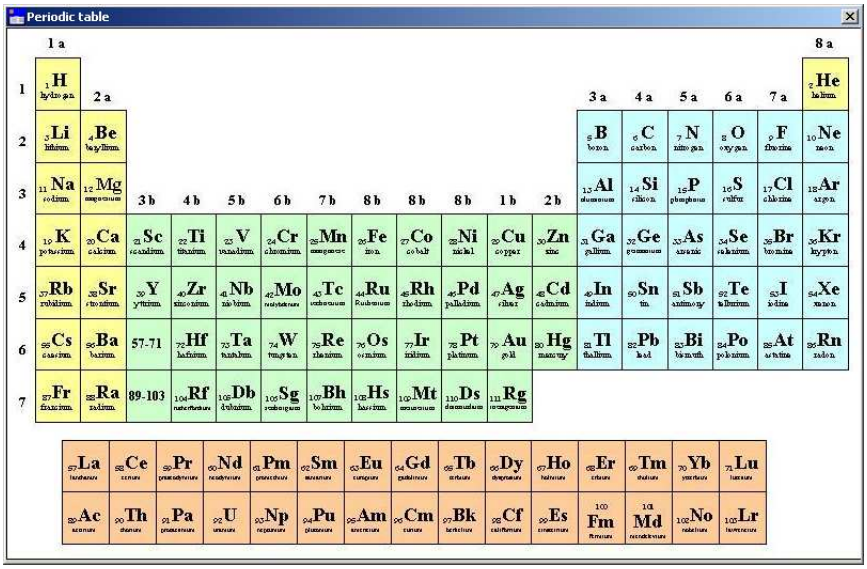

Figure 2: The periodic table, as displayed by the program

- the principal quantum number (n),
- the azimuthal quantum number (l),
- the magnetic quantum number (m_l),
- the spin quantum number (m_s).

The principal quantum number can take the values 1, 2, 3, 4, 5, 6, or 7. The different principal quantum numbers correspond to different shells ($n = 1$ corresponds to the shell K, $n = 2$ to the shell L, etc.; the remaining shells are named M, N, O, P, and Q). The azimuthal quantum number specifies the shape of an atomic orbital. The values of the azimuthal quantum number are $0, 1, \ldots, n-1$, where n is the principal quantum number. Azimuthal quantum numbers are also denoted by letters: $l = 0$ by s, $l = 1$ by p, $l = 2$ by d, $l = 3$ by f, etc. This quantum number also has an orbital meaning, namely it specifies the subshell of the shell [14]. The values of the principal quantum number and the azimuthal quantum number determine the energy level of the electron. Accordingly, the shells and subshells are often referred to as energy levels and sublevels.

The values of the magnetic quantum number can be $-l$, ..., 0, ..., $+l$. The magnetic quantum number determines the energy shift of an atomic orbital due to an external magnetic field. It indicates spatial orientation. The number of the orbitals at a given azimuthal quantum number l is $2 \cdot l + 1$. The maximum number of the orbitals in a shell is n^2, where n is the principal quantum number [5]. Finally,

the spin quantum number is the intrinsic angular momentum of the electron. The values of the spin quantum number are $-\frac{1}{2}$ and $+\frac{1}{2}$.

To determine the electron configuration of an atom we have to use certain rules and principles. According to the Aufbau Principle, electrons fill orbitals starting at the lowest available energy states before filling higher states. Orbitals are generally filled according to the $(n+l)$ rule. This rule states that orbitals with a lower $(n+l)$ value are filled before those with higher $(n+l)$ values. In case of equal $(n+l)$ values, the orbital with a lower n value is filled first.

The Pauli Exclusion Principle states that no two electrons can have the same four quantum numbers. If n, l and m_l are the same m_s must be different, so that the electrons have opposite spins. Therefore a subshell can contain up to $4 \cdot l + 2$ electrons and a shell can contain up to $2 \cdot n^2$ electrons. If multiple orbitals of the same energy are available, Hund's rule says that unoccupied orbitals are filled before occupied orbitals are reused, by electrons having different spins.

The position of an atom in the periodic table is defined by the number of protons of the atom. In the periodic table, rows are called periods and the columns are called groups. There are two types of groups. The primary groups are indicated by the letter '*a*', and the secondary groups are indicated by the letter '*b*'. The periodic table consists of 8 primary groups, 8 secondary groups and 7 periods. The row number is related to the value of the principal quantum number [5].

The valence shell is the outermost shell of an atom. The electrons in the valence shell are referred to as valence electrons. The number of valence electrons determines the number of the primary group and the number of shells determines the number of the period. Valence electrons are important in determining how an element reacts chemically with other elements, i.e. what its chemical behaviour is. In a group (primary or secondary) the number of valence electrons is the same, therefore atoms belonging to the same group have similar chemical properties [5].

The oxidation number is a measure of the degree of oxidation of an atom in a substance. The oxidation number is the real or hypothetical charge that an atom would have if all bonds to atoms of different elements were completely ionic. There are several oxidation number rules. The oxidation number of a free element is zero. In a simple ion the oxidation number is equal to the net charge on the ion. The algebraic sum of oxidation numbers of all atoms in a neutral molecule must be zero.

Electronegativity is a chemical property that describes the power of an atom to attract electrons towards it. Electronegativity cannot be directly measured and must be calculated from other atomic or molecular properties. The most commonly used method of calculation is that originally proposed by Pauling. This gives a dimensionless quantity on a relative scale running from 0.7 to 4.0. In our intelligent program we use these values [5].

3 Tutorials

In this section we discuss three tutorials developed for teaching the principal characteristics of elements.

3.1 Tutorial 1 – Quantum numbers

In this subsection we discuss the features of the program related to answering the following questions:

- What kind of shells are there?
- What kind of subshells has a given shell?
- How many electron orbitals has a given shell?
- How many electron orbitals has a given subshell?
- What is the maximum number of electrons in a given shell?
- What is the maximum number of electrons in a given subshell?
- What kind of magnetic quantum numbers and spin quantum numbers of a given shell and subshell are there, according to the Pauli Exclusion Principle?

To answer the above questions we transform the basics of chemistry, as outlined in Section 2, to the following Prolog facts and rules:

```
principal(1).  principal(2).  principal(3).  principal(4).
principal(5).  principal(6).  principal(7).

azimuthal(0).  azimuthal(1).  azimuthal(2).
azimuthal(3).  azimuthal(4).  azimuthal(5).  azimuthal(6).

magnetic(-6).  magnetic(-5).  magnetic(-4).
magnetic(-3).  magnetic(-2).  magnetic(-1).
magnetic(0).   magnetic(1).   magnetic(2).
magnetic(3).   magnetic(4).   magnetic(5).   magnetic(6).

spin(0.5).     spin(-0.5).

shell(N):-                  principal(N).
subshell(N, L):-            principal(N), azimuthal(L), L<N.
shell_orbitals(N, C) :-     principal(N), C is N*N.
shell_electrons(N, E) :-    principal(N), E is 2*N*N.
subshell_orbitals(L, C) :-  azimuthal(L), C is 2*L+1.
subshell_electrons(L, E) :- azimuthal(L), E is 4*L+2.
quantum(N, L, M, S):-       principal(N), azimuthal(L), L<N,
                            magnetic(M),
                            M>=(-L), M<L+1, spin(S).
```

The first four blocks of Prolog facts list the possible values of the four kinds of quantum numbers (principal, azimuthal, magnetic and spin). Next, the rules for

`shell(N)` and `subshell(N, L)` define the possible shell numbers and the possible pairs of shell and subshell numbers, respectively. Subsequently, given a shell `N` or a subshell `L`, the rules for `shell_orbitals(N, C)`, `shell_electrons(N, E)`, `subshell_orbitals(L, C)`, and `subshell_electrons(L, E)` return the number of corresponding orbitals and electrons. Finally, the rule for `quantum(N, L, M, S)` is capable of listing all allowed combinations of the four kinds of quantum numbers.

The Prolog queries below, when typed in the console window following the `| ?-` prompt, provide the answers to the questions listed at the beginning of the present subsection. The Prolog built-in predicate `fail` is used to enumerate all solutions of a query, while the predicates `write` and `nl` serve for displaying the answer:

```
| ?- shell(J), write(J), nl, fail.
| ?- subshell(3, L), write(3), write(' '), write(L), nl, fail.
| ?- shell_orbitals(3, C).
| ?- shell_orbitals(N, C), write(N), write(' '), write(C), nl, fail.
| ?- shell_electrons(4, E).
| ?- shell_electrons(N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- subshell_orbitals(3, C).
| ?- subshell_orbitals(N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- subshell_electrons(4, E).
| ?- subshell_electrons (N, C), write(N), write(' '), write(C),
       nl, fail.
| ?- quantum(2, 1, M, S), write(M), write(' '), write(S), nl, fail.
| ?- quantum(2, L, M, S), write(L), write(' '),
       write(M), write(' '), write(S), nl, fail.
| ?- quantum(N,L,M,S), write(N), write(' '), write(L), write(' '),
       write(M), write(' '), write(S), nl, fail.
```

There are several of object-oriented extensions of Prolog, such as Visual Prolog, Win-Prolog, etc. These implementations combine the advantages of logic programming and object-oriented programming. As our software for chemistry education was created using Win-Prolog 4.600, we could implement an appropriate graphical interface, which is demonstrated by some examples below.

When the *Quantum numbers* button is pressed in the main window, the panel shown in Figure 3 is displayed.

Next, using the *Shells* button in Figure 3, one obtains the window shown in Figure 4. This window provides important information about the shells. If the *Shells and subshells* button is pressed in Figure 3, then the panel in Figure 5 is presented to the user. Here one can select a shell (e.g. set the principal quantum number to 5). When the *Display* button is pressed in this window, all possible subshell numbers are enumerated in the listbox in the bottom right corner. The values listed are generated by the rule `subshell(N, L)`.

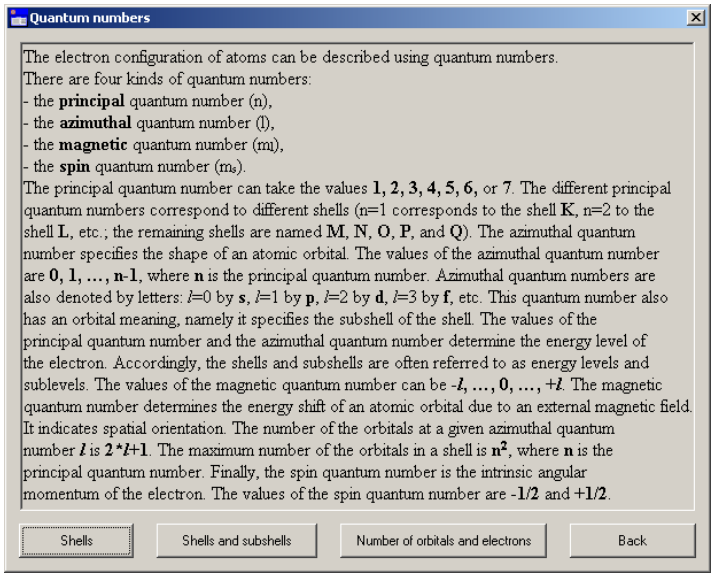

Figure 3: Quantum numbers

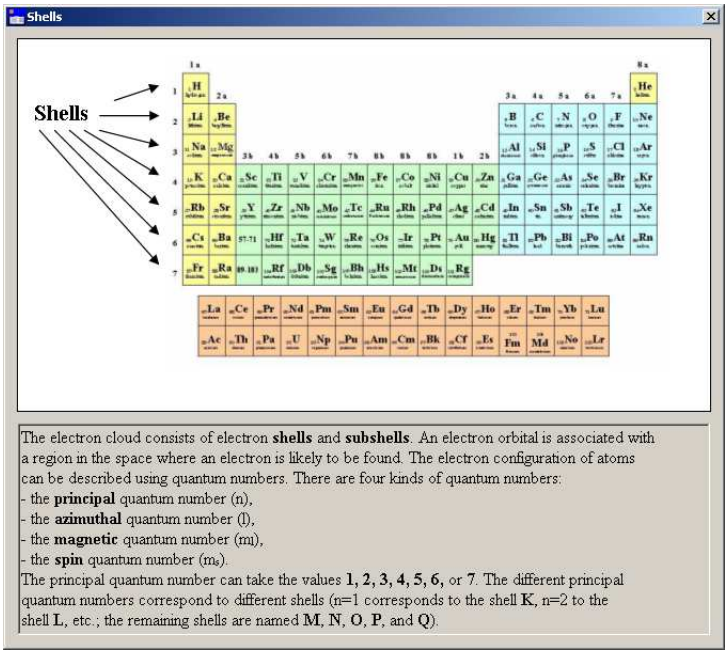

Figure 4: Shells

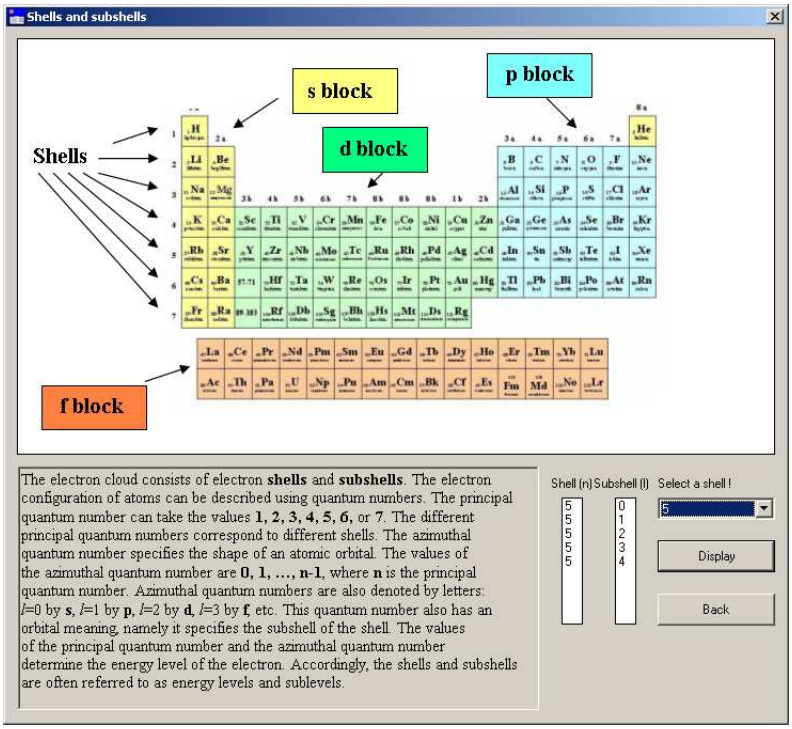

Figure 5: Shells and subshells

Let us return to Figure 3. Using the *Number of orbitals and electrons* button of this window, we get the panel shown in Figure 6. Here, when the user specifies a shell (e.g. by stating that the principal quantum number is 6), and presses the *Display* button, the program enumerates in the listboxes on the left hand side of the panel all possible subshell numbers, and, for each of these, it lists the number of orbitals and electrons. To calculate these possibilities, the program uses the rules `subshell(N, L)`, `subshell_orbitals(L, C)`, and `subshell_electrons(L, E)`.

3.2 Tutorial 2 – Electron configurations of the elements

This subsection deals with obtaining the correct electron configuration of a given element. The program can show which electron orbital is filled, and how many electrons there are in an orbital. The corresponding rules use the Aufbau Principle and the $(n + l)$ rule.

The facts used here are the very same as the ones presented in Section 3.1. The rules are the following:

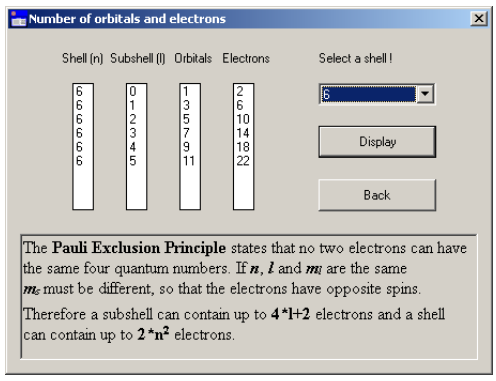

Figure 6: The number of orbitals and electrons

```
electron_orbital(E, N, L):-
        E>0, N<8,
        electron(L, E, E_L, NewE),
        write(N), write(' '), write(L),
        write(' '), write(E_L), nl,
        next_n_l(N, L, NewN, NewL),
        electron_orbital(NewE, NewN, NewL).
electron_orbital(_, N, _):-
        N=8.
electron_orbital(E, _, _):-
        E=0.

electron(L, E, E_L, NewE):-
        E0 is 4*L+2, E>=E0, E_L is E0, NewE is E-E0.
electron(L, E, E_L, NewE):-
        E0 is 4*L+2, E<E0,  E_L is E,  NewE is 0.

next_n_l(N, L, NewN, NewL):-
        L=0, S is N+1, NewL is N // 2, NewN is S-NewL.
next_n_l(N, L, NewN, NewL):-
        L>0, NewN is N+1, NewL is L-1.
```

The predicate `electron_orbital(E, N, L)` has 3 arguments: `E` is the number of electrons, `N` is the principal quantum number, and `L` is the azimuthal quantum number. Its task is to display the correct electron configuration of `E` electrons, starting from the principal quantum number `N` and azimuthal quantum number `L`. To display the complete electron configuration of `E` electrons, the predicate should thus be invoked in the form `electron_orbital(E, 1, 0)`.

The whole predicate can be expressed using three rules. The first rule checks if there are any remaining electrons (`E>0`), and that the principal quantum number

is valid (`N<8`). Next, it calls an auxiliary predicate `electron`, which, given the azimuthal quantum number `L`, splits the number of electrons `E` in two parts: `E_L` is the number of electrons to be placed in the subshell `L`, and `NewE` is the number of remaining electrons (here, `E` = `E_L`+`NewE` always holds). Having displayed the triple (`N`,`L`,`E_L`), a second auxiliary predicate is invoked: `next_n_l(N, L, NewN, NewL)` receives the pair (`N`,`L`) and returns the next such pair in (`NewN`,`NewL`), according to the $(n + l)$ rule. Finally, the predicate is invoked recursively, with the remaining number of electrons, the new shell number, and the new subshell number.

The second and third rule of the predicate `electron_orbital` serve for stopping the recursion, when we run out of shells (`N=8`), or when there are no more electrons to place (`E=0`).

The predicate `electron(L, E, E_L, NewE)` first calculates how many electrons (`E0`) can be in the given subshell `L`. The first rule deals with the case when we have at least `E0` electrons, while in the second rule we have a situation where all the electrons can be placed in the given subshell.

The predicate `next_n_l(N, L, NewN, NewL)` determines the new n and l values using the $(n + l)$ rule. It generates `NewN` from `N`, and `NewL` from `L`. In the first case, when `L=0`, the $n + l$ value increases by 1. In this case `NewL` becomes `N//2` (where `//` is the integer division operator). In the second case, the $n + l$ value does not change: `NewN is N+1` and `NewL is L-1`.

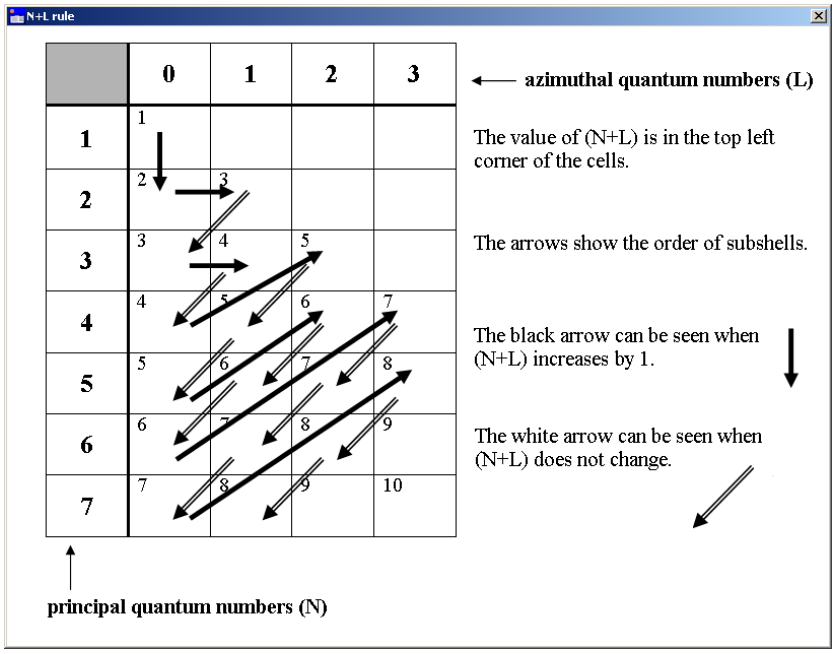

Figure 7: The N+L rule, as displayed by the educational tool

Figure 7 displays the order of subshells, i.e. how the n and l values change in

subsequent invocations of the predicate `next_n_l`.

Figure 8 shows a window for determining the electron configuration. In this particular case the number 80 has been entered, denoting the atomic number, i.e. the number of electrons. When the *Display* button of the window is pressed, the program lists the possible subshells, together with the number of electrons in that subshell, in the bottom left listboxes. It uses the predicate invocation `electron_orbital(E, 1, 0)` to obtain the numbers shown.

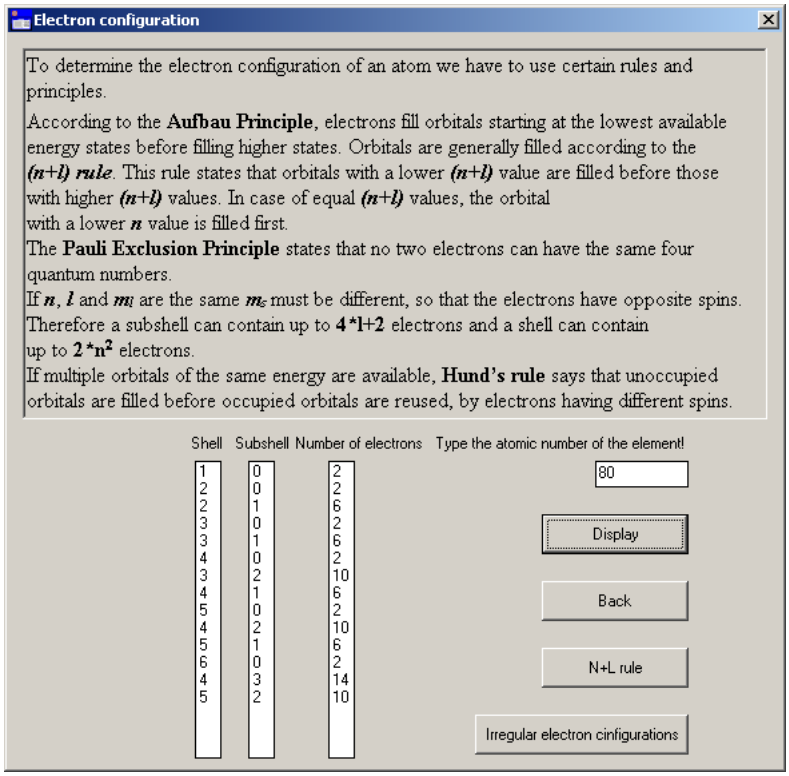

Figure 8: The electron configuration of a given element

There are 20 elements, out of the 111, for which the exact electron configuration cannot be predicted by these rules. The deviation from the regular electron configuration is not significant (only 1 or 2 electrons are in another atomic orbital).

These elements are in the d or f blocks. Some of the irregular electron configurations are predicted on the basis of the stability of half- or wholly filled subshells (for example: Cr, Cu, Mo, Ag or Au). The corresponding configurations are s^1d^5 or s^1d^{10}.

There are some irregular configurations in the f blocks. When atomic orbitals $6s$ and $7s$ are filled, the next electron occupies an orbital d, rather than a $4f$ or a $5f$. But appropriate electrons occupy orbitals f, and orbital d is filled completely

later [16].

To deal with these irregularities, the button labelled *Irregular electron configurations* is provided in the window. When this button is pressed, the program displays a new panel shown in Figure 9. This window shows the elements having an irregular electron configuration.

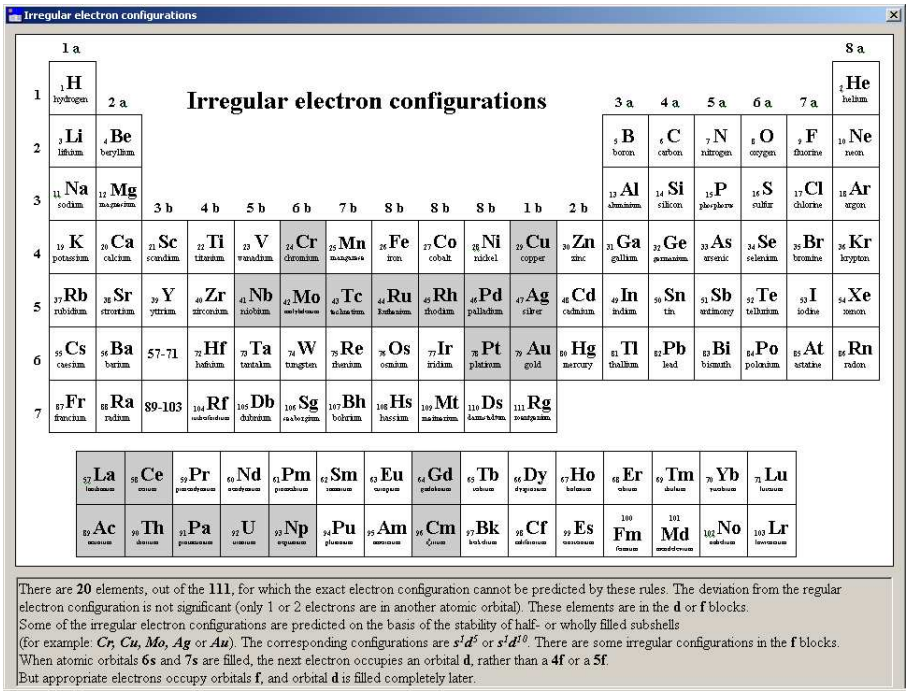

Figure 9: Irregular electron configurations

We believe that it is important that our rule-based system, in addition to determining regular electron configurations, is also able to draw attention to irregular electron configurations. These observations can motivate students to examine such exceptions in more detail.

3.3 Tutorial 3 – Oxidation numbers and electronegativity

In this subsection we show how the program determines the typical oxidation numbers of a given element. Furthermore we discuss questions of the following type:

- Which are the typical oxidation numbers of the elements in a group?
- How much electronegativity has an element?
- If there is a bond between elements, which of the two elements has higher electronegativity?

- How much electronegativity is there in a group?
- How do the electronegativity values change in a period and a group?

To store information on chemical elements, we use the predicate `element`, which has the following 6 arguments:

- the atomic number of the element,
- the name of the element,
- the period number of the element,
- the group number of the element,
- the type of the group: primary (letter '*a*') or secondary (letter '*b*'),
- the value of the electronegativity.

There are 111 known elements in the periodic table. We show only a subset of the 111 facts:

```
element(1, 'hydrogen', 1, 1, 'a', 2.1).
element(2, 'helium', 1, 8, 'a', 0).
element(3, 'lithium', 2, 1, 'a', 1.0).
element(4, 'beryllium', 2, 2, 'a', 1.5).
element(5, 'boron', 2, 3, 'a', 2.0).
element(6, 'carbon', 2, 4, 'a', 2.5).
element(7, 'nitrogen', 2, 5, 'a', 3.0).
element(8, 'oxygen', 2, 6, 'a', 3.5).
element(9, 'fluorine', 2, 7, 'a', 4.0).
element(10, 'neon', 2, 8, 'a', 0).
element(11, 'sodium', 3, 1, 'a', 0.9).
element(12, 'magnesium', 3, 2, 'a', 1.2).
element(13, 'aluminium', 3, 3, 'a', 1.5).
element(14, 'silicon', 3, 4, 'a', 1.8).
element(15, 'phosphorus', 3, 5, 'a', 2.1).
element(16, 'sulfur', 3, 6, 'a', 2.5).
element(17, 'chlorine', 3, 7, 'a', 3.0).
element(18, 'argon', 3, 8, 'a', 0).
element(19, 'potassium', 4, 1, 'a', 0.8).
element(20, 'calcium', 4, 2, 'a', 1.0).
element(21, 'scandium', 4, 3, 'b', 1.3).
element(22, 'titanium', 4, 4, 'b', 1.6).
element(23, 'vanadium', 4, 5, 'b', 1.6).
```

The predicate `oxidation_number`, shown below, has two arguments. The first one is the name of the element and the second one is the possible oxidation number of the element. An element can have several oxidation numbers.

The first rule expresses that for every element, the oxidation number 0 is appropriate. This is because the oxidation number of a free element is zero. Subsequent rules express the relation between the group of the element and the oxidation number.

```
oxidation_number(X, 0):- element(_, X, _, _, _, _).
oxidation_number(X, Y):- element(_, X, _, Y, _, _), Y<6.
oxidation_number(X, Y):- element(_, X, _, Y, 'b', _), Y>5, Y<8.
oxidation_number(X, 3):- element(_, X, _, 5, 'a', _).
oxidation_number(X,-3):- element(_, X, S, 5, 'a', _), S<6.
oxidation_number(X,-2):- element(_, X, S, 6, 'a', _), S<6.
oxidation_number(X, 4):- element(_, X, S, 6, 'a', _), S>2, S<7.
oxidation_number(X,-1):- element(_, X, _, 7, 'a', _).
oxidation_number(X, 1):- element(_, X, S, 7, 'a', _), S>2, S<7.
oxidation_number(X, 5):- element(_, X, S, 7, 'a', _), S>2, S<7.
oxidation_number(X, 3):- element(_, X, 4, O, 'b', _), O>3.
oxidation_number(X, 2):- element(_, X, 4, O, 'b', _), O>4.
oxidation_number(X, 2):- element(_, X, S, 8, 'b', _), S>4, S<7.
oxidation_number(X, 4):- element(_, X, S, 8, 'b', _), S>4, S<7.
```

However, experimental evidence shows that there exist several additional values. These are handled using appropriate Prolog facts, such as the ones listed below.

```
oxidation_number('thallium', 1).  oxidation_number('carbon', -4).
oxidation_number('carbon', 2).    oxidation_number('tin', 2).
oxidation_number('lead', 2).      oxidation_number('nitrogen', 4).
oxidation_number('nitrogen', 2).  oxidation_number('phosphorus', 4).
oxidation_number('oxigen', -1).   oxidation_number('sulfur', 2).
oxidation_number('sulfur', 6).    oxidation_number('selenium', 6).
oxidation_number('tellurium', 6). oxidation_number('polonium', 2).
oxidation_number('chlorine', 3).  oxidation_number('chlorine', 7).
oxidation_number('iodine', 7).    oxidation_number('copper', 2).
oxidation_number('zinc', 1).      oxidation_number('gold', 3).
oxidation_number('mercury', 1).   oxidation_number('vanadium', 4).
```

Building on the predicates `element` and `oxidation_number`, we now define some further predicates, related to the questions listed at the beginning of the present subsection.

The rule for `group_oxnumber(A, B, C)`, shown below, determines the values of oxidation numbers of a given group. The rule for `compare_en(A, B)` can show which element's electronegativity is higher. The rule `electronegativity(X, En)` retrieves the electronegativity of a given element. The two rules `en_period(S, En)` and `en_group(P, Q, En)` produce the values of the electronegativity in a selected group or in a selected period, respectively.

```
group_oxnumber(A, B, C):-
        element(_, X, _, A, B, _),
        oxidation_number(X, C).

compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X>Y,
        write(A), nl.
compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X=Y,
        write('They have equal electronegativity'), nl.
compare_en(A, B):-
        element(_, A, _, _, _, X),
        element(_, B, _, _, _, Y), X<Y,
        write(B), nl.

electronegativity(X, En):-
        element(_, X, _, _, _, En).

en_group(P, Q, En):-
        element(_, _, _, P, Q, En).

en_period(S, En):-
        element(_, _, S, _, _, En).
```

If one types the following queries into the console window, one can get the answers to some questions of the type listed at the beginning of the present subsection.

```
| ?- group_oxnumber(4, 'a', C), write(C), nl, fail.
| ?- group_oxnumber(1, 'a', C),  write(C), nl, fail.
| ?- compare_en('carbon',  'hydrogen').
| ?- compare_en('hydrogen',  'hydrogen').
| ?- compare_en('bromine',  'chlorine').
| ?- oxidation_number('carbon', C), write(C), nl, fail.
| ?- electronegativity('carbon',  En).
| ?- en_group(5, 'a', En), write(En), nl, fail.
| ?- en_group(3, 'b', En), write(En), nl, fail.
| ?- en_period(5, En), write(En), nl, fail.
```

Figure 10 shows two example questions submitted in the interactive environment: we ask the oxidation numbers of a given element (17 – chlorine) and of a group (3.b).

When you press the *Display* button on the left hand side of the window, the program lists the possible oxidation numbers of a given element in the listbox. For this, it uses the set of `element` facts. When you press the *Display* button on the

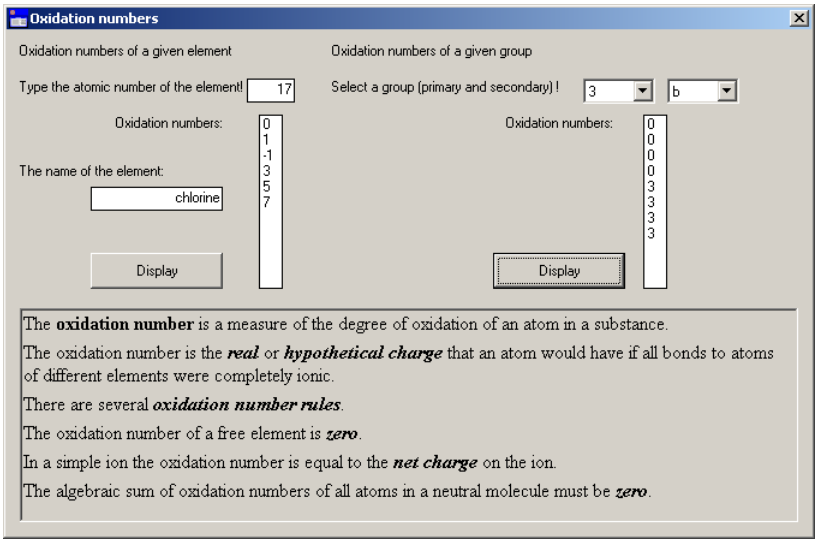

Figure 10: Possible oxidation numbers of a given element and a group

right hand side of the window, the program enumerates in the listbox the possible oxidation numbers of the given group. It uses the `group_oxnumber(A, B, C)` rule for this.

In Figure 11 we present a panel of the program showing the value of the electronegativity of a given element (80 - mercury), of a period (4), and of a group (1.a).

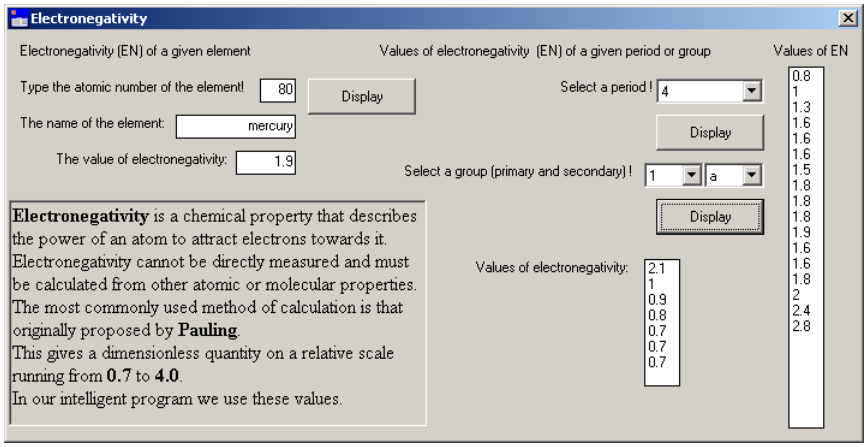

Figure 11: The electronegativity of a given element, period and a group

When you press the *Display* button on the left hand side of the window, the program displays the value of the electronegativity of the given element. It uses the `element` facts. When you press the *Display* button on the right hand side of the window, the program lists the possible values of the electronegativity of a given period and a given group in the listboxes. The `en_period(S, En)` and the `en_group(P, Q, En)` rules are used in this task.

4 Summary

Improving problem-solving skills is a significant aim of education. Logic programming languages are useful tools for computer-assisted problem-solving. Logic programs consist of facts and rules, therefore one can construct rule-based expert systems easily. One can use these tools in chemistry education, too. There are a lot of facts and rules in several subfields of chemistry, learning these is often very difficult for the students. To improve the effectiveness of our educational activities we have to develop new teaching methods. One of such methods is to create (with the active co-operation of the students) a specific rule-based expert system, which contains facts and rules of the given subject.

While constructing the system, the students understand and memorise important facts and rules with much less effort than using traditional learning techniques. If they are confronted with an exception, they are motivated to ask questions, and try to answer these. Thus, we can reach our main objective, namely the improvement of the problem-solving skills of the students.

The development of this educational material is in progress. Future work includes predicting possible chemical bonds and reactions, and determining reaction equations. Our aim is not only to create educational materials, but to build a proper chemical expert system, which also includes an electronic user guide.

References

[1] Ardac, D. Solving quantum number problems: An examination of novice performance in terms of conceptual base requirements. *Journal of Chemical Education*, 79(4):510–513, 2002.

[2] Birk, J. P. *Predicting Inorganic Reactions: The Development of an Expert System in Expert System Applications in Chemistry.* American Chemical Society Symposium Series No.408. ACS Books, Washington, D.C., 1989.

[3] Birk, J. P. The computer as student - an application of artificial intelligence. *Journal of Chemical Education*, 69(4):294–295, 1992.

[4] Birk, J. P. Oxidation number rules: A program to test the effect of various rules on the assignment of oxidation numbers. *Journal of Chemical Education Software*, 6B1, 1993.

[5] Bodonyi, F. *Summary of Chemistry (in Hungarian).* Műszaki Könyvkiadó, Budapest, 1987.

[6] Borgulya, I. *Expert systems, methods and applications (in Hungarian).* ComputerBooks Kiadói Kft., Budapest, 1995.

[7] Brücher, E. *General Chemistry (Structure), educational material (in Hungarian).* KLTE Szervetlen és Analitikai Kémiai Tanszéke, Debrecen, 1992.

[8] Heller, S. R.and Bigwood, D. W. Expert systems in chemistry – applications to data quality. *Proceedings of the 2nd ICIK'87*, pages 99–120, 1987.

[9] Holder, D. A., Johnson, B. G., and Karol, P. J. A consistent set of oxidation number rules for intelligent computer tutoring. *Journal of Chemical Education*, 79(4):465–467, 2002.

[10] Iza, N. and Gil, M. A mnemonic method for assigning the electronic configurations of atoms. *Journal of Chemical Education*, 72(11):1025–1026, 1995.

[11] Mabrouk, S. T. The periodic table as a mnemonic device for writing electronic configurations. *Journal of Chemical Education*, 80(8):894–896, 2003.

[12] Sántáné-Tóth, E. *Knowledge-based Technology, expert systems (in Hungarian).* Dunaújvárosi Főiskola Kiadói Hivatala, Dunaújváros, 2000.

[13] Shalfield, R., Spenser, C., Steel, B., and Westwood, A. *WIN-PROLOG User Guide.* Logic Programming Associates Ltd., London, 2007.

[14] Strong, J. A. The periodic table and electron configurations. *Journal of Chemical Education*, 63(10):834–836, 1986.

[15] Szlávi, P. and Zsakó, L. Informatics as a particular field of education. *Teaching Mathematics and Computer Science*, 3(2):283–294, 2005.

[16] Tőkés, B. and Donáth-Nagy, G. *Chemical lectures and laboratory exercises (in Hungarian).* Scientia Kiadó, Kolozsvár, 2002.