An automated proof of Arrow's theorem

Kenryo INDO

Kanto Gakuen University, 200 Fujiagu, Ota, Gunma, 373-8515 Japan

Abstract. This paper presents a simple PROLOG implementation for social choice problem. A model of society consists of alternatives, agents, and preferences of agents. Social welfare function (SWF) organizes different preferences among agents into a social preference. Kenneth J. Arrow (1963) proved that any SWF which satisfies a set of conditions of the IIA, of the Pareto, and of the unrestricted domain should be dictatorial. My PROLOG program for 3 alternatives and 2 agents proves the Arrow's impossibility theorem. With a minor modification, a version of the theorem without the Pareto condition firstly proved by Wilson (1972) also can be computed by using my program.

Keywords: social choice theory, impossibility theorem, PROLOG.

1. Introduction

Social choice theory deals with the following problem: Suppose we have a group of agents facing a number of alternatives collectively; each agent has a preference over which alternative is chosen individually. Which alternatives should the group as a whole adopt?

Examples of this problem are familiar from everyday life: Let us suppose that a class has a choice between three possible time slots for the TA's office hours, say, either morning (m), afternoon (n), or evening (e). Each student has a preference with respect to one or the other. The instructor has to balance these preferences to determine a time slot.

Kenneth J. Arrow (1963) proved such a mission is impossible if he/she can not suppose any restriction on the preference of students. More precisely, there is no social welfare function (SWF) which satisfies simultaneously a set of conditions, (U), (P), (IIA), and *not* (D). The intuitive contents of the 4 conditions are as follows.

- *Unrestricted domain* (U): Every logically possible preference never to be inhibited.
- *Pareto condition* (P): *Unanimity*. If they both prefer X to Y then X must be chosen.
- *Independence of irrelevant alternatives* (IIA): For any preference if society prefers X to Y, so does unless someone changes his/her preference between X and Y.
- *Dictatorship* (D): Society prefers X to Y whenever a same single member does do.

In other words, if there are at least three alternatives, any social choice rule which satisfies (P) and (IIA) must be dictatorial (D) assuming all logically possible preferences permitted (Arrow (1963), pp.51-9, pp.97-100; Sen (1995), p.4).

PROLOG (i.e., programming in logic) is familiar to computer scientists, especially in AI community. By using it, the logical content of the impossibility theorem may be translated into a rule (a *Horn*-clause),

(rule A)

```
is_Arrovian_SWF(Swf):-

    is_logically_possible(Swf),

    is_Pareto_consistent(Swf),

    is_IIA_consistent(Swf),

    not( is_dictatorial(Swf) ).
```

The code I wrote in reality is of recursion over possible profiles (in section 3). Nevertheless a rule below may indicate the declarative meaning of that theorem. Specifically since the set of conditions {(*not* D), U, IIA, P} is *unsatisfiable* by Arrow's theorem. The theorem can be equivalently modeled as follows.

(rule B)

```
is_dictatorial(Swf):-

    is_logically_possible(Swf),

    is_Pareto_consistent(Swf),

    is_IIA_consistent(Swf).
```

PROLOG system successfully prove the condition (D), in the left hand side ("the *head*" of the clause) if the conjunctive (U), (P), and (IIA) in the right hand side ("the *body*" of the clause) are satisfied simultaneously.

The proof procedure such as stated above is called "the *resolution principle*" a computational realization of syllogisms. Then the system attempts to a "pattern match" to the SWF, a variable *term* in above code, generating possible social welfare functions modeled by the conjunctive by try and error (i.e., the backtracks). Rule A should have failed. The rule B can succeed with a matched term (or possibly keep up until it can not prove the cases any more).

However, because of the exponentiation, it is impossible to prove by means of naive "generate and test" method. Even in a small case of, for example, 2 agents and 3 alternatives, there are 6 possible rankings for strict ordering model, and 6^{36} possible

candidates of SWF defined over 6^2 possible profiles. In weak order case, there are 13 possible rankings permitting tie, and 13^{169} possible candidates for the SWF defined over 13^2 possible profiles. Precisely, for the case of n agents and m alternatives, there are $(m!^n)$th power of m! candidates in strict preferences.

The impossibility theorem provides good exercise to write efficient code with recursion. The code I made actually is of the case of 2 agents with 3 alternatives. It may be extended for larger, but the complexity is exponential order. It seems a challenging work to design a simple proof for the theorem automatically.

The paper organizes as follows. Next section shows an example of PROLOG modeling for social choice problem. In Section 3 our system can provide an automated proof of the Arrow's impossibility theorem for the case of 3 alternatives and 2 agents. Further, with minor modification we have a generalized impossibility result without the Pareto condition firstly proven by Robert Wilson (1972) in Section 4. Section 5 argues a visualization of the process of proof with a version of my program in binary relations. Section 6 mentions the related research fields. Last section concludes.

2. Logic programming for social choice problem

In this section I will show how PROLOG can be applied to model a social choice problem by using the example in previous section. For the ISO-standard grammar and system predicates, see Cocksin and Mellish (2003). First of all, I model the alternatives and the individuals, next the individual or social rankings over the alternatives. I conclude with a possible construction for the social welfare function.

2-1. individual preferences

Now, suppose two students, Alice and Bob, who are prefer morning to evening, and evening to morning respectively. Other pairs of comparison are same for both students, say, morning to afternoon, and afternoon to evening.

In PROLOG, the set of alternatives, the members of the society, and the preferences of the members respectively may be modeled as the following facts.

```
alternatives( [m,a,e]).

members_of_society( [alice,bob]).

preference( alice, [(m,e),(m,a),(a,e)]).

preference( bob, [(e,m),(m,a),(a,e)]).
```

Traditionally the rational (individual/social) choice theory assumes that a complete, transitive, and anti-symmetric for strict preference models. So we can consistently use lists for the basic representation of preferences if the agent's preference is rational in this sense. For the Alice's preference, it can be stated by using, `[m, a, e]`, a list as

```
ranking( alice, [m,a,e]).
```

While Bob's preference obviously circulates by transitive reasoning, so it can not be represented as a list. Instead, suppose the Bob's preference

```
preference( bob, [(e,m),(a,m),(a,e)]).
```

Or equivalently,

```
ranking( bob, [a,e,m]).
```

Weak preferences can be handled by minor modifications, by adding 3-1 elements list which signifies the indifferences. But we will focus on strict cases in this paper for the sake of simplicity.

In summary, the preferences of Alice and Bob are coded simply as follows.

```
ranking( alice, [m,a,e]).

ranking( bob, [a,e,m]).
```

2-2. binary relations

Next I show a rule with variables to reproduce the binary preference relations from the rankings.

```
prefer_x_to_y( J, X, Y, R):-

   ranking( J, R),

   x_precedes_y_in_list( X, Y, R).

x_precedes_y_in_list( X, Y, R):-

   append( _,[X|Z],R),

   member( Y, Z).
```

A variable in PROLOG starts with a capital letter which will be matched during the query, or with an underscore when anonymously used.

2-3. reasoning about preference

The PROLOG system acts as if proving a theorem, or seeking a given goal and solving a set of constraints. Figure 2-1 shows such an example in the context of integer domain.

```
?- member(X, [1,2,3]),member(Y,[3,4,5]),
|    member(Z,[13,17]),Z is X^2+Y^2.
X = 1
Y = 4
Z = 17 ;

X = 2
Y = 3
Z = 13 ;

No
?-
```

Figure 2-1. The goal is to find a pair of integers (X, Y) such that X is in {1, 2, 3}, Y is in {3, 4, 5} and the sum of squares is 13 or 17.

```
?- prefer_x_to_y( alice, m, Y, R).

Y = a
R = [m, a, e] ;

Y = e
R = [m, a, e] ;

No
?-
```

Figure 2-2. What (Y) is not preferred to morning by Alice who has ranking R.

It seeks a clause which has the negation of goal predicate as the head and produces sequentially the subsidiary goals, predicates in the body and repeats until there is no other solution. Figure 2-2 shows a sample execution of reasoning in previous social choice problem, the goal is a `prefer_x_to_y/4` and the subsidiary goals are a `ranking/2` and an `x_precedes_y_in_list/3`.

I would like to interpret the query in Figure 2-2 such like as that "What (Y) is not preferred to morning by Alice who has ranking R." And the output of `PROLOG` system that "I found the case where her ranking is m>a>e and either 'afternoon' or 'evening' is preferred." Such a dialog can be simulated easily, but I omit because of limited space. Subsequently, I verbalize a proof for the impossibility theorem in Section 3 and Appendix A.

2-4. logically possible preferences

It may be beneficial to the later analyses that in advance we prepare the 6 logically possible rankings in `PROLOG` database with numberings.

```
possible_ranking_0([m,a,e], r(1)).

possible_ranking_0([m,e,a], r(2)).

possible_ranking_0([a,m,e], r(3)).

possible_ranking_0([a,e,m], r(4)).

possible_ranking_0([e,m,a], r(5)).

possible_ranking_0([e,a,m], r(6)).
```

It is noteworthy that if some clauses were removed, we would have a model of restricted domain.

Then we rewrite a rule for possible rankings of the individual agent and the society.

```
possible_ranking( J, R):-

   members_of_society( N),

   member( J,N),

   possible_ranking_0( _, r(R)).
```

Next we can write a rule for possible profile of individual rankings.

```
possible_ranking_profile((R1,R2)):-

   possible_ranking( alice,R1),

   possible_ranking( bob, R2).
```

2-5. social welfare function

A social welfare function (SWF) selects a ranking for each possible profile of rankings of all the member of society. Next I give a dictatorial rule (to Alice) at some profile in previous example.

```
dictatorial_swf_for_alice( R1, R2, R1):-

   ranking( alice,R1),

   ranking( bob, R2).
```

But the instructor would think a non-dictatorial way, for example, by a majority vote or unanimity rule. However, Arrow's theorem tells us that he/she can never success unless violating the conditions referred in introductory section.

A code which generates the SWF recursively is such a below.

```
try_SWF_assignment_1( [], []).

try_SWF_assignment_1( [(R1,R2)->R|F], [(R1,R2)|Q]):-

   try_SWF_assignment_1( F, Q),

   possible_ranking( society, R).

is_logically_possible_SWF( F):-

   findall( P, possible_ranking_profile(P), Q),

   try_SWF_assignment_1( F, Q).
```

PROLOG programmers frequently use recursions in order to compute iteratively, instead of For or While statements in other conventional programming languages which are not allowed in the standard PROLOG language.

2-6. display in tabular form

Next code displays the generated SWF in a table format without labels or lines.

```
show_swf(F):-

   nl,

   forall(

     bagof( S, Q^member((_,Q)->S,F), L),

     (nl,write(L))

   ).
```

The user may easily modify the table into a specified form. The code is left as an exercise for the reader.

3. Automated proof of the impossibility theorem

This section models firstly the Pareto condition and the IIA condition which constraint on the SWF. Then I modify the rule to generate SWF recursively incorporates these constraints. Lastly, I show an automated proof of the Arrow's theorem.

3-1. the Pareto condition

The Pareto condition (P) can be translated into a rule in PROLOG such as

```
is_Pareto_consistent( (R1,R2)->R ):-
   not(
     violates_Pareto_condition( _,(R1,R2)->R)
   ).


violates_Pareto_condition((X,Y),(R1,R2)->R):-
   prefer_x_to_y( X, Y, R1),
   prefer_x_to_y( X, Y, R2),
   possible_ranking_0( _, R),
   not( prefer_x_to_y( X, Y, R)).
```

That is, if the rankings of students are same, also the society is. This is not exact literally for the condition, since originally stated in binary form. Indeed it is rather weak, but the consequence of the theorem is same.

3-2. the IIA condition

The condition (IIA) informally means such as "What society prefers from a pair of alternatives always can be determined irrelevant to what a true preference profile is except for that pair."

Firstly, I rewrite rules for binary preferences and their profiles for the simplicity of that for the condition (IIA).

```
prefer_x_to_y_B( (X, Y), R, B):-
   distinct_pair(X,Y),
   (prefer_x_to_y( X, Y, R)->B=1;B=0).
```

```
binary_choice_profile( (X, Y), (R1,R2), (B1,B2)):-
   prefer_x_to_y_B( (X, Y), R1, B1),
   prefer_x_to_y_B( (X, Y), R2, B2).
```

By next rule PROLOG system try to catch contradicted pairs of alternatives at a profile given attempting assignments to a candidate SWF.

```
violates_IIA_condition( (X, Y), (P, Q)-> R, F):-
   not( F=[]),
   member( (P1,Q1)-> S, F),
   binary_choice_profile( (X, Y), (P,Q), B),
   binary_choice_profile( (X, Y), (P1,Q1), B),
   not( binary_choice_profile( (X,Y), (R,S), (A,A))).
```

Then a test of the (IIA) for a given SWF candidate may be written such a rule below.

```
is_IIA_consistent( (R1,R2)->R, F):-
   not((
     violates_IIA_condition( _, (R1,R2)->R, F)
   )).
```

But this code is not useful in order to generate the SWF because of the complexity, so we should revise it. Please note that if we confine the possible_ranking_0/2 to only 2 alternatives the IIA is vacuously satisfied

3-3. coping with the complexity

Thus, the combinatorial nature just fit the logic programming. However, complexity problem arises. Even in this small case there are 6^6^2 possible candidates of SWF (i.e., the 6 possible rankings over the 36 possible profiles of rankings).

In order to prove, or disprove, the Arrow's theorem, you never adopt a naïve generate-and-test method, so I modify previous rule for SWF by incorporating above constraints checking both (P) and (IIA) accordingly.

```
   try_SWF_assignment_2( [], []).

   try_SWF_assignment_2( [(R1,R2)->R|F], [(R1,R2)|Q]):-
```

```
        try_SWF_assignment_2( F, Q),

        possible_ranking_0( _, R),

        is_Pareto_consistent( (R1, R2)-> R),

        is_IIA_consistent( (R1, R2)->R, F).

    is_Arrovian_SWF( F):-

        findall( P, possible_ranking_profile(P), Q),

        try_SWF_assignment_2( F, Q).
```

Therefore we can rule out violations to the Arrovian SWF cumulatively. In order to proof the theorem, simply type the following query at a prompt (and an enter key) which results in the two dictatorial SWFs (see Figure 3-1).

```
?- is_Arrovian_SWF(A),

show_swf(A), nl,write('---'),fail.
```

Indeed, the code will lightly produce only dictatorial SWFs either for Alice or Bob. In the upper table form in Figure 3-1, the Bob's dictatorial SWF is shown which has 6 rows each of which is the ranking of him. As for the lower table form, the Alice's dictatorial SWF consists of columns each of which is just her ranking. (Appendix A shows the verbalization of the proof.) There is no room for possibility of democratic decision making, so the impossibility theorem is proven.

```
?- is_Arrovian_SWF(A), show_swf(A), nl,write('---'),fail.

[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
---
[r(1), r(1), r(1), r(1), r(1), r(1)]
[r(2), r(2), r(2), r(2), r(2), r(2)]
[r(3), r(3), r(3), r(3), r(3), r(3)]
[r(4), r(4), r(4), r(4), r(4), r(4)]
[r(5), r(5), r(5), r(5), r(5), r(5)]
[r(6), r(6), r(6), r(6), r(6), r(6)]
---

No
?-
```

Figure 3-1. A proof of the Arrow's theorem

4. Impossibility without the Pareto condition

A version of the Arrow's theorem without the Pareto condition but under weak citizen sovereignty condition can be proven. The result firstly proved by Wilson (1972) for weak preferences.

– *Citizen sovereignty* (CS): Every logically possible ordered pair should be included in the SWF at least at one preference profile.

4-1. the Wilson's theorem

For strict preferences, since total indifference (i.e., the null SWF) is ruled out, there only exists either dictatorial or anti-dictatorial. A proof for the theorem and the code can be readily obtained as follows.

```
try_SWF_assignment_4( [], []).

try_SWF_assignment_4( [(R1,R2)->R|F], [(R1,R2)|Q]):-

   try_SWF_assignment_4( F, Q),

   possible_ranking_0( _, R),

   is_IIA_consistent( (R1, R2)->R, F).


is_almost_Arrovian_SWF( F, K):-

   findall( P, possible_ranking_profile(P), Q),

   try_SWF_assignment_4( F, Q),

   setof( R, P^member( P->R, F), S),

   length( S, K).
```

After a reload with the code above, the proof will be obtained by following command. The results are shown as Figure 4-1.

```
?- is_almost_Arrovian_SWF(A,6),

show_swf(A), nl,write('---'),fail.
```

```
?- is_almost_Arrovian_SWF(A,6), show_swf(A), nl,write('---'),fail.

[r(6), r(6), r(6), r(6), r(6), r(6)]
[r(4), r(4), r(4), r(4), r(4), r(4)]
[r(5), r(5), r(5), r(5), r(5), r(5)]
[r(2), r(2), r(2), r(2), r(2), r(2)]
[r(3), r(3), r(3), r(3), r(3), r(3)]
[r(1), r(1), r(1), r(1), r(1), r(1)]
---
[r(6), r(4), r(5), r(2), r(3), r(1)]
[r(6), r(4), r(5), r(2), r(3), r(1)]
[r(6), r(4), r(5), r(2), r(3), r(1)]
[r(6), r(4), r(5), r(2), r(3), r(1)]
[r(6), r(4), r(5), r(2), r(3), r(1)]
[r(6), r(4), r(5), r(2), r(3), r(1)]
---
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
[r(1), r(2), r(3), r(4), r(5), r(6)]
---
[r(1), r(1), r(1), r(1), r(1), r(1)]
[r(2), r(2), r(2), r(2), r(2), r(2)]
[r(3), r(3), r(3), r(3), r(3), r(3)]
[r(4), r(4), r(4), r(4), r(4), r(4)]
[r(5), r(5), r(5), r(5), r(5), r(5)]
[r(6), r(6), r(6), r(6), r(6), r(6)]
---

No
?-
```

Figure 4-1. A proof of the Wilson's theorem

4-2. the Wilson's theorem revisited

Now, we have a result which summarizes the experimentations using above code for varying K=1, 2, …, 6. K denotes the number of distinct ordered pairs included in the SWF. To do this, I added a rule, `test_Wilson`/1, but I omit here.

```
1111111111111111111111111111111111
2222222222222222222222222222222222
3333333333333333333333333333333333
4444444444444444444444444444444444
5555555555555555555555555555555555
6666666666666666666666666666666666
6:swfs found at level:1
```

Figure 4-2 (a). A proof of a version of Wilson's theorem: K=1

```
2122111111112122112122111111111111111  6466444444446466446466444444444444444
1211221111111211221211221111111111111  4644664444444644664644664444444444444
2222221111112222222222221111111111111  6666664444446666666666664444444444444
3311313311311111111111113311311111111  4444446466444444444444446466446466444
1133131133131111111111111133131111111  6466446466446466446466446466446466444
3333333333331111111111113333331111111  4644666466444644664644 646644646644
1111112122111111111111112122112122 11  6666666466446666666666666466446466444
2122112122112122112122112122112122 11  2222222222222222222225552225552225555
1211222122111211221211222122112122 11  5552225552225552222225552225552225555
2222222122112222222222222122112122 11  2225552225552225552225552225552225555
1111111111113311313311311111113311311  5555555555555555552225552225552225555
3311313311313311313311313311313311311  2222222222222222225555555555555555555
1133131133133311313311311133133311311  5552225552225552225555 555555555555555
3333333333333311313311313333333311311  2225552225552225555555555555555555555
1111111211221111111111111211221211 22  6655656655655555555555565565555555555
2122111211222122112122111211221211 22  5566565566565555555555556656555555555
1211221211221211221211221211221211 22  6666666666665555555555566666655555555
2222221211222222222222221211221211 22  5555555555556655656655655555556655555
1111112222221111111111112222222222222  6655656655656655656655656655656655565
2122112222222122112122112222222222222  5566556566566556 665565566566566556565
1211222222221211221211222222222222222  6666666666666655656655656666666655656
5552225552225552222222222222222222222  5555555555555566565566565555555566566
2225552225552225552222222222222222222  6655656655655566565566566556556566566
5555555555555555552222222222222222222  5566565566565566565566565566565566566
2222222222222222225552225552225552222  6666666666665566565566566666665566566
5552225552225552225552225552225552222  4444444644664444444444444644664644664
2225552225552225555552225552225552222  6466444644664646 64464664446446464466
5555555555555555555552225552225552222  4644664644664644664644664644664644664
1111111111111133131133131111111133133  6666664644666666666666664644664644664
3311313311311133131133133331131133133  5555555555556666666666665555556666666
1133131133131133131133131133131133133  6655656655656666666666666655656666666
3333333333331133131133133333333133133  5566565566566666666666665566566666666
1111111111113333333333331111113333333  4444446666664444444444446666666666666
3311313311313333333333333311313333333  6466446666666 64664464664466666666666
1133131133133333333333333113313333333  4644666666664644664644666666666666666
4443334443334443333333333333333333333  84:swfs found at level:2
```

Figure 4-2 (b). A proof of a version of Wilson's theorem: K=2

```
0:swfs found at level:3
0:swfs found at level:4
0:swfs found at level:5
```

Figure 4-2 (c). A proof of a version of Wilson's theorem: K=3,4,5

```
66666644444455555522222233333311111 1
64523164523164523164523164523164523 1
12345612345612345612345612345612345 6
11111122222233333344444455555566666 6
4:swfs found at level:6
```

Figure 4-2 (d). A proof of a version of Wilson's theorem: K=6

See Figures 4-2 (a), (b), and (c). First case, K=1, consists of 6 constant SWFs each of which is obviously unaffected by the IIA respectively. Second case, K=2, has 84, nontrivial, two-valued, all of which consist of precisely 4 ordered pairs. Three cases, K=3, 4, 5 are empty. Figure 4-2(d), K=6, shows a reproduction of previous section. Under the condition below, K=6 is only permissible, so it proves the theorem.

5. Binary relation based modeling

Previous sections demonstrated how PROLOG can be applied for the preference based modeling and the social choice theory. In this section, I argue a version of previous program which uses the binary representation for preferences.

Under the condition IIA, we are permitted to model the preferences for each pair of alternatives, instead of the rankings, for the individuals and the society, so that the IIA condition has embedded into the code. It is a double recursion exercise of PROLOG, inessential to demonstrate here, so, the code has moved to appendix B.

My code (in appendix B) remodels the same problem in previous sections in terms of the binary relations. Figure 5-1 shows that the Arrow's theorem is proved again. And with minor modifications, as I commented in the source code, it also proves for Wilson's. (The code is left as an exercise for the reader.)

```
?- swf_bin(W), show_swf_bin(W),fail.

pair: (m, a)
    [ (m, m->m), (m, a->a)]
    [ (a, m->m), (a, a->a)]
pair: (m, e)
    [ (m, m->m), (m, e->e)]
    [ (e, m->m), (e, e->e)]
pair: (a, e)
    [ (a, a->a), (a, e->e)]
    [ (e, a->a), (e, e->e)]
end of SWF
pair: (m, a)
    [ (m, m->m), (m, a->m)]
    [ (a, m->a), (a, a->a)]
pair: (m, e)
    [ (m, m->m), (m, e->m)]
    [ (e, m->e), (e, e->e)]
pair: (a, e)
    [ (a, a->a), (a, e->a)]
    [ (e, a->e), (e, e->e)]
end of SWF

No
?-
```

Figure 5-1. Another proof of the Arrow's theorem

6. Related work

Many proofs of Arrow's theorem and the versions have been proposed. But it seems me not always clear in paper based proofs how a dictatorship (or the set of decisiveness), establishes by propagating these constraints from one profile/pair to another. Even graphical representations proposed by some researchers seem not always clear of the additional operations required nevertheless for the best.

PROLOG has been applied for many real industrial or business problems other than solving puzzles (i.e., the expert systems). But at least as long as I know there is no previous attempt to apply PROLOG to the social choice theory, only except for the Nash implementation theory by the author (Indo, 2002).

There are a few where the proof of Arrow's theorem is concerned, even by using computer programming. The only exception I found was Takekuma (1997) who used Mathematica to prove the Arrow's theorem for the same case as present paper under weak preferences. But his proof was governed manually, not just automated, and the program is not comprehensible without knowledge about the specific software.

However, my program consists of the declarable PROLOG codes which proves the impossibility theorem automatically and visualizes the resulted SWF by table forms. Therefore the user may capture a clearer in the process of the proof how the IIA and the P constraints the domain of social choice respectively.

Indeed the proof of impossibility theorem may be seen as solving constraint satisfaction problem (and using the constraint propagation technique). I showed a PROLOG implementation of such an idea in present paper by a traditional recursive goal seeking for the Arrovian SWF.

In my personal opinion, the proof can be seen as a precursor of constraint based logic programming technologies (Hooker, 2000) because it use the domain specific "reduction" and "branching" so as to prove the dictatorship of the SWF.

Because of the logical modeling it has merits of symbolic representations without translating the problem into an integer programming. It may be compared with the solver software for mathematical optimization, (and they relates certainly in several aspects (Hooker, 2000)).

But why computer simulations in general not widely used in social choice problems as yet except for the probability of voting paradoxes which had come into forty or thirty years ago by behavioral scientists?

I think one reason is the transparency of modeler's thought which maps the application domain in to the programming codes. Unlike conventional optimization problems, such like for linear (or quadratic) programming, shortest path network algorithm, or assignment problem, the conditions over preference domain models are only artificially translated into the constraints of the integer programming as well as the game theoretic model of "matching."

Finally, I would like to relate my model to the mechanism design (and so to game theory). Its computational version (Nisan and Ronen, 2001) has significant applications for combinatorial auction design, network traffic control, group meeting scheduling systems, etc. But for unrestricted domain, there is no non-dictatorial rule which is not manipulable by Gibbard-Satterthwaite theorem. It is logically equivalent to Arrow's theorem and can be proved in the same manner. The PROLOG version is also available from the author.

7. Conclusion

In this paper I presented a simple PROLOG implementation for social choice problem and the impossibility theorems (Arrow, 1963; Wilson, 1972). Thereby I try a new tack to study social choice problems which would be called SCPSLP (Social Choice Problem Solver in Logic Programming).

PROLOG turns out to be useful in studying basic social choice theory and developing tools to scrutinize the combinatorial properties of the social welfare function. It rather fits (in less than 150 lines!) and provides a clearer grasp for the logical structure of the theorem and proof. The users of my system, at least potentially, would be researchers or students in economics, or political science, the traditional applications of the theory as well as in computer science.

By Wilson's theorem we know that the key successful factor of non-dictatorial SWF is to find a subset of profiles where the (IIA) condition does not hold such that the decisive set can not be decomposed into the subsets. This domain restriction method will be examined in another paper.

References

[1] Arrow, K. J. (1963). "Social Choice and Individual Values," 2nd edition, Yale University Press. (Originally published by Wiley)
[2] Clocksin, W. F. and C. S. Mellish (2003). "Programming in Prolog : Using the ISO Standard," 5th edition, Springer.
[3] Hooker, J. (2000). "Logic-base Methods for Optimization: Combining Optimization and Constraint Satisfaction," Wiley.
[4] Indo, K. (2002). Implementing Nash implementation theory in PROLOG, mimeo.
[5] Nisan, N. and A. Ronen (2001). Algorithmic mechanism design, *Games and Economic Behavior* **35**: 166-196.
[6] Sen, A. (1995). Rationality and social choice, *The American Economic Review* **38**: 1-24.
[7] Takekuma, S. (1997). A proof of arrow's impossibility theorem by MATHEMATICA, *Hitotsubashi Journal of Economics* **38**: 139-148.
[8] Wilson, R. (1972). Social choice theory without the Pareto principle, *Journal of Economic Theory* **5**: 478-486.

Appendix A. The proof process put into words

We want to show how the IIA condition, accompanied with the Pareto condition, propagates among pairs of alternatives and constraints possible assignments for the SWF under the transitivity (and the completeness) of preferences. A verbalization of the automated proof for the first dictatorial SWF in section 3 is as follows.

```
% file output start time , [date(2006/3/25), time(18:19:37)]

%----------  start from here ------------%

[1]: A plan : (r(6), r(6))->r(1) violates the Pareto condition against the pair : (e, a)
[2]: A plan : (r(6), r(6))->r(2) violates the Pareto condition against the pair : (e, m)
[3]: A plan : (r(6), r(6))->r(3) violates the Pareto condition against the pair : (e, a)
[4]: A plan : (r(6), r(6))->r(4) violates the Pareto condition against the pair : (e, a)
[5]: A plan : (r(6), r(6))->r(5) violates the Pareto condition against the pair : (a, m)
[6]: I suppose at profile : (r(6), r(6))->r(6) without violation so far.
The SWF under construction: ----------------------------------6
[7]: A plan : (r(6), r(5))->r(1) violates the Pareto condition against the pair : (e, a)
[8]: A plan : (r(6), r(5))->r(2) violates the Pareto condition against the pair : (e, m)
[9]: A plan : (r(6), r(5))->r(3) violates the Pareto condition against the pair : (e, a)
[10]: A plan : (r(6), r(5))->r(4) violates the Pareto condition against the pair : (e, a)
[11]: I suppose at profile : (r(6), r(5))->r(5) without violation so far.
The SWF under construction: ----------------------------------56
[12]: A plan : (r(6), r(4))->r(1) violates the Pareto condition against the pair : (e, m)
[13]: A plan : (r(6), r(4))->r(2) violates the Pareto condition against the pair : (e, m)
[14]: A plan : (r(6), r(4))->r(3) violates the Pareto condition against the pair : (e, m)
[15]: I suppose at profile : (r(6), r(4))->r(4) without violation so far.
The SWF under construction: ----------------------------------456
[16]: A plan : (r(6), r(3))->r(1) violates the Pareto condition against the pair : (a, m)
[17]: A plan : (r(6), r(3))->r(2) violates the Pareto condition against the pair : (a, m)
[18]: I suppose at profile : (r(6), r(3))->r(3) without violation so far.
The SWF under construction: ----------------------------------3456
[19]: A plan : (r(6), r(2))->r(1) violates the Pareto condition against the pair : (e, a)
[20]: I suppose at profile : (r(6), r(2))->r(2) without violation so far.
```

The SWF under construction: ----------23456
[21]: I suppose at profile : (r(6), r(1))->r(1) without violation so far.
The SWF under construction: ----------123456
[22]: A plan : (r(5), r(6))->r(1) violates the Pareto condition against the pair : (e, m)
[23]: A plan : (r(5), r(6))->r(2) violates the Pareto condition against the pair : (e, m)
[24]: A plan : (r(5), r(6))->r(3) violates the Pareto condition against the pair : (e, m)
[25]: A plan : (r(5), r(6))->r(4) violates the Pareto condition against the pair : (e, a)
[26]: I suppose at profile : (r(5), r(6))->r(5) without violation so far.
The SWF under construction: ----------5123456
[27]: A plan : (r(5), r(5))->r(1) violates the Pareto condition against the pair : (e, m)
[28]: A plan : (r(5), r(5))->r(2) violates the Pareto condition against the pair : (e, m)
[29]: A plan : (r(5), r(5))->r(3) violates the Pareto condition against the pair : (e, m)
[30]: A plan : (r(5), r(5))->r(4) violates the Pareto condition against the pair : (e, a)
[31]: I suppose at profile : (r(5), r(5))->r(5) without violation so far.
The SWF under construction: ----------55123456
[32]: A plan : (r(5), r(4))->r(1) violates the Pareto condition against the pair : (e, m)
[33]: A plan : (r(5), r(4))->r(2) violates the Pareto condition against the pair : (e, m)
[34]: A plan : (r(5), r(4))->r(3) violates the Pareto condition against the pair : (e, m)
[35]: A plan : (r(5), r(4))->r(4) violates the IIA condition against the pair : (m, a) and : (r(5), r(6)->r(5)).
[36]: A plan : (r(5), r(4))->r(5) violates the IIA condition against the pair : (a, e) and : (r(6), r(1)->r(1)).
[37]: A plan : (r(5), r(4))->r(6) violates the IIA condition against the pair : (m, a) and : (r(5), r(6)->r(5)).
[38]: A plan : (r(5), r(5))->r(6) violates the Pareto condition against the pair : (m, a)
[39]: I suppose at profile : (r(5), r(6))->r(6) without violation so far.
The SWF under construction: ----------6123456
[40]: A plan : (r(5), r(5))->r(1) violates the Pareto condition against the pair : (e, m)
[41]: A plan : (r(5), r(5))->r(2) violates the Pareto condition against the pair : (e, m)
[42]: A plan : (r(5), r(5))->r(3) violates the Pareto condition against the pair : (e, m)
[43]: A plan : (r(5), r(5))->r(4) violates the Pareto condition against the pair : (e, a)
[44]: I suppose at profile : (r(5), r(5))->r(5) without violation so far.
The SWF under construction: ----------56123456
[45]: A plan : (r(5), r(4))->r(1) violates the Pareto condition against the pair : (e, m)
[46]: A plan : (r(5), r(4))->r(2) violates the Pareto condition against the pair : (e, m)
[47]: A plan : (r(5), r(4))->r(3) violates the Pareto condition against the pair : (e, m)
[48]: I suppose at profile : (r(5), r(4))->r(4) without violation so far.
The SWF under construction: ----------456123456
[49]: A plan : (r(5), r(3))->r(1) violates the IIA condition against the pair : (m, a) and : (r(5), r(4)->r(4)).
[50]: A plan : (r(5), r(3))->r(2) violates the IIA condition against the pair : (m, a) and : (r(5), r(4)->r(4)).
[51]: I suppose at profile : (r(5), r(3))->r(3) without violation so far.
The SWF under construction: ----------3456123456
[52]: A plan : (r(5), r(2))->r(1) violates the Pareto condition against the pair : (e, a)
[53]: I suppose at profile : (r(5), r(2))->r(2) without violation so far.
The SWF under construction: ----------23456123456
[54]: I suppose at profile : (r(5), r(1))->r(1) without violation so far.
The SWF under construction: ----------123456123456
[55]: A plan : (r(4), r(6))->r(1) violates the Pareto condition against the pair : (a, m)
[56]: A plan : (r(4), r(6))->r(2) violates the Pareto condition against the pair : (a, m)
[57]: A plan : (r(4), r(6))->r(3) violates the Pareto condition against the pair : (e, m)
[58]: I suppose at profile : (r(4), r(6))->r(4) without violation so far.
The SWF under construction: ----------4123456123456
[59]: A plan : (r(4), r(5))->r(1) violates the Pareto condition against the pair : (e, m)
[60]: A plan : (r(4), r(5))->r(2) violates the Pareto condition against the pair : (e, m)
[61]: A plan : (r(4), r(5))->r(3) violates the Pareto condition against the pair : (e, m)
[62]: A plan : (r(4), r(5))->r(4) violates the IIA condition against the pair : (m, a) and : (r(6), r(1)->r(1)).
[63]: A plan : (r(4), r(5))->r(5) violates the IIA condition against the pair : (a, e) and : (r(4), r(6)->r(4)).
[64]: A plan : (r(4), r(5))->r(6) violates the IIA condition against the pair : (a, e) and : (r(4), r(6)->r(4)).
[65]: A plan : (r(4), r(6))->r(5) violates the Pareto condition against the pair : (a, m)
[66]: I suppose at profile : (r(4), r(6))->r(6) without violation so far.
The SWF under construction: ----------6123456123456
[67]: A plan : (r(4), r(5))->r(1) violates the Pareto condition against the pair : (e, m)

[68]: A plan : (r(4), r(5))->r(2) violates the Pareto condition against the pair : (e, m)
[69]: A plan : (r(4), r(5))->r(3) violates the Pareto condition against the pair : (e, m)
[70]: A plan : (r(4), r(5))->r(4) violates the IIA condition against the pair : (a, e) and : (r(4), r(6)->r(6)).
[71]: I suppose at profile : (r(4), r(5))->r(5) without violation so far.
The SWF under construction: ----------56123456123456
[72]: A plan : (r(4), r(4))->r(1) violates the Pareto condition against the pair : (a, m)
[73]: A plan : (r(4), r(4))->r(2) violates the Pareto condition against the pair : (a, e)
[74]: A plan : (r(4), r(4))->r(3) violates the Pareto condition against the pair : (e, m)
[75]: I suppose at profile : (r(4), r(4))->r(4) without violation so far.
The SWF under construction: ----------456123456123456
[76]: A plan : (r(4), r(3))->r(1) violates the Pareto condition against the pair : (a, m)
[77]: A plan : (r(4), r(3))->r(2) violates the Pareto condition against the pair : (a, e)
[78]: I suppose at profile : (r(4), r(3))->r(3) without violation so far.
The SWF under construction: ----------3456123456123456
[79]: A plan : (r(4), r(2))->r(1) violates the IIA condition against the pair : (a, e) and : (r(4), r(5)->r(5)).
[80]: I suppose at profile : (r(4), r(2))->r(2) without violation so far.
The SWF under construction: ----------23456123456123456
[81]: I suppose at profile : (r(4), r(1))->r(1) without violation so far.
The SWF under construction: ----------123456123456123456
[82]: A plan : (r(3), r(6))->r(1) violates the Pareto condition against the pair : (a, m)
[83]: A plan : (r(3), r(6))->r(2) violates the Pareto condition against the pair : (a, m)
[84]: A plan : (r(3), r(6))->r(3) violates the IIA condition against the pair : (a, e) and : (r(4), r(2)->r(2)).
[85]: A plan : (r(3), r(6))->r(4) violates the IIA condition against the pair : (a, e) and : (r(4), r(2)->r(2)).
[86]: A plan : (r(3), r(6))->r(5) violates the Pareto condition against the pair : (a, m)
[87]: I suppose at profile : (r(3), r(6))->r(6) without violation so far.
The SWF under construction: ----------6123456123456123456
[88]: A plan : (r(3), r(5))->r(1) violates the IIA condition against the pair : (m, e) and : (r(3), r(6)->r(6)).
[89]: A plan : (r(3), r(5))->r(2) violates the IIA condition against the pair : (m, e) and : (r(3), r(6)->r(6)).
[90]: A plan : (r(3), r(5))->r(3) violates the IIA condition against the pair : (m, e) and : (r(3), r(6)->r(6)).
[91]: A plan : (r(3), r(5))->r(4) violates the IIA condition against the pair : (a, e) and : (r(3), r(6)->r(6)).
[92]: I suppose at profile : (r(3), r(5))->r(5) without violation so far.
The SWF under construction: ----------56123456123456123456
[93]: A plan : (r(3), r(4))->r(1) violates the Pareto condition against the pair : (a, m)
[94]: A plan : (r(3), r(4))->r(2) violates the Pareto condition against the pair : (a, m)
[95]: A plan : (r(3), r(4))->r(3) violates the IIA condition against the pair : (m, e) and : (r(3), r(5)->r(5)).
[96]: I suppose at profile : (r(3), r(4))->r(4) without violation so far.
The SWF under construction: ----------456123456123456123456
[97]: A plan : (r(3), r(3))->r(1) violates the Pareto condition against the pair : (a, m)
[98]: A plan : (r(3), r(3))->r(2) violates the Pareto condition against the pair : (a, m)
[99]: I suppose at profile : (r(3), r(3))->r(3) without violation so far.
The SWF under construction: ----------3456123456123456123456
[100]: A plan : (r(3), r(2))->r(1) violates the IIA condition against the pair : (a, e) and : (r(3), r(5)->r(5)).
[101]: I suppose at profile : (r(3), r(2))->r(2) without violation so far.
The SWF under construction: ----------2345612345612345612345
[102]: I suppose at profile : (r(3), r(1))->r(1) without violation so far.
The SWF under construction: ----------12345612345612345612345
[103]: A plan : (r(2), r(6))->r(1) violates the Pareto condition against the pair : (e, a)
[104]: A plan : (r(2), r(6))->r(2) violates the IIA condition against the pair : (m, e) and : (r(3), r(4)->r(4)).
[105]: A plan : (r(2), r(6))->r(3) violates the Pareto condition against the pair : (e, a)
[106]: A plan : (r(2), r(6))->r(4) violates the Pareto condition against the pair : (e, a)
[107]: A plan : (r(2), r(6))->r(5) violates the IIA condition against the pair : (m, a) and : (r(5), r(3)->r(3)).
[108]: I suppose at profile : (r(2), r(6))->r(6) without violation so far.
The SWF under construction: ----------612345612345612345612345
[109]: A plan : (r(2), r(5))->r(1) violates the Pareto condition against the pair : (e, a)
[110]: A plan : (r(2), r(5))->r(2) violates the IIA condition against the pair : (m, e) and : (r(2), r(6)->r(6)).
[111]: A plan : (r(2), r(5))->r(3) violates the Pareto condition against the pair : (m, a)
[112]: A plan : (r(2), r(5))->r(4) violates the Pareto condition against the pair : (m, a)
[113]: I suppose at profile : (r(2), r(5))->r(5) without violation so far.
The SWF under construction: ----------5612345612345612345612345

[114]: A plan : (r(2), r(4))->r(1) violates the IIA condition against the pair : (m, e) and : (r(2), r(5)->r(5)).
[115]: A plan : (r(2), r(4))->r(2) violates the IIA condition against the pair : (m, e) and : (r(2), r(5)->r(5)).
[116]: A plan : (r(2), r(4))->r(3) violates the IIA condition against the pair : (m, e) and : (r(2), r(5)->r(5)).
[117]: I suppose at profile : (r(2), r(4))->r(4) without violation so far.
The SWF under construction: ---------456123456123456123456123456
[118]: A plan : (r(2), r(3))->r(1) violates the IIA condition against the pair : (m, a) and : (r(2), r(4)->r(4)).
[119]: A plan : (r(2), r(3))->r(2) violates the IIA condition against the pair : (m, a) and : (r(2), r(4)->r(4)).
[120]: I suppose at profile : (r(2), r(3))->r(3) without violation so far.
The SWF under construction: --------3456123456123456123456123456
[121]: A plan : (r(2), r(2))->r(1) violates the Pareto condition against the pair : (e, a)
[122]: I suppose at profile : (r(2), r(2))->r(2) without violation so far.
The SWF under construction: -------23456123456123456123456123456
[123]: I suppose at profile : (r(2), r(1))->r(1) without violation so far.
The SWF under construction: ------123456123456123456123456123456
[124]: A plan : (r(1), r(6))->r(1) violates the IIA condition against the pair : (m, a) and : (r(2), r(3)->r(3)).
[125]: A plan : (r(1), r(6))->r(2) violates the IIA condition against the pair : (m, a) and : (r(2), r(3)->r(3)).
[126]: A plan : (r(1), r(6))->r(3) violates the IIA condition against the pair : (m, e) and : (r(2), r(4)->r(4)).
[127]: A plan : (r(1), r(6))->r(4) violates the IIA condition against the pair : (a, e) and : (r(3), r(2)->r(2)).
[128]: A plan : (r(1), r(6))->r(5) violates the IIA condition against the pair : (m, a) and : (r(2), r(3)->r(3)).
[129]: I suppose at profile : (r(1), r(6))->r(6) without violation so far.
The SWF under construction: -----6123456123456123456123456123456
[130]: A plan : (r(1), r(5))->r(1) violates the IIA condition against the pair : (m, e) and : (r(1), r(6)->r(6)).
[131]: A plan : (r(1), r(5))->r(2) violates the IIA condition against the pair : (m, e) and : (r(1), r(6)->r(6)).
[132]: A plan : (r(1), r(5))->r(3) violates the Pareto condition against the pair : (m, a)
[133]: A plan : (r(1), r(5))->r(4) violates the Pareto condition against the pair : (m, a)
[134]: I suppose at profile : (r(1), r(5))->r(5) without violation so far.
The SWF under construction: ----56123456123456123456123456123456
[135]: A plan : (r(1), r(4))->r(1) violates the IIA condition against the pair : (m, e) and : (r(1), r(5)->r(5)).
[136]: A plan : (r(1), r(4))->r(2) violates the Pareto condition against the pair : (a, e)
[137]: A plan : (r(1), r(4))->r(3) violates the IIA condition against the pair : (m, e) and : (r(1), r(5)->r(5)).
[138]: I suppose at profile : (r(1), r(4))->r(4) without violation so far.
The SWF under construction: ---456123456123456123456123456123456
[139]: A plan : (r(1), r(3))->r(1) violates the IIA condition against the pair : (m, a) and : (r(1), r(4)->r(4)).
[140]: A plan : (r(1), r(3))->r(2) violates the Pareto condition against the pair : (a, e)
[141]: I suppose at profile : (r(1), r(3))->r(3) without violation so far.
The SWF under construction: --3456123456123456123456123456123456
[142]: A plan : (r(1), r(2))->r(1) violates the IIA condition against the pair : (a, e) and : (r(1), r(5)->r(5)).
[143]: I suppose at profile : (r(1), r(2))->r(2) without violation so far.
The SWF under construction: -23456123456123456123456123456123456
[144]: I suppose at profile : (r(1), r(1))->r(1) without violation so far.
The SWF under construction:123456123456123456123456123456123456
%---------- end of data ------------%
% file output end time , [date(2006/3/25), time(18:19:38)]

Appendix B. The source code for binary relation based modeling[1]

```
% The source code of "An automated proof of Arrow's theorem:
% A version in binary relations"
% By Kenryo INDO (Kanto Gakuen University)
% program name: swf_bin.pl (23 Mar 2006)
% language: PROLOG (tested on swi-prolog 5.0.10)

% modeling preference relations
```

[1] Earlier versions are downloadable. (http://www.us.kanto-gakuen.ac.jp/indo/front_e.html)

```
%-----------------------------------------------------
alternatives( [m,a,e]).

possible_ranking_0([m,a,e], r(1)).
possible_ranking_0([m,e,a], r(2)).
possible_ranking_0([a,m,e], r(3)).
possible_ranking_0([a,e,m], r(4)).
possible_ranking_0([e,m,a], r(5)).
possible_ranking_0([e,a,m], r(6)).

x_precedes_y_in_list( X, Y, R):-
  append( _,[X|Z],R), member( Y, Z).

prefer_x_to_y( X, Y, Rn):-
  possible_ranking_0( R, Rn),
  x_precedes_y_in_list( X, Y, R).

distinct_pair_lex( X, Y):- distinct_pair( X, Y),
  alternatives(A),
  x_precedes_y_in_list( X, Y, A).

distinct_pair( X, Y):-
  alternatives( A),
  member(X, A),
  exsists_in_preference(X),
  member(Y, A),
  Y \= X,
  exsists_in_preference(Y).

exsists_in_preference(X):-
  \+ \+ (
    possible_ranking_0(R,_),
    member( X,R)
  ).

% SWF in binary comparisons
%-----------------------------------------------------
swf_bin(W):-
  findall( (X,Y), distinct_pair_lex(X,Y),Lb),
  swf_bin_1( Lb, W),
  \+ inconsistent_bin( W, _).

swf_bin_1([],[]).
swf_bin_1([(X,Y)|L],[(X,Y):Fb|W]):-
  swf_bin_1(L,W),
  swf_bin_2([X,Y],(X,Y),Fb),
 %noimposed_bin((X,Y),Fb),  %use this for Wilson's theorem
  true.

swf_bin_2([],_,[]).
swf_bin_2([Rb1|H1],(X,Y), [Swf|W]):-
  swf_bin_2(H1,(X,Y), W),
```

```
   swf_bin_3(Rb1,(X,Y),[X,Y],Swf).

swf_bin_3( _, _,[],[]).
swf_bin_3( Rb1,(X,Y),[Rb2|H2],[(Rb1,Rb2)->Rb|F]):-
   swf_bin_3( Rb1,(X,Y),H2,F),
   unanimity_bin((X,Y),[Rb,Rb1,Rb2]),  %comment out if without (P).
   member(Rb,[X,Y]).

unanimity_bin((X,_),[X,X,X]).
unanimity_bin((_,Y),[Y,Y,Y]).
unanimity_bin((_,_),[_,R,S]):- not(R = S).

inconsistent_bin( [B1:Fma,B2:Fme,B3:Fae], [V1,V2,V3]):-
   flatten( Fma, Wma),
   flatten( Fme, Wme),
   flatten( Fae, Wae),
   member( (A,S)->P, Wma), V1= B1: ((A,S)->P),
   member( (B,T)->Q, Wme), V2= B2: ((B,T)->Q),
   member( (C,U)->R, Wae), V3= B3: ((C,U)->R),
   Circulars= [(m,e,a),(a,m,e)],
   \+ member( (A,B,C), Circulars),
   \+ member( (S,T,U), Circulars),
   member( (P,Q,R), Circulars).

noimposed_bin((X,Y),Fb):-
   flatten(Fb,Fb1),
   \+ \+ member(_->X,Fb1),
   \+ \+ member(_->Y,Fb1).

% tabular form display
%--------------------------------------------------
show_swf_bin(W):-
   distinct_pair(X,Y),
   show_swf_bin((X,Y),W),
   fail.
show_swf_bin(_):-
   nl,
   write('end of SWF').
show_swf_bin((X,Y),W):-
   member((X,Y):Fb,W),
   nl,
   write(pair:(X,Y)),
   forall(
     member(A,Fb),
     (nl,tab(4),write(A))
   ).

% end of the program
%--------------------------------------------------
```