Implementing Nash Implementation Theory on Prolog: A Logic Programming Approach for Rational Agent Systems

By Kenryo Indo

Faculty of Economics, Department of Management, Kanto Gakuen University, Fujiagu 200, Ota, 3738515, Japan.

Abstract. The objective of Nash implementation theory is to design the decentralized society of rational agents with some game-theoretical mechanisms by means of which it attains the desirable state of the world with respect to the criteria (i.e., social choice rule). In this paper, a simulation modeling using Prolog for the game-theoretical mechanisms and social choice environment of rational agents has demonstrated. In principle, it is straightforward to devise the rational social choice environment and the abstract game theoretical mechanisms via the logic programming technique of Prolog. And we can verify the conditions of Nash-implementability and the related conditions easily, and even we can generate or mutate social choice rules or preference domains to satisfy desirable conditions. We demonstrated simulation results of Nash implementaion for several small models, and also observed impossibility results of Hurwicz-Schmeidler and Sen-Peleg on “ud22”, a small unrestricted domain.

Contents:

1. Introduction

Simulation modeling for the society as a multi-agent system (i.e., a system of artificial agents) is the new stream of the social science. It has foundations from (distributed) artificial intelligence, the theories of computationally implemented intelligent programs such as the knowledge representation, the automated-reasoning, the learning techniques by neural networks or reinforcement updating rules, etc. However, traditionally most influential formalization of this field is of a logic based (For example, Shoham(1994), Worldridge(2000)).

Whereas there are also candidates for the theoretical foundations of a multi-agent systems from traditional veins of mathematical theory of rational decision making and its related computational modeling and simulation prevailing in economic / management science-------Such as the decision theory, the game theory, the social choice theory, the competitive theory of market and price, the mathematical optimization and mathematical programming, the statistical data analysis, etc. In the recent literature, for example, Kraus(2001) analyses multi-agent negotiation systems by the game theoretical framework .

The new approach has set out the common question, which can be thought as a classical one in social science, to both engineers and researchers who is interested in multi-agent systems. How the cooperation of distributed autonomous agents can be attained? How the social objective and the each individual's objective are balanced? Is there any desirable conditions for the institutive / administrative rules for such agent society? What the appropriate communication protocol in order to achieve this, or what the minimal one in that it can reduce the traffic of messages among them? If there exists the institutive rules and the communication protocols for the society which has the given desirable conditions, how to design and test it? And that the theory with validated proof in mathematical language is also computationally tractable if appropriately implemented on the computer?

In order to answer above questions operatively, among the vein of normative or prescriptive theories of decision-making, we will focus on Nash implementation theory (Maskin, 1998; Maskin and Sjöström, 2002). In the context of the theory, a social planner or a game / mechanism designer has the objective to design a decentralized mechanism for the

society of idealistic rational agents by autonomous activity of them the desirable state of the world with respect to a given criteria has to be achieved.

In the language of game theorists, formally, “implementation” of the criteria that determine desirable outcomes of society must be targeted by a SCC (social choice correspondence), which is a map from the possible preference profiles of the agents to a subsets of the possible outcomes. Given an SCC and a possible preference profiles, the planner tries to design a rule of the game, i.e., a mechanism that is the tuple of game forms, thereby the SCC outcomes to be achieved, even if the true state of the world (what the agents mutually know) is not known to the planner.

Game theorist will try to predict that the autonomously maximizing behavior of agents lead to some tuple of best responses, i.e., Nash equilibrium. And the planner as a game theorist will devise such a mechanism with some game forms (i.e., a game in formal sense without agents' true preferences, which maps the acts played by agents to the game outcomes) and some solution concept, especially by “Nash equilibrium”, which predicts the result of the game play by autonomous rational agents as a tuple (i.e., profile) of best responses of all agents.

Both of what the general conditions if the SCC is implementable (the necessary part) and the conditions as for what the implementable SCCs is (the necessary and sufficient part) have been uncovered in the literature up to the early 1990's. The necessary condition is monotonicity introduced by E.Maskin in 1970's. Maskin proved with his mechanism it is also sufficient if N>2 and NVP is satisfied. But NVP is not necessary (Maskin, 1998).

Whereas the field has been developed by other researchers, as for the necessary and sufficient condition including N=2, it was many years later since the problem has set out, condition μ (i.e., strong monotonicity, restricted veto power, and unanimity) for $N \geq 3$, and condition $\mu 2$ (i.e., in addition to the condition, self-selection with maximal elements) for N=2 has found by the two groups of game theorists independently (Moore and Repullo, 1990; Dutta and Sen, 1991). However the attainable set C(a, Rj) and the set B are open in their conditions. Sjöström introduced an iterative algorithm to specify them (Sjöström, 1991), and Suh(1996) extended the algorithm to check the Strong Nash implementability. If an

unrestricted domain (i.e., all possible combination of orderings or its strict part), the condition reduced to essential monotonicity for N≥3, and additionally two conditions for N=2, individual rationality (no blocked outcome) and MR-property (Danilov, 1992). And despite of these conditions in complete information setting, using more elaborated mechanisms, such as virtual implementation, any SCC is implementable.

The current paper devises a Prolog based modeling and simulation for Nash implementation theory with the conditions and mechanisms devised by the authors cited above and so on. Computational environment for simulation in the paper is notebook PC NEC LM50H/6, Celeron(tm) processor, 256MB RAM, Windows Me. However, in very small domain with a few agents and several outcomes, I will report on the Prolog based simulation provides the useful toolbox to pursue such as the agents' rational preferences, the verifying or designing the game-theoretical mechanisms and various conditions under which various social choice rules can be achieved via Nash equilibria.

The rest of the paper is organized as follows: Section 2 review the Prolog based agent architecture to be exploited later. Section 3 describes a simple model of the rational choice domain and the social choice correspondence (SCC) via Prolog. Section 4 describes theoretical objects in the Nash implementation. These are game theoretic mechanisms, message space for mechanism, best response of agent, and Nash equilibrium. Section5, it is the heart of the paper, summarizes both of the Nash implementation theorems and the Prolog simulation results of checking Nash implementability of some examples of appeared in the literature and a small unrestricted domain (I will call it "ud22"). In section 6, we utilize the backtracking incorporated in Prolog, to demonstrate an automated method of verification for the various conditions of implementability with a fixed domain model across possible SCCs on the "ud22". The "ud22" is the model of unrestricted domain with two agents and two outcomes, and the strict preference of them over the outcomes. Finally, we briefly discuss on the relation to the notion of "bounded rationality" in section 7.

2. Making artificial intelligent agent using Prolog

In this section we briefly review the Prolog, a computer language which

is familiar to the AI researchers, but is not so the economic scientists. Prolog is a one of the famous computer language to develop artificial intelligent reasoners or problem solvers who has the ability of logical reasoning, planning, natural language processing, learning knowledge, and any goal seeking system operating in the virtual discrete symbolic domain. It has own detabase (DB) system in which the knowledge resources (i.e., horn-clauses) that may be stored or abolished dynamically and the agent can use them as the fact or the inference rule to prove that the given goal proposition is true. In this paper, I used the “Swi-Prolog version 1.9.0”, a portable and fast “WAM”(Warren Abstract Machine) based Prolog implementation, provided by Jan Wielemaker, University of Amsterdam, in order to develop and test the experimental system.

Automated deduction by Prolog

If you ask about Prolog syntax, predicates, and how to use it, probably Wielemaker’s manual on Swi-Prolog (Wielemaker, 1994) is useful. An artificial intelligent agent which has implemented by Prolog (see Shoham, 1994) can be incorporated both of the method of knowledge representation subsystem based on the first order predicate logic and list notation both of them are pervasive in AI research, and the automated recursive theorem proving subsystem based on the “resolution principle” with the “unification algorithm” by R.A.Robinson. For example the term “X”, a variable, is unified with “a”, “b”, “f(a,Z)”, and so on.

A “resolution” is the outcome of syllogism, the basic inference rule in logics, with refutation procedure. If the user put a goal predicate to the system, by automated-backtracking mechanism of which incorporated in, the Prolog system try to seek the head of clause in its DB to match the name of predicate and its “arity” (i.e., the number of arguments.) and to unify the terms in the head and then the body of that clause recursively until the fact has found such that the terms which satisfies an original goal and all derivative subgoals from the goal simultaneously.

For example, if it has the knowledge base that consists of 2 facts and 1 rule such as { bird(a). fly(b). fly(X):- bird(X). }, as an example of Prolog programming, the agent can deduce the goal "?- fly(Z)”, Z is unified as “Z=a” or “Z=b”. This can be interpreted as the sentence that “Both the individual ‘a’ and the individual ‘b’ can fly.”

Logicl representation for knowledge

Each clause in Prolog DB is a “horn-clause” (i.e., a finite disjunctive of literals with at most a single positive literal, Lh, “Lh or not Lb1 or not Lb2 or … or not Lbn.”, or equally, “Lh←Lb1 and Lb2 and … and Lbn.”) must end with a period “.”. Any literal is a term or term with its negation.

The de facto standard on the syntax is of “Edinburgh” Prolog, In the above example, each of “a”, ”b”, “X”, “bird(a)” and “fly(X)” and so on is a “term”. A term may be an individual symbol, a variable symbol (a variable is start with any upper case alphabet or “_” if anonymous), a strings that is symbols with a pair of single quotation as the delimiter, a function of them using functor, or a list which is explained bellow. Each of “bird” or “fly” is the functor of these terms.

The symbol “:-“, called “neck”, is special functor which separates the rule the head “fly(X)”, the left -hand-side of “:-“, and the body, “bird(X)”, the right-hand-side of it which represents the (conjunction of) conditions or subgoals should be satisfied if the goal “bird(X)” succeeds. The conditional “←” (if) is its analog in mathematical logics. The variable term in head and body of rule is unified with it of the goal sentence. A fact is a rule with empty conditions. That is “bird(X).” ←→ “bird(X):- true.”

Each of the “true /0” and the “fail /0” with arity 0 is special predicate which is always succeed or always fail. Negation of predicate occurred in the body is represented by a preposition “¥+” or “not”. A semicolon, “;”, represent disjunctive of conditions, and a “->” denotes If-Then rule. Cut operator, “!”, is a special predicate with arity 0 which protects the backtracking before the location where the “!” has put on but itself is always true.

List notation and recursive definition of predicate

The list notation and the recursive definition of predicates are characteristic property of Prolog programming. A list, L1=[a,b,…], briefly noted as [a|L2], is a finite sequence of terms, which is corresponding to the binary tree with a root node that branched into the left child node “a” and the right remainder, L2=[b,c,…], which is aloso a list the sublist of L1 after “a”.

In background, any list is defined by using the functor “,”. For example, “[a,b,c]=’,’(a,’,’(b,’,’(c,[]))”. The constructor, “=..”, is convenient to transform list and function. Ex., “fly(a) =.. [fly,a]”.

If you use “member /2” which is the incorporated predicate in Prolog

defined as {member(X,[X|_]). member(X,[Y|Z]):-member(X,Z). }. “bird(X):-member(X, [swallow, penguin, eagle]).”, an example where X is a member of the list if it appear the top of it, or the top of the new list after the several iterative removals of the older top elements. This is the very strategy in principle to prove that any complex goal planned to be achieved recursively in Prolog system.

Rational decision making

As insisted in the AI related textbooks about Prolog, probably you can create an intelligent reasoner with Prolog system easily than other procedural computer languages such as Pascal, Basic, or C. Or possibly is the intelligent (bounded) rational artificial agent who makes decisions in the social environment too. As an analogy of AI, that of symbolic problem solving domain, I will call this the “rational (social) choice domain” which to be explicated in the next section.

We demonstrate a simple example of Prolog based implementation of the rational choice domain as follows.

```
alternative(a).
alternative(b).
agent(1).
is_prefer_to(agent(1), state(s1), a, b).
is_prefer_to(agent(1), state(s2), b, a).
rational_choice(Agent,State,Choice):-
        ¥+ is_prefer_to(Agent,State, X, Choice).

< execution example of a goal >
?- rational_choice(1,s1,X).
X = a
Yes
?- rational_choice(1,S,b).
S = s2
Yes
```

Fig.1 A simple example of rational choice domain of single agent.

3. Rational choice domain and social choice correspondence

In this section, we discuss a simple model of the rational choice domain and the social choice correspondence (SCC) via Prolog. We have already developed a simple model of rational agent in the previous section. Here we will elaborate it.

Traditionally in this field, rational choice of such agent is represented by the "preference", $Ri \subseteq A \times A$, a complete, transitive, and reflective binary relation (i.e., the weak order) defined over pairs of the possible outcomes A={a, b, …}. If deterministic, any outcome is same as the alternatives (i.e., the acts) selected by the agent j. For example a R_j b represents the fact that the alternative "a" is preferred to the alternative "b" by the agent j. Indifference relation, denoted by a Ij b, is defined as a Rj b and b Rj a. We write its strict part as Pj. It is defined as a Pj b and not a Ij b.

Prolog based modeling for each of Rj, Ij, and Pj is displayed in Fig.2, Fig.3, and Fig. 4 respectively. A predicate "prefer_profiles" in Fig.5 model Rs, a collection of all possible preference profiles, and an "environment" models where the choice environment variables are collected in.

We can extend the rational choice domain to the social choice environment easily. The code of Prolog is displayed in Fig. 6. Generally, social choice rule (SCR) is mapping from the states (or the preference profiles of all agents) to the outcomes (i.e., alternatives). Social choice function is single-valued SCR. Social choice correspondence (SCC) is SCR which maps a subset of the outcomes. We can regard an SCC as the abbreviated version of the social welfare function (SCF) in the sense of Arrow. The latter can be regarded as an aggregated preference of the representative virtual agent (like the liberal "leviathan"), and the SCC can be regarded as the optimal solutions with respect to the SWF.

```
/*  samples of the preference domain  */
% the King Solomom example from Wolinsky and Rubinstein(1994)
preference(1,s1,[a,b,d]).
preference(2,s1,[b,d,a]).
preference(1,s2,[a,d,b]).
preference(2,s2,[b,a,d]).
difference(_,_,[s,s,end]).
```

```
% the example 2 of Moore-Repullo(1990) with the "bad outcome" z.
preference(1,S11,[a,c,d,b,z]):-member(S11,[s1,s3]).
preference(2,S21,[b,c,d,a,z]):-member(S21,[s1,s4]).
preference(1,S12,[a,d,c,b,z]):-member(S12,[s2,s4]).
preference(2,S22,[b,d,c,a,z]):-member(S22,[s2,s3]).
difference(_,_,[s,s,s,e,end]).
```

Fig.2 Prolog based modeling of preference domain.

In Fig.2 we display the examples of two preference domains modeled as Prolog programs one of them is of 2 agents and 3 outcomes, and the other is of 2 agent and 5 outcomes. The predicate “preference” is the core of the model in that any other preference-relevant object, even though a “is_prefer_to” is a derivative from it. Since the preference is complete, transitive, and reflective, any ordering can be represented by a (flat) list and its corresponding “difference / 3” predicate which specify which the type of relation between the nth position and its successor is “w” (weak), “s” (strict), or “e” (indifferent).

```
/* states, agents, alternatives(outcomes of mechanism). */
alternatives(As):-
        findall(A,(preference(_,_,R),member(A,R)),B), sort(B,As),!.
all_agents(Is):-findall(J,preference(J,_,_),B),sort(B,Is),!.
agents(Is):-all_agents(Is).
states(Ss):-findall(S,preference(_,S,_),B),sort(B,Ss),!.
alternative(A):-alternatives(As),member(A,As).
agent(I):-agents(N),member(I,N).
state(S):-states(Ss),member(S,Ss).
```

Fig.3 A Prolog based modeling of rational choice domain.

In Fig.3 the basic objects are collected from the preference /3 in DB using “findall / 3” rather than the fact clauses which include the list of the possible objects. Each of the three argumets of “findall” is respectively the term to be collected, the conditions to be unified with clauses in DB, and

the collected list. Other predicates “bagof” and “setoff” do similar task but the exsistence quantifier, “^”, can be used to the variables in the conditions.

```
/*   The model of preference for rational agents      */
is_prefer_to(I,S,X,Y):-
   preference(I,S,Order),
   nth1(J,Order,X),
   nth1(K,Order,Y),
   J =< K.
is_prefer_to(I,S,X,Y):-
   preference(I,S,Order),
   nth1(J,Order,X),
   nth1(K,Order,Y),
   J > K,
   difference(I,S,Diffs),
   forall(
    (nth1(L,Diffs,Diff),
      L >= K, L < J
    ),
    Diff = e).

% indifference relation.
indifferent_to(I,S,X,Y):-
   is_prefer_to(I,S,X,Y),
   is_prefer_to(I,S,Y,X).

% strict ordering.
is_strict_ prefer_to(I,S,X,Y):-
   is_prefer_to(I,S,X,Y),
   preference(I,S,Order),
   nth1(J,Order,X),
   nth1(K,Order,Y),
   J < K,
   difference(I,S,Diffs),
   nth1(L,Diffs,s),
```

```
L >= J, L =< K.
```

Fig.4 Preference relations derived from “preference / 3”.

```
 /* preference profiles. */
prefer_profiles(Is,Ss,Rs):-
    subset_of_agents(Is,_N),
    states(Ss),
    bagof(Rks,
       S^(
         bagof(Rk,
            I^(member(I,Is),preference(I,S,Rk)),
         Rks)
       ),
    Rs).

/* social choice environment for N persons. */
environment([Is,Ss,As],[N,K,L],Rs):-!,
      subset_of_agents(Is,N),
      states(Ss),
      length(Ss, K),
      alternatives(As),
      length(As, L),
      prefer_profiles(Is,Ss,Rs).
```

Fig.5 Preference profiles and social choice environment.

```
/* sample SCC; social choice correspondece F:S-->->A.   */
% ks: the King Solomon's choice function.
scc(ks,s1,[a]).
scc(ks,s2,[b]).

% mr: the SCC from Moore and Repullo which satisfies the condition M2.
scc(mr,s1,[c]).
scc(mr,s2,[d]).
scc(mr,s3,[c]).
```

```
scc(mr,s4,[c]).
```

Fig.6 Example of social choice correspondence.

4. Game-theoretical objects and Nash equilibrium

In this section, we discuss the core parts of an implementation theory and a game theoretic mechanism. Firstly for each of theoretical objects we display it in mathematical formulation, secondly a Prolog based modeling is demonstrated

games and Nash equilibrium

The elements of a "game" (of the standard form under complete information) in the game theory is, N={1,2, …,n}, the set of players (i.e., the agents), Si={si1,si2, …} for each i∈N, the set of strategies (i.e., the messages) of each player, A={a,b,c, …}, the set of game outcomes (i.e., the alternatives), g: S→A, S=×(Si) the outcome function, and Ri for each i∈ N ,the set of preference (or utility function) of each agent defined over the set of outcomes.

Best response of the agent is the strategy by her / him and there is no strategy which brings about a strictly preferred outcome for her / him if other agents do not change. Nash equilibrium is the outcome of the game under the profile of best responses of all agents. For each agent j and an outcome a, we define the lower contour set is L (a, Rj):= {x∈A | a Rj x}, and the attainable set is C(a, Rj):={ x∈A | (∃s) a∈g(s), x∈g(s / sj'), s / sj' ∈S} where g( ) is the outcome function and s /sj represents unilateral mutation of j's strategy. The Nash equilibrium (NE) of the game is defined as b∈A such that C(b, Ri)⊆L(b, Ri) for all i∈N, which represents the best response profile. We can redefine it more explicitly by using the maximal elements of the set X⊆A, M(X, Rj):= {a∈A | a∈X⊆L(a, Rj)}, NE as b∈A such that b∈M(C(b, Ri), Ri) for all i∈N.

```
/*   lower contour set   */
lcc([I,S,R],A,Lcc) :-
      alternative(A),
      agent(I),
      state(S),
      preference(I,S,R),
```

```
        findall(L,
         (
            alternative(L),
            is_prefer_to(I,S,A,L)
         ),
           Lcc0),
        sort(Lcc0,Lcc).

/*    maximal elements    */
maximal(A,X,[I,S,R]):-
        preference(I,S,R),
        subset_of_alt(X,_),X¥=[],
        member(A,X),
        lcc([I,S,R],A,Lcc),
        subset(X,Lcc).
maximal_set(Y,X,[I,S,R]):-
        preference(I,S,R),
        subset_of_alt(X,_),X¥=[],
        findall(A,maximal(A,X,[I,S,R]),Max),
        sort(Max,Y).
```

Fig.7 Lower contour set and the maximal elements.

```
/* best response */
best_response([Scc,E,Is],[GF,S,C,Msg],I,[P1s,Czs,Lcc],Br):-
        E = environment([Is,_Ss,_As],[N,_K,_L],_Rs),
        E,
        scc(Scc),
        agent(I), subset_of_agents(Is,N),
        member(I,Is),
        alternative(C),
        game_forms(GF,Ps),
        mtest(GF, Scc,Msg,Is),
        write('wait..'),
    % the mutation set is included in the lower contuor set
    % and so any unilateral deviation can not improve the Nash outcome.
```

```
    findall((P1,Cz),
          ( mutate(GF,Scc,I,Is,Msg,Mz),
          alternative(Cz),
          member(P1,Ps),
          G =.. [GF,P1,Scc],
          mechanism(G,Mz,[Cz])
          ),
         PCzs1),
    sort(PCzs1,PCzs),
    findall(P1,Cz^(member((P1,Cz),PCzs)),P1s1),
    sort(P1s1,P1s),
    findall(Cz,P1^(member((P1,Cz),PCzs)),Czs1),
    sort(Czs1,Czs),
    lcc([I,S,_],C,Lcc),
    (subset(Czs,Lcc)-> Br = yes; Br = no).
```

Fig.8 Best response.

```
mutate_mechanism(GF,Scc,J,Is,[P,P1],[C,C1],[M,M1]):-
   G=..[GF,P,Scc],
   alternative(C),
   mechanism(G,M,[C]),
   member(J,Is),
   mutate(GF,Scc,J,Is,M,M1),
   G1=..[GF,P1,Scc],
   alternative(C1),
   mechanism(G1,M1,[C1]).
check_br_of_profile(Is,E,[GF,Scc,S,C,P,Msg],Mode):-
    forall(
       member(I,Is),
       (
         BrRecord = [Br,S,C,I,GF,P,Scc,Is,Msg,P1s,Czs,Lcc],
         (br_result(BrRecord)->true;
         (
           best_response([Scc,E,Is],[GF,S,C,Msg],I,[P1s,Czs,Lcc],Br),
           (¥+Mode=full -> true ;assert(br_result(BrRecord)))
```

```
          )
          )
         )
      ).
nash(Mode,C,S,Msg,G,H,E,Is,Result):-
      E = environment([Is,_,_],[_N,_,_],_),
      E,
      G =.. [GF,P,Scc],
      reversions(V),      member(H,V),
      scc(Scc),state(S),
      alternative(C),
      mtest(GF,Scc, Msg,Is),
      mechanism(G,Msg,[C]),
      nl,wn(' Now checking whether the message profile. '),
      write_message(S,C,P,Msg,Scc),
      check_br_of_profile(Is,E,[GF,Scc,S,C,P,Msg],Mode),
      collect_br_members([S,C,GF,Scc,Is,Msg],BrMembers0),
      sort(BrMembers0,BrMembers),
      display_br_results([S,C,GF,Scc,Is,Msg],BrMembers,Mode),
      write_nash_equilibrium(BrMembers,Is),
      ( BrMembers = Is -> Result=yes ; Result=no).
```

Fig.8 Nash equilibrium.

5. Nash implementation: theory and simulation

In this section, we demonstrate both of a theoretical model and a simulation result of Prolog based simulation of Nash implementation.

Nash implementabillty----known results

The notion of Nash implementation is explained as follow. If there exists a game theoretical mechanism such that if the outcome of the autonomous maximization of agents brings about a Nash equilibrium, it always matches the SCC outcomes––– that is no out-of-SCC outcome to be a Nash equilibrium in the state, and for every SCC outcome there is a Nash equilibrium which yields the outcome in the state.

In the literature, the necessary and sufficient condition for Nash implementation has explicated. We will soon display several conditions

and theorems without proof.

If for some mechanism for every state of the world (or preference profiles) the set of Nash equilibrium (i.e., Nash correspondence) is always the same as the SCC, we say the SCC is implementable in Nash equilibrium, or is "Nash-implementable" for short. Since the game forms has separated from the true preferences, even if the planner does not know the true preferences of the agents, the goal of the planner above mentioned can be achieved.

Maskin's classical theorem (Maskin,1977/98)

SCC f: Rs→2^A /{}. Rs is a admissible preference profiles.

Definition. (Monotonicity) An SCC f is monotone if for all R, R'∈Rs, a ∈f(R), L(a, Rj)⊆L(a, Rj') for all j→ a∈f(R')

Definition. (No Veto Power) An SCC f satisfies NVP (No Veto Power) if for all R∈Rs, a∈M(A, Rj) for all j≠i→ a∈f(R).

Theorem. (Nash implementability (Maskin,1977/98)) If n>2 or n=2, the SCC f is Nash implementable only if f is monotone. And it is also sufficient if f has NVP and n>2.

Hurwicz-Schmeidler impossibility with two-agent unrestricted domain

Theorem. (cf., Maskin(1998), Moore and Repullo(1990)) If n=2, Rs=Ps is a strict part of unrestricted domain, the SCC f: Rs→2^A /{} is Pareto efficient, then f is Nash implementable iff f is dictatorial.

Peleg impossibility as for Sen's minimal liberalism condition

Definition. (Unanimity) An SCC f satifies unanimity if for all R∈Rs, a ∈M(A, Rj) for all j→ a∈f(R).

Definition. (Desisiveness) Let j∈N, x, y∈A, f : Rs→2^A /{} is an SCC. An agent j is decisive for x over y, if for all R∈Rs, and for all B⊆A, x Pj y, x∈B, and not x∈f(R) → not y∈f(R).

Definition. (Sen's Minimal liberalism(ML)) There exist {k,j}⊆N, (x,y), (z,w)∈A×A, x≠z, y≠w, such that k is decisive for x over y, and j is decisive for z over w.

Theorem. (Peleg,1998, p.76) If n=2, Rs is a unrestricted domain, the SCC f: Rs→2^A /{} satisfies unanimity and condition ML, then f is not Nash implementable.

Danilov's theorem for unrestricted domain (Danilov,1992)

SCC f: Rs→2^A. Rs is a universal domain (i.e., unrestricted domain) that consists of all possible preference profiles or, Ps, the strict part of it.

Definition. (Essential elements) For SCC f, $j \in N$, $a \in X \subseteq A$ is essential for j in X if $a \in f(R)$ for some $R \in Rs$, such that $L(a, Rj) \subseteq X$. The set of essential elements is denoted by Ess(f,j,X).

Definition. (Essential monotonicity) An SCC f is essentially monotone if for all R, $R' \in Rs$, $a \in f(R)$, $Ess(f,j,L(a, Rj)) \subseteq L(a, Rj')$ for all $j \rightarrow a \in f(R')$

Definition. (Blocking relation) B(f,j,X), i.e., for An SCC f, an agent j blocks a set $X \subseteq A$ via f if there is a $R \in Rs$ such that $X \cap f(R'/Rj) = \{\}$ for all $R' \in Rs$.

Definition. (Individual rationality) An SCC f is individually rational if for all $R \in Rs$, $a \in f(R) \rightarrow$ There is no $j \in N$ who blocks L(a, Rj).

Proposition. (A part of proposition 3 of Danilov and so on) If f is an essentially monotone SCC, $B(f,j,X) \leftrightarrow Ess(f,j,X)$ is not empty. And f satisfies NVP $\leftrightarrow$ every agent blocks only an empty set via f $\rightarrow$ f is individually rational.

Definition. (MR-property) $N=\{1,2\}$. For an SCC f has the MR-property if for any unblocked pair (X1,X2) such that $X1, X2 \subseteq A$, not B(f,1,X1), and not B(f,2,X2), there is $a \in A$, an element of the MR-elements which is defined as $MR(f,X1,X2) := \{b \in X1 \cap X2 \mid X1 = L(b,R1), X2 = L(b,R2)\}$.

Theorem. (Nash implementability in unrestricted domain and an empty outcome permitted (Danilov,1992)) If n>2, the SCC f: $Rs \rightarrow 2^A$ is Nash implementable iff f is essentially monotone. If n=2, f is Nash implementable iff in addition individually rational and f has MR-property.

Yamato's approximatedly universal domain (Yamato(1992). See also Danilov(1992) footnote 2.)

Definition. (Condition D) Rs satisfies condition D if there is no triple (a,R,j), $a \in A$, $R \in Rs$, $j \in N$, such that $X = L(a,Rj)$, $\{b \in A \mid (\exists R' \in Rs) L(b,Rj') = X, L(b,Rk') = A \text{ for all } k \neq j.\} = \{\}$.

Theorem. (Theorem 1 and 2 of Yamato(1992)) If admissible preferences Rs satisfies condition D, an SCC f: $Rs \rightarrow 2^A / \{\}$ is Nash implementable only if f is essentially monotone. And if n>2, it is also sufficient.

Moore-Repullo and Dutta-Sen's theorem (Moore-Repullo,1990)

Definition. (Condition μ) Rs is an admissible preference profiles. There exsit $B \subseteq A$ and $C(a, Rj) \subseteq A$ for all $R \in Rs$, $a \in f(R)$, $j \in N$, such that for all R, $R' \in Rs$,

$a \in M(C(a, Rj), Rj')$ for all $j \rightarrow a \in f(R')$, …(i)

$c \in M(C(a, Rj), Rj')$ and $c \in M(B, Rk')$ for all $k \neq j \rightarrow c \in f(R')$, …(ii)

$d \in M(B, Rk')$ for all $k \in N \rightarrow d \in f(R')$. …(iii)

Definition. (Condition $\mu 2$) $N=\{i,j\}$. In addition to condition μ, for all $R, R' \in Rs$ such that $a \in f(R)$ and $b \in f(R')$, there exists $e \in C(a, Ri) \cap C(b, Rj)$, such that for all $R'' \in Rs$,

$e \in M(C(a, Ri), Ri'') \cap M(C(b, Rj'), Rj'') \rightarrow e \in f(R'')$. …(iv)

Theorem. (Full characterization (Moore-Repullo,1990)) If $n>2$ ($/n=2$), an SCC $f: Rs \rightarrow 2^A /\{\}$ is Nash implementable iff condition μ ($/\mu 2$) is satisfied.

Corollary. (Moore-Repullo's corollary 3) If $n=2$, an SCC $f: Rs \rightarrow 2^A /\{\}$ is monotone (i), has restricted veto power (ii), and there is a bad outcome, then f is Nash implementable.

Sjöström's iterative elimination algorithm (Sjöström,1991)

Proposition. (B^* and $C^*(a, Rj)$) $B^* := \cup \{B \subseteq A \mid B$ satisfies (iii) of condition $\mu.\}$ can be iteratively computed by $B[1] := A$; $B[k+1] := B[k] \cap \{a \in B[k] \mid (\forall j) B[k] \subseteq L(a, Rj) \rightarrow a \in f(R).\}$ which converges to B^* in finite steps. And similarly for any $j \in N$, $Rj \in Rs[j]$, and $a \in A$, $C^*(a,Rj) := \cup \{Cj \subseteq L(a, Rj) \cap B^* \mid B=B^* \rightarrow$ condition (ii) is satisfied.$\}$ can be computed by $C[1](a,Rj) := L(a, Rj) \cap B^*$; $C[k+1](a,Rj) := C[k](a,Rj) \cap \{b \in C[k](a,Rj) \mid (\forall j)\ L(a,Rj) \subseteq L(b,R'j), B^* \subseteq L(b,R'k)$ for all $k \neq j \rightarrow b \in f(R').\}$ which converges to $C^*(a,Rj)$ in finite steps.

Lemma. (reformulations of μ and $\mu 2$ (Sjöström,1991)) If for any $j \in N$, $Rj \in Rs[j]$, and $a \in f(R'/Rj)$ for some $R' \in Rs$, $a \in C^*(a, Rj)$, condition μ holds. $\leftrightarrow$ condition (i) holds when $C(a, Rj)=C^*(a, Rj)$, and condition $\mu 2$ holds. $\leftrightarrow$condition (i) and (iv) hold when $C(a, Rj)=C^*(a, Rj)$.

mechanisms under complete information

In the context of Nash implementation theory, a game is separated as a pair of game forms and the agent's true profile of their preferences. That is, there are games as much as possible true preference profiles. In other words, a game g() is parameterized by the possible preference profiles, $g(s,R) \in A$. Additionally some solution concept, especially Nash equilibrium is needed to predict the result of the game play.

A mechanism consists of the game forms (without agents' preference profiles) and the message space where the each message has defined the outcome when it applied to the game form. As a player of the game, each agent sends the messages to the planner. If the messages actually sent by

all agents, then the outcome of the game is determined by the mechanism only by the match of the pattern with one of the game forms, without explicitly knows the planner about the true preference profile of the members of the society.

With regard to the sufficiency part of the proof of Nash implementation theorem in the literature, the authors made the mechanisms for the implementable SCCs, not the existence proof. Table 1 summarizes such conditions and the mechanisms developed by researchers in this field. We exhibit Prolog modeling these conditions and mechanisms in appendix B and appendix C instead of their counterparts in the mathematical formulation. Available mechanisms via Prolog are

--- gM Maskin-Vind's cononical mechanism for n>2.
--- gD Danilov's mechanism for n=2.
--- gMR Moore-Repullo's mechanism for n>2.
--- gMR2 Moore-Repullo's mechanism augmented by Sjöström's algorithm for n=2.

We postpone the discussion of Prolog simulation of these mechanisms to next subsection.

Almost commonly in those mechanisms (in Table 1) a game form is applied in accordance with the degree of (dis)agreement in the message profile sent by the agents. That is (1) perfect agreement, (2) perfect agreement with only one unilateral deviator, and (3) otherwise. We can easily to see that augmented by an integer game (or a modulo n roulette), an objection flag (a bit from {0,1}), the classical mechanism of Maskin has been extended in the literature, and elaborated into the full characterization. Saijo(1988) explains historical steps of introducing the former augmentation and discusses the reduction of message space size.

Author (Year)
Solution Concept
#Agents
Contents of Message
Domain
Type of Proof
Class of Scc
Maskin (1977/1999)

Nash equilibrium
n > 2
a preference profile, an Scc outcome, and a integer (mod n)
restricted
sufficient part monotonicity, NVP

Saijo (1988)
Nash equilibrium
n > 2
a preferences of self and a neigbour, an outcome,
and a integer (mod n)
restricted
sufficient part monotonicity, NVP

Dutta and Sen (1991)
Nash equilibrium
n = 2
a preferences profile, the outcome, a bit, and a integer
restricted
necessary and sufficient
condition â:
condiiton ì of Moore andRepullo
and selef-selection with maximal elements

Moore and Repullo (1990)
Nash equilibrium
n = 2
a preferences profile, the Scc outcome, an outcome, and a integer
restricted
necessary and sufficient
condition ì 2 : almost same as condiiton â of Dutta and Sen.

Moore and Repullo (1990)
Nash equilibrium
n > 2
a preferences profile, the Scc outcome, an outcome, and a integer
restricted
necessary and sufficient
condition ì:
a stronger version of monotonicity, RVP, unanimity with open set

Danilov (1992)

Nash equilibrium

n = 2

a profile of subset[*1] of alternatives, the Scc outcome,
and an integer.

unrestricted

necessary and sufficient

essential-monotonicity, individual rationality, MR-property [*2]

(*1, *2: In Danilov(1992), an empty set is permitted for *1, and scc value may be empty set therefore inconsistent mechanism which may yield no outcome implement such an SCC.)

Danilov (1992)

Nash equilibrium

n > 2

a preference profile, the Scc outcome, and a integer.

unrestricted

necessary and sufficient

essential-monotonicity

Yamato (1992)

Nash equilibrium

n > 2

a preferences profile, the Scc outcome, a bit, and a integer (mod n)

semi-restricted necessary and sufficient

essential-monotonicity

Dutta and Sen (1991b)

Strong Nash equilibrium

n > 1

a preferences profile, the Scc outcome, a bit, and a integer

restricted

necessary and sufficient

condition ã:

Suh (1996)

Strong Nash equilibrium

n > 1

a preferences profile, the Scc outcome, a bit, and a integer (mod n)

restricted

necessary and sufficient
a version of condition ãd

Table.1 The cases of mechanism design for Nash implementation

Simulation via Prolog

In rather directly, by checking Nash equilibrium over the possible pairs of states and outcomes we can check Nash implementability. We demonstrate two SCCs with restricted domain appeared in the literature (the results displayed in Fig. 9 and Fig.10), and a dictatorial and a non constant implementable SCCs with a small unrestricted domain (in Fig.11 and Fig.12) which are found from the auto-checking on the domain ud22. As for auto-generation and testing on ud22, it is explained in next section.

It may be a help for reader who has an intimate knowledge of mathematical logics that we can think Nash implementation as an analogy of the soundness and completeness of deductive systems. Indeed it is relatively easy to verify when we simulate our models since we are ensured the truthful messages yield a Nash outcome (i.e., revelation principle). But the counterpart of it is rather hard to establish the alibi by simulation.

Testing Nash implementability via prolog simulation

**** the model specification ******

game form: gM
members of society: [1,2,3]

Scc:
g1(s1) = [x,w] g1(s2) = [x,y]
is Maskin-monotone.
is Essentially-monotone.

preferene domain:
[agent,state,preference,difference]
[1,s1,[x,z,y,w],[s,s,s,end]]

[1,s2,[x,z,y,w],[s,s,s,end]]
[2,s1,[y,z,x,w],[s,s,s,end]]
[2,s2,[y,z,x,w],[s,s,s,end]]
[3,s1,[z,x,w,y],[s,s,s,end]]
[3,s2,[z,x,y,w],[s,s,s,end]]

other tests for this model
[mm,em,no,no,no,nvp,rvp,unan,no,no,no,neli,mlib]
checking the on-SCC patterns:
[s1,w][s1,x][s2,x][s2,y]

For state s1,outcome=w [in,g1],rule=1
message profile is
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], w, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], w, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], w, 0

agent=1,P1s=[2],Czs=[w],Lcc=[w] <is_best_response>
agent=2,P1s=[2],Czs=[w],Lcc=[w] <is_best_response>
agent=3,P1s=[2],Czs=[w,y],Lcc=[w,y] <is_best_response>
Br_Members=[1,2,3]

This action profile is a Nash equilibrium.

For state s1,outcome=x [in,g1],rule=1
message profile is
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0

agent=1,P1s=[2],Czs=[w,x,y],Lcc=[w,x,y,z] <is_best_response>
agent=2,P1s=[2],Czs=[w,x],Lcc=[w,x] <is_best_response>
agent=3,P1s=[2],Czs=[w,x,y],Lcc=[w,x,y] <is_best_response>
Br_Members=[1,2,3]

This action profile is a Nash equilibrium.

For state s2,outcome=x [in,g1],rule=1
message profile is
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0

agent=1,P1s=[2],Czs=[w,x,y],Lcc=[w,x,y,z] <is_best_response>
agent=2,P1s=[2],Czs=[w,x],Lcc=[w,x] <is_best_response>
agent=3,P1s=[2],Czs=[w,x,y],Lcc=[w,x,y] <is_best_response>
Br_Members=[1,2,3]

This action profile is a Nash equilibrium.

For state s2,outcome=y [in,g1],rule=1
message profile is
[[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0
[[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0
[[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0

agent=1,P1s=[2],Czs=[w,y],Lcc=[w,y] <is_best_response>
agent=2,P1s=[2],Czs=[w,x],Lcc=[w,x,y,z] <is_best_response>
agent=3,P1s=[2],Czs=[w,y],Lcc=[w,y] <is_best_response>
Br_Members=[1,2,3]

This action profile is a Nash equilibrium.

checking the off-SCC patterns:
[s1,y][s1,z][s2,w][s2,z]

For state s1,outcome=y [out,g1],rule=2
message profile is
[[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
[[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0

```
 agent=1,P1s=[1,2],Czs=[w,x],Lcc=[w,y] <is_not_best_response>
 agent=2,P1s=[2,3],Czs=[y],Lcc=[w,x,y,z] <is_best_response>
 agent=3,P1s=[2,3],Czs=[x,y],Lcc=[y] <is_not_best_response>
Br_Members=[2]

←---  omitted  ---→

For state s2,outcome=w  [out,g1],rule=2
message profile is
      [[x,z,y,w],[y,z,x,w],[z,x,w,y]], w, 0
      [[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0
      [[x,z,y,w],[y,z,x,w],[z,x,y,w]], y, 0

 agent=1,P1s=[1,2],Czs=[y],Lcc=[w] <is_not_best_response>
 agent=2,P1s=[2,3],Czs=[w,y],Lcc=[w] <is_not_best_response>
 agent=3,P1s=[2,3],Czs=[w],Lcc=[w] <is_best_response>
Br_Members=[3]


Summary Results
[s1,w,yes,in]
[s1,x,yes,in]
[s1,y,no,out]
[s1,z,no,out]
[s2,w,no,out]
[s2,x,yes,in]
[s2,y,yes,in]
[s2,z,no,out]

Statistics
cputime= [16.250004,increased from,118.250002]
inferences= [5905757,increased from,2128947]
```

Fig.9 A Nash implementation via canonical mechanism gM for “jp”, a domain model of the voting example by Jackson and Palfray (2001).

Testing Nash implementability via prolog simulation

**** the model specification ******

game form: gMR2
members of society: [1,2]

Scc:
md(s1) = [a] md(s2) = [b] md(s3) = [c]
is Maskin-monotone.
is not Essentially-monotone.
does not have Moore-Repullo property.
is not individually-rational.

preferene domain:
[agent,state,preference,difference]
[1,s1,[a,d,c,b],[s,s,s,end]]
[1,s2,[d,a,b,c],[s,s,s,end]]
[1,s3,[c,a,d,b],[s,s,s,end]]
[2,s1,[b,a,d,c],[s,s,s,end]]
[2,s2,[b,a,d,c],[s,s,s,end]]
[2,s3,[b,c,a,d],[s,s,s,end]]

other tests for this model
[mm,no,no,no,no,no,rvp,unan,po,no,no,no,mlib]
checking the on-SCC patterns:
[s1,a][s2,b][s3,c]

For state s1,outcome=a [in,md],rule=1
message profile is
[[a,d,c,b],[b,a,d,c]], a, a, 0
[[a,d,c,b],[b,a,d,c]], a, a, 0

agent=1,P1s=[1,2,3],Czs=[a,b,c,d],Lcc=[a,b,c,d] <is_best_response>
agent=2,P1s=[1,2,4],Czs=[a,c,d],Lcc=[a,c,d] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s2,outcome=b [in,md],rule=1
message profile is
[[d,a,b,c],[b,a,d,c]], b, a, 0
[[d,a,b,c],[b,a,d,c]], b, a, 0

agent=1,P1s=[1,2,3],Czs=[b,c],Lcc=[b,c] <is_best_response>
agent=2,P1s=[1,2,4],Czs=[a,b,c,d],Lcc=[a,b,c,d] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s3,outcome=c [in,md],rule=1
message profile is
[[c,a,d,b],[b,c,a,d]], c, a, 0
[[c,a,d,b],[b,c,a,d]], c, a, 0

agent=1,P1s=[1,2,3],Czs=[a,b,c,d],Lcc=[a,b,c,d] <is_best_response>
agent=2,P1s=[1,2,4],Czs=[a,c,d],Lcc=[a,c,d] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

checking the off-SCC patterns:
[s1,b][s1,c][s1,d][s2,a][s2,c][s2,d][s3,a][s3,b][s3,d]

For state s1,outcome=b [out,md],rule=1
message profile is
[[d,a,b,c],[b,a,d,c]], b, a, 0
[[d,a,b,c],[b,a,d,c]], b, a, 0

agent=1,P1s=[1,2,3],Czs=[b,c],Lcc=[b] <is_not_best_response>
agent=2,P1s=[1,2,4],Czs=[a,b,c,d],Lcc=[a,b,c,d] <is_best_response>
Br_Members=[2]

←--- omitted ---→

For state s3,outcome=b [out,md],rule=1
message profile is
[[d,a,b,c],[b,a,d,c]], b, d, 0
[[d,a,b,c],[b,a,d,c]], b, d, 0

agent=1,P1s=[1,2,3],Czs=[b,c],Lcc=[b] <is_not_best_response>
agent=2,P1s=[1,2,4],Czs=[a,b,c,d],Lcc=[a,b,c,d] <is_best_response>
Br_Members=[2]

Summary Results
[s1,a,yes,in]
[s1,b,no,out]
[s1,c,no,out]
[s1,d,no,out]
[s2,a,no,out]
[s2,b,yes,in]
[s2,c,no,out]
[s2,d,no,out]
[s3,a,no,out]
[s3,b,no,out]
[s3,c,yes,in]
[s3,d,no,out]

Statistics
cputime= [287.000039,increased from,15776.005000]
inferences= [57671637,increased from,502268136.000000]

Fig.10 A Nash implementation via Moore-Repullo mechanism gMR2 for "md", a domain model of the example by Moore (1993).

Testing Nash implementability via prolog simulation

**** the model specification ******

game form: gD
members of society: [1,2]

Scc:
udd1(s1) = [x] udd1(s2) = [x] udd1(s3) = [y] udd1(s4) = [y]
is Maskin-monotone.
is Essentially-monotone.
has Moore-Repullo property (defined by Danilov).
is individually-rational.

preferene domain:
[agent,state,preference,difference]
[1,s1,[x,y],[s,end]]
[1,s2,[x,y],[s,end]]
[1,s3,[y,x],[s,end]]
[1,s4,[y,x],[s,end]]
[2,s1,[x,y],[s,end]]
[2,s3,[x,y],[s,end]]
[2,s2,[y,x],[s,end]]
[2,s4,[y,x],[s,end]]

other tests for this model
[mm,em,mr,ir,no,no,rvp,unan,po,no,dict,no,no]
checking the on-SCC patterns:
[s1,x][s2,x][s3,y][s4,y]

For state s1,outcome=x [in,udd1],rule=1
message profile is
[x], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[x],Lcc=[x,y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s2,outcome=x [in,udd1],rule=1
message profile is
[x], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[x],Lcc=[x] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s3,outcome=y [in,udd1],rule=1
message profile is
[y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s4,outcome=y [in,udd1],rule=1
message profile is
[y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[x,y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

checking the off-SCC patterns:

[s1,y][s2,y][s3,x][s4,x]

For state s1,outcome=y [out,udd1],rule=1
message profile is
[y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[y] <is_not_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
Br_Members=[2]

For state s1,outcome=y [out,udd1],rule=1
message profile is
[y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[y] <is_not_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
Br_Members=[2]

←---- omitted ----→

For state s4,outcome=x [out,udd1],rule=3
message profile is
[], 1
[x], 1

agent=1,P1s=[2,3],Czs=[x,y],Lcc=[x] <is_not_best_response>
agent=2,P1s=[2,3],Czs=[x,y],Lcc=[x] <is_not_best_response>
Br_Members=[]

Summary Results
[s1,x,yes,in]
[s1,y,no,out]
[s2,x,yes,in]
[s2,y,no,out]

[s3,x,no,out]
[s3,y,yes,in]
[s4,x,no,out]
[s4,y,yes,in]

Statistics
cputime= [31.937509,increased from,13376.003438]
inferences= [4718599,increased from,220725287.000000]

Fig.11 A Nash implementation via Danilov mechanism gD for the dictatorial SCC for agent 1 on ud22.

Testing Nash implementability via prolog simulation

**** the model specification ******

game form: gD
members of society: [1,2]

Scc:
urd2(s1) = [x,y] urd2(s2) = [y] urd2(s3) = [y] urd2(s4) = [y]
is Maskin-monotone.
is Essentially-monotone.
has Moore-Repullo property (defined by Danilov).
is individually-rational.

preferene domain:
[agent,state,preference,difference]
[1,s1,[x,y],[s,end]]
[1,s2,[x,y],[s,end]]
[1,s3,[y,x],[s,end]]
[1,s4,[y,x],[s,end]]
[2,s1,[x,y],[s,end]]
[2,s3,[x,y],[s,end]]
[2,s2,[y,x],[s,end]]
[2,s4,[y,x],[s,end]]

other tests for this model
[mm,em,mr,ir,no,no,rvp,unan,no,no,no,no,no]
checking the on-SCC patterns:
[s1,x][s1,y][s2,y][s3,y][s4,y]

For state s1,outcome=x [in,urd2],rule=1
message profile is
[x,y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s1,outcome=y [in,urd2],rule=1
message profile is
[y], 0
[y], 0

agent=1,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s2,outcome=y [in,urd2],rule=1
message profile is
[y], 0
[y], 0

agent=1,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
agent=2,P1s=[1,2],Czs=[x,y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s3,outcome=y [in,urd2],rule=1
message profile is
[y], 0
[y], 0

agent=1,P1s=[1,2],Czs=[y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

For state s4,outcome=y [in,urd2],rule=1
message profile is
[y], 0
[y], 0

agent=1,P1s=[1,2],Czs=[y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[y],Lcc=[x,y] <is_best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

checking the off-SCC patterns:
[s2,x][s3,x][s4,x]

For state s2,outcome=x [out,urd2],rule=1
message profile is
[x,y], 0
[x,y], 0

agent=1,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <is_best_response>
agent=2,P1s=[1,2],Czs=[x,y],Lcc=[x] <is_not_best_response>
Br_Members=[1]

←---- omitted ----→

For state s4,outcome=x [out,urd2],rule=3
message profile is
[x], 1
[x], 1

agent=1,P1s=[2,3],Czs=[x,y],Lcc=[x] <is_not_best_response>
agent=2,P1s=[2,3],Czs=[x,y],Lcc=[x] <is_not_best_response>
Br_Members=[]

Summary Results
[s1,x,yes,in]
[s1,y,yes,in]
[s2,x,no,out]
[s2,y,yes,in]
[s3,x,no,out]
[s3,y,yes,in]
[s4,x,no,out]
[s4,y,yes,in]

Statistics
cputime= [0.000624,increased from,15264.005625]
inferences= [3145886,increased from,361758799.000000]

Fig.12 A Nash implementation via Danilov mechanism gD for a non constant, non dictatorial SCC for agent 1 on ud22.

6. Checking implementability on a small unrestricted domain

In this section, we demonstrate a screening method for the implemetable SCCs on given domain. Utilizing the backtracking incorporated in Prolog, we can generating possible SCCs recursively and automatically verifies the conditions of Nash-implementability (in Fig.13) under the given domain model. Since this program for auto-generating SCCs in Fig.13 is iterative mutation based on already setuped domain

model, we can use it for any model which have made to test at least neglecting the computational load argued bellow.

In figures 14—16, we display summarry results in which the data from auto-generating simulation of all possible SCCs on the “ud22”, a model of the strict part of unrestricted domain with 2 agents, 2 outcomes. We can verify the implementation theorem by Danilov, whether essential monotonicity, individual rationality, and MR-property are satisfied simultaneosly (i.e., ess+ir+mr=1) in Fig.14. And we can observe a Hurwicz-Schmeidler impossiblity result when focused on Pareto efficient (par=1) non-empty (emptiness=0) SCCs in Fig.15, and a Peleg impossibility result when focused on minimally liberal (mlib=1) and unaminity (unan=1) efficient non-empty SCCs in Fig.16.

The computational load of this simulation can be evaluated as the size of SCC space enumerated. Since in this domain the number of possible ordering (and states) is 2 (and 2^2), and any SCC at any state is a subset of alternatives (which permit the empty set when we use the Danilov's mechanism), the number of possible SCCs in ud22 is

(2^#alternatives)^(#orderings^#agents)=(2^2)^(2!^2)=256.

(Of course, precisely this should be multiplied by the complexity of computing the given set of conditions for each SCC.) But if we try “ud23”, the computational complexity of this approximation to be

(2^3)^(3!^2)=325*10^39.

Fortunately, we know the theorems for unrestricted domain for N=2 and N>=3 (Nash implementation theorems (Danilov, 1992) and famous impossibility result for N=2, for example, see Maskin(1999), Peleg's lemma as for Sen's minimal liberalism (1998), etc.). Therefore we can use this knowledge in order to find the implementable SCCs on this domain as well as to debug the routine.

As stated in the previous section, we found the 13 Nash-implementable SCCs by checking whether the SCC satisfies em + ir + mr, i.e., essentially-monotone, individually rational, and Moore-Repullo property respectively in the sense of Danilov(1992).

The 7 SCCs of them are all consistent, that is each of them has no empty output in any state, including the constant (2) and the dictatorial (2). And the 6 of them are inconsistent rules which has no output in some state or no output at all.

```
 tests_for_scc(F,Is,[G1,G2,G3,G4,BO,G5,G6,G7,G8,G9,G10,G11,G12]):-
   scc(F),   subset_of_agents(Is,_),
   (monotone(F,Is) -> G1 = mm; G1 = no),
   (ess_monotone(F,Is) -> G2 = em; G2 = no),
   (has_MR_property(F,Is) -> G3 = mr; G3 = no),
   (test_irat_n2(F,Is) -> G4 = ir; G4 = no),
   ((bad_outcome(Bad,F,Is)) ->BO = bad(Bad); BO = no),
   (nvp(F,Is) -> G5 = nvp; G5 = no),
   (rvp(F,Is) -> G6 = rvp; G6 = no),
   (unanimity(F,Is) -> G7 = unan; G7 = no),
   (has_pareto_property(F,Is) -> G8 = po; G8 = no),
   (has_condorcet_property(F,Is) -> G9 = co; G9 = no),
   (dictatorial(F,Is) -> G10 = dict; G10 = no),
   (neli(F,Is) -> G11 = neli; G11 = no),
   (mlib(F,Is) -> G12 = mlib; G12 = no).
gen_test_sccs(SC_tuples,Is,Goals):-
   generate_scc(scc1,SC_tuples), tests_for_scc(scc1,Is,Goals).
generate_scc(F,Cc):-
   member(F,[scc0,scc1,scc2]),
   states(Ss), length(Ss,K),
   scc_tuple(Cc,K,Ss),
   clear_scc(F),   % note: this position is sensitive.
   update_scc(F,Cc).
scc_tuple([],0,[]).
scc_tuple([[S,X]|Cr],K,[S|Sc]):-
   scc_tuple(Cr,K1,Sc),
   K is K1 + 1,
   states(Ss),reverse(Ss,Sr),nth1(K,Sr,S),
   subset_of_alt(X,_N1).
   %X ¥= [].   %←---Otherwise than using Danilov's mechanism gD.
update_scc(scc1,Cc):-
   forall(member([S,X],Cc),assert(scc(scc1,S,X))).
clear_scc(scc1):-
   forall(scc(scc1,S,W),retract(scc(scc1,S,W))).
```

Fig.13 Auto generate and test program for possible SCCs given domain.

emptyness	scc(s1)	scc(s2)	scc(s3)	scc(s4)	satisfied conditions	scc_alias
4	[]	[]	[]	[]	[mm,em,mr,ir,no,no,no,no,po,co,no,neli,mlib]	umd0
2	[x]	[]	[]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no,no,mlib]	umd1
1	[x]	[y]	[]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no,no,mlib]	umd2
1	[x]	[x]	[]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no,no,mlib]	umd3
1	[x]	[]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no,no,mlib]	umd4
0	[y]	[y]	[y]	[y]	[mm,em,mr,ir,no,no,no,no,no,no,no,no,no]	urd1
0	[x,y]	[y]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,no,no,no,no,no]	urd2
0	[x]	[x]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,dict,no,no]	udd1
1	[x]	[]	[x]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no,no,mlib]	umd5
0	[x]	[y]	[x]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,dict,no,no]	udd2
0	[x]	[x]	[x]	[x]	[mm,em,mr,ir,no,no,no,no,no,no,no,no,no]	urd3
0	[x]	[x]	[x]	[x,y]	[mm,em,mr,ir,no,no,rvp,unan,no,no,no,no,no]	urd4

Fig.14 A generated SCCs which satisfy em+ir+mr on ud22 domain.

empty	scc(s1)	scc(s2)	scc(s3)	scc(s4)	mm	em	mr	ir	bad	nvp	rvp	unan	po	co	dict	neli	mlib	mr+ir+em	neutral
0	[x]	[y]	[y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0
0	[x]	[x]	[y]	[y]	1	1	1	1	0	0	1	1	1	0	1	0	0	1	0
0	[x]	[x,y]	[y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x]	[y]	[x]	[y]	1	1	1	1	0	0	1	1	1	0	1	0	0	1	0
0	[x]	[x]	[x]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0
0	[x]	[x,y]	[x]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x]	[y]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x]	[x]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x]	[x,y]	[x,y]	[y]	1	1	0	1	0	1	1	1	1	0	0	0	1	0	0

Fig.15 A Hurwicz-Schmeidler impossibility for Pareto correspondence.

empty	scc(s1)	scc(s2)	scc(s3)	scc(s4)	mm	em	mr	ir	bad	nvp	rvp	unan	po	co	dict	neli	mlib	mr+ir+em	neutral
0	[x]	[x,y]	[y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x,y]	[x,y]	[y]	[y]	1	1	0	1	0	0	1	1	0	0	0	0	1	0	1
0	[x]	[x,y]	[x]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x,y]	[x,y]	[x]	[y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1
0	[x]	[y]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x,y]	[y]	[x,y]	[y]	1	1	0	1	0	0	1	1	0	0	0	0	1	0	1
0	[x]	[x]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0
0	[x,y]	[x]	[x,y]	[y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1
0	[x]	[x,y]	[x,y]	[y]	1	1	0	1	0	1	1	1	1	0	0	0	1	0	0
0	[x,y]	[x,y]	[x,y]	[y]	1	1	0	1	0	1	1	1	0	0	0	0	1	0	1
0	[x]	[x,y]	[y]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x,y]	[x,y]	[y]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x]	[x,y]	[x]	[x,y]	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x,y]	[x,y]	[x]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x]	[y]	[x,y]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x,y]	[y]	[x,y]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x]	[x]	[x,y]	[x,y]	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x,y]	[x]	[x,y]	[x,y]	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0
0	[x]	[x,y]	[x,y]	[x,y]	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0
0	[x,y]	[x,y]	[x,y]	[x,y]	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0

Fig.16 A Peleg impossibility for Sen's minimal liberalism.

7. Some implications for the “bounded rationality”

Finally, we briefly discuss on the relation to the notion of “bounded rationality”. It seems me that the line of research adopted in the paper has some implicit implications as for the notion introduced by H.A.Simon. We summarize the three points as follows.

Firstly despite of its normative vein, using computational modeling and simulation naturally lead to the descriptive theories as for the behavior of the bounded-rational agents who are granted the limited computational resources.

This is the notion of “bounded rationality” which is repeatedly cited in the literature as that of developed by H.A. Simon who has put its importance on the computational aspect of the cognitive limitation of the agent who is intended to act as rational idealistic agent but cannot do it. And this aspect of the notion may be easily modeled within our paradigm by the agent's preference modified as the rule with the body that consists of the conditions for search to find the true ordering of the alternatives.

However, simultaneously he inherited from the research of decision making in the real human organization (i.e., administrative behavior in the firm).

That is, secondly beside the first notion of “boundedness” of rationality, in real human behavior is not in the course that is stipulated by the theory of rational choice, not only in the descriptive sense (i.e., which is caused by the unintended systematic errors of “cognitive-behavioral” system of human decision maker.), but also in the “intentional” levels. It may be rather the problem of misspecification of the problem itself, in the “design activity” (in the sense of Simon), which is the search for the appropriate description of the problem and the candidates of solution for the problem with “satisficing” principle, also the term H.A.Simon has introduced in opposition to the “optimization”.

What the hidden incentive-relevant variables behind the description is the problem that Nash implementation theory has attempted to solve in abstract model, but it seems me that this has not been appropriately treated by the theory. Also as it is the issue that Simon has deliberatively avoided when the prescriptive theory has applied in real context of society, despite of that the learning process is the very problem he wanted to

resolve. I thought that the misspecification of the problem is the key factor because of the designer's incomplete knowledge, which cause the bounded rationality, as well as which is constrained by it.

At most case, the misspecification of the problem stems from also the cognitive limitation, i.e., members do not know true state. But it may be caused by the fraud by members in the society, and it causes the emotional affect related to the "value" of the clients so that she / he does not want to do truth-telling, to do obey the rational solution, or even to do say so. Not only the true value of the incentive relevant variables, the true variables and the other parameters that consists the objective function for the organization and for her/him-self are hidden at least partly, and it prevent people away from obedience to the stipulations by the normative theory of rational choice, or even conversely it make them act in obedience at least on the surface.

Thirdly, we can regard the pair of the preference ordering and the social choice rule as a social cognition, or a distributed knowledge representation system of the agents. The lower contour set (lcc), more correctly the essential elements and the blocking relation introduced by Danilov, for the agent consists the "focal sets" for the agent, which has its (decentralized or centralized) analogy in the partition —or nest — based information structure, a model of the knowledge and the common knowledge of game players — or the belief function, an ambiguity-relevant information processing model in decision science.

8. Conclusion

In this paper we have discussed a Prolog based computer simulation approach for Nash implementation theory. I found that the Prolog based intelligent agent system provides powerful toolbox to pursue such as modeling the agents' rational preferences, checking or generating the best response message profiles and the Nash equilibrium under any state, verifying, designing or learning the game-theoretical mechanisms and various conditions under which various social choice rules can be achieved via Nash equilibria.

However our demonstration was restricted only in very small domain with a few agents and several outcomes. In order to ensure usefulness in larger domain, the computational complexity reduction technique should

be developed, especially for the proof of the non-existence of Nash equilibrium. The auto generation and verification for the “SCC”s under fixed preference domain models with respect to the theoretical conditions of implementability is rather efficient than the direct simulation of Nash implementability. However it is not so hard to find a Nash equilibrium when the pair of state and outcome being in the member of SCC because of the “revelation principle” which ensures the existence of a Nash equilibrium under the truth-telling strategy.

On the other hand, the auto covariate-enumaration of SCCs with preference domain itself (and possibly both of them) satisfying selected desirable conditions is strong tool for mechanisms designer in principle even though computationally inefficient. Beside an enhancement of computational power, it might be desirable to use logic programming technique on the syntax level of proofs, instead of the direct simulation of the preference-relevant objects, but it is open task up to now.

If hurdle these, we want to hack the way to advance mechanisms for incomplete information, sequential mechanisms, and mechanisms with more elaborated solution concepts such as strong / undominated / virtual / repeated implementation, all of which may be investigated further. Beside the above mentioned, a mixed strategy equilibrium or an imperfect information games require probabilistic treatment (i.e., risk simulation), which is open task up to now. However, there is a straightforward extension of our model into the risk simulation by multiplication of clauses which approximate the probability distribution.

Appendix A. Tools for list operation and subset enumeration

```
/* nth1 /3 of SWI-prolog. if you need to write, use this. */
nth1(A, B, C) :- integer(A), !, D is A - 1, nth0_1(D, B, C).
nth1(A, B, C) :- var(A), nth0_2(D, B, C), A is D + 1.
nth0_1(0, [A|B], A) :- !.
nth0_1(A, [B|C], D) :- E is A - 1, nth0(E, C, D).
nth0_2(0, [A|B], A).
nth0_2(A, [B|C], D) :- nth0_2(E, C, D), succ(E, A).
/* projection operator. */
list_projection([],[],[]).
```

```
list_projection([X|Y],[_A|B],C):-
    X = 0,
    list_projection(Y,B,C).
list_projection([X|Y],[A|B],[A|C]):-
    X = 1,
    list_projection(Y,B,C).
/* subset enumeration is possible */
subset_of(A,N,As):-
    length(As,L),
    length(D,L),
    list_projection(D,As,B),
    length(B,N),
    sort(B,A).
/* natural number sequence less than N. */
desc_nnseq([],N):-N<0,!.
desc_nnseq([0],1).
desc_nnseq([A|Q],N):-
    A is N - 1,
    length(Q,A),
    desc_nnseq(Q,A).
asc_nnseq(Aseq,N):-desc_nnseq(Dseq,N),sort(Dseq,Aseq).
/* bag0/3 : allow multiplicity */
bag0([],_A,0).
bag0([C|B],A,N):-length([C|B],N),bag0(B,A,_N1),%N is N1 + 1,
  member(C,A).
/* bag1/3 : do not allow multiplicity */
bag1([],_A,0).
bag1([C|B],A,N):-length([C|B],N),bag1(B,A,_N1),%N is N1 + 1,
  member(C,A),¥+member(C,B).
/* ordering /3 */
ordering(A,B,C):-bag1(A,B,C).
```

Appendix B. Conditions

```
% ---------------------------------------------------------  %
%   Condition D   : checking the domain
%   whether it is approximately universal.
```

```
% ---------------------------------------------------------  %
% reference: Yamato(1992), p.487.
condition_D(Is,Rs):-
    alternatives(As),
    subset_of_agents(Is,_),
    prefer_profiles(Is,Ss,Rs),
    forall(
     (
       nth1(K,Ss,S),       % K: the index for the state
       nth1(K,Rs,R),
       nth1(L,Is,J),       % J: the index for the agent
       nth1(L,R,RJ),
       alternative(A),
       lcc([J,S,RJ],A,Lcc),
       member(B,Lcc),wn([S,J,A,B])
     ),
       (
         lcc([J,S1,_RJ1],B,Lcc),wn([J,S1,B,Lcc]),
         forall((nth1(_L1,Is,J1),J1¥=J),
            (
              wn([J,S1,B,Lcc]),lcc([J1,S1,_RJ2],B,As)
            )
         )
       )
    ).
%%%%%%%%%%%%%%%%%%%%%%%%%%
% conditions for sccs
%%%%%%%%%%%%%%%%%%%%%%%%%%
% ---------------------------------------------------------  %
%       Maskin monotonicity: checking via lcc.
% ---------------------------------------------------------  %
monotone(F,Is):-
   scc(F),
   subset_of_agents(Is,_N),
   % there is a reversal everywhere an outcome retracted from scc.
   forall(
```

```
        (
          state(S),
          scc(F,S,C),
          member(A,C),
          state(S1),% S1 ¥= S,
          scc(F,S1,C1),
          ¥+ member(A,C1),%write([A,member_of,C,in,S,out_of,C1,in,S1]),
          true),
        (
          member(I,Is),
          lcc([I,S,_R],A,K1),
          lcc([I,S1,_R1],A,K2),%wn([lcc_of,I,K1,K2]),
          ¥+ subtract(K1,K2,[]),%wn(ok),
          true)
     ).
% ------------------------------------------------------------  %
%   Strong Monotonicity (Essential Monotonicity)
% ------------------------------------------------------------  %
% reference: Danilov(1992)
ess_monotone(F,Is):-
   scc(F),
   subset_of_agents(Is,_N),
   % for each dropped scc element, there is a reversal in the ess_outcomes.
   forall(
        (
          state(S),
          scc(F,S,C),
          member(A,C),
          state(S1),% S1 ¥= S,
          scc(F,S1,C1),
          ¥+ member(A,C1)
        ),
        (
          member(I,Is),
          lcc([I,S,_],A,Lcc1),    % [1]
          ess(F,I,Lcc1,Ess),       % [2]
```

```
        lcc([I,S1,_],A,Lcc2),
        ¥+ subtract(Ess,Lcc2,[])
      )
    ).
/* Ess: the essential set */
ess(F,I,X,Ess):-
    agent(I),scc(F),
    subset_of_alt(X,_),
    findall(A,
      S^R^(alternative(A),is_essential(A,X,[I,S,R],F)),
      Y),
    sort(Y,Ess).
is_essential(A,X,[I,S,R],F):-
     subset_of_alt(X,_),
     agent(I),
     state(S),
     preference(I,S,R),
     scc(F,S,C),
     member(A,C),
     lcc([I,S,R],A,Lcc),
     subset(Lcc,X).
% ------------------------------------------------------   %
% blocking and individual rationality
% ------------------------------------------------------   %
% reference: Danilov(1992).
is_blocked_by(X,J,[S,RJ],Is,F):-
     alternative(X),
     subset_of_agents(Is,_N),
     nth1(K,Is,J),
     state(S),
     preference(J,S,RJ),
     scc(F,S,_SccVal0),
     forall(
       (
         prefer_profile(Is,S1,R1),
         nth1(K,R1,RJ)
```

```
        ),
        (
          scc(F,S1,SccVal),
          ¥+member(X,SccVal)
        )
      ).
is_blocked_by(X,J,Is,F):-
      scc(F),
      subset_of_agents(Is,_),
      member(J,Is),
      preference(J,S,R),
      max_blocking(Bs,[J,S,R],Is,F),
      subset_of_alt(X,_),
      subset(X,Bs).
/*   unblocked pair   */
unblocked_pair([X1,X2],[J1,J2],F):-
      unblocked(X1,J1,[J1,J2],F),
      unblocked(X2,J2,[J1,J2],F).
unblocked0(X,I,Is,F):-
      scc(F),
      subset_of_agents(Is,_N),member(I,Is),
      subset_of_alt(X,_M),X¥=[],
      forall(member(A,X),
         ¥+is_blocked_by(A,I,[_S,_R],Is,F)).
unblocked(X,I,Is,F):-
      scc(F),
      subset_of_agents(Is,_N),member(I,Is),
      subset_of_alt(X,_M),X¥=[],
      ¥+ is_blocked_by(X,I,Is,F).
/* subadditivity of blocking   */
test_subadditivity([I,J],F,Result):-
      unblocked_pair([X,Y],[I,J],F),
      intersection(X,Y,W),
      Result=[unblocked,[X,Y],intersect,W].
test_sa(S,F):-results_to_file([test_subadditivity,[1,2],F,_R],'sa.txt',S).
% ------------------------------------------------------   %
```

```
% individual rationality
% (individually rational outcomes)
% ---------------------------------------------------------   %
% reference: Danilov(1992).
i_rationals(As,S,Rn,Is,F):-
      scc(F),
      subset_of_agents(Is,_),
      prefer_profile(Is,S,Rn),
      findall(A,
       (
         alternative(A),
         forall(member(J,Is),
          ( nth1(K,Is,J),
             nth1(K,Rn,RJ),
             is_I_rational(A,[J,S,RJ],Is,F)))
       ),
      As1),sort(As1,As).
is_I_rational(A,[J,S,RJ],Is,F):-
      subset_of_agents(Is,_N),
      scc(F),
      member(J,Is),
      preference(J,S,RJ),
      alternative(A),
      lcc([J,S,RJ],A,Lcc),
      ¥+is_blocked_by(Lcc,J,Is,F).
/* a test for IR */
test_irat_n2(F,[I,J]):-
   scc(F),
   two_person([I,J]),
   forall(scc_defined_state(S,F), (
        test_irat_n2(F,[I,J],[S,_V,_A,yes])
    )).
test_irat_n2(F,[I,J],[S,V,A,B]):-
      scc(F,S,V),
      i_rationals(A,S,_R,[I,J],F),
      subset_query(V,A,B),
```

```
      wn([state,S,scc,V,ir_set,A,'ir?',B]).
subset_query(V,A,B):-
      subset(V,A) -> B = yes; B = no.
% ---------------------------------------------------------  %
% Moore-Repullo property for unrestricted domain with two agents.
% ---------------------------------------------------------  %
% reference: Danilov(1992).
is_MR_element(A,[X1,X2],[J1,J2],[[R1,R2],S],F):-
      alternative(A),
      scc(F,S,C),
      member(A,C),
      prefer_profile([J1,J2],S,[R1,R2]),
      lcc([J1,S,R1],A,L1),
      lcc([J2,S,R2],A,L2),L1=X1,L2=X2.
is_MR_elements(MR,[X1,X2],[J1,J2],F):-
      scc(F),
      two_person([J1,J2]),
      subset_of_alt(X1,_N1),
      subset_of_alt(X2,_N2),
      findall(A,
        (
          scc_defined_state(S,F),
          prefer_profile([J1,J2],S,R),
          is_MR_element(A,[X1,X2],[J1,J2],[R,S],F)
        ),
      MR0),sort(MR0,MR).
   % it may have sorted by setof/3 but fail under MR=[].
has_MR_property(F,[J1,J2]):-
      scc(F),
      two_person([J1,J2]),
      forall(
        unblocked_pair([X1,X2],[J1,J2],F),
          % --note---The fact of existence of an unblocked pair can be
          %   regarded as the analog of the 'self-selection constraint'
          %   in the term of Dutta-Sen(1991). ------- %
       (
```

```
        is_MR_elements(MR,[X1,X2],[J1,J2],F),MR¥=[],
        wn(['MR-elements:',MR,'for unblocked pair:',[X1,X2]])
      )
    ).
/*    Other important correspondences      */
     %%    Pareto, Condorcet, Dictatorial, NVP, RVP. %%
     %%    : with unanimity, minimal liberalism, etc.
% ------------------------------------------------------   %
% the (weak) Pareto optimal correspondence
% ------------------------------------------------------   %
has_pareto_property(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    forall(scc(F,S,C),(wpo(Cpo,Is,S),subset(C,Cpo))).
% weak pareto optimal correspondence.
wpo(Cpo,Is,S):-
    subset_of_agents(Is,_),
    state(S),
    findall(A,is_wpo(A,S,Is),Cpo0),
    sort(Cpo0,Cpo).
% alternative A is efficient for agent J in state S.
is_wpo(A,S,Is):-
    is_weak_parato_optimal(A,S,Is).
is_weak_parato_optimal(A,S,Is):-
    subset_of_agents(Is,_),
    %preference_profile(Is,S,_P),
    state(S),
    alternative(A),
    forall(alternative(B),is_prefer_to(_J,S,A,B)).
% ------------------------------------------------------   %
% the majority rule for strict preferences
% ------------------------------------------------------   %
has_condorcet_property(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    forall(scc(F,S,C),(condorcet(Con,Is,S),subset(C,Con))).
```

```
condorcet(Con,Is,S):-
    subset_of_agents(Is,_),
    state(S),
    findall(A,(
      alternative(A),
      forall(alternative(B),
      pairwise_majority(A,B,_Na,_Nb,S,Is))
    ),Con0),
   sort(Con0,Con).
pairwise_majority(A,B,Na,Nb,S,Is):-
    subset_of_agents(Is,_),
    state(S),
    alternative(A),
    alternative(B),
    bagof(J,(member(J,Is),is_prefer_to(J,S,A,B)),Jas),
    bagof(J,(member(J,Is),is_prefer_to(J,S,B,A)),Jbs),
    length(Jas,Na),
    length(Jbs,Nb),
    Na >= Nb.
% ------------------------------------------------------   %
% dictator, dictatorial scc
% ------------------------------------------------------   %
dictatorial(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    member(J,Is),
    is_a_dictator(J,F,Is).
is_a_dictator(J,F,Is):-
    subset_of_agents(Is,_),
    agent(J),member(J,Is),
    scc(F),
    alternatives(Alts),
    forall((scc(F,S,C),member(A,C)),
       lcc([J,S,_R],A,Alts)
    ),
    forall(lcc([J,S,_R],A,Alts),
```

```
        (scc(F,S,C),member(A,C))
    ).
% ---------------------------------------------------------- %
% NVP (no veto power) and RVP (restricted veto power)
% ---------------------------------------------------------- %
% note: rvp/2, the ristricted version of nvp in the sense
%   that there is no_veto_power unless   the outcome
%   is strictly worse than any outcome in range(scc).
% ---------------------------------------------------------- %
nvp(F,Is):-
    no_veto_power(F,Is,full).
rvp(F,Is):-
    no_veto_power(F,Is,ristricted).
unanimity(F,Is):-
    subset_of_agents(Is,_),
    scc(F),
    alternatives(As),
    forall(
     (
       alternative(A),
       scc_defined_state(S,F),
       findall(J,(member(J,Is),lcc([J,S,R],A,As)),Is0),
       sort(Is0,Is)
     ),
     (scc(F,S,C), wn([[unanimity,Is0,A],[scc,S,C]]),
       member(A,C),
       forall(lcc([J,S,R],A,L),
       wn([ok,[preference,J,S,R],[lcc,A,L]])))
    ).
% veto outcome under scc
veto_outcome(A,J,S,Is,F):-
    subset_of_agents(Is,_),
    scc(F),
    agent(J),
    state(S),
    scc(F,S,C),
```

```
    alternative(A),
    ¥+member(A,C),
    alternatives(As),
    forall((agent(K),K¥=J),
      lcc([K,S,_Rk],A,As)
    ).
% nvp by using lcc:
% (equivalence has confirmed as for no_veto_power0/
% using veto_outcome, or no_veto_power1/ using weak).
no_veto_power(F,Is,full):-
    subset_of_agents(Is,_),
    scc(F),
    alternatives(As),
    forall(
     (
      alternative(A),
      scc_defined_state(S,F),
      agent(I),member(I,Is),
      subtract(Is,[I],Is1),
      findall(J,lcc([J,S,R],A,As),Is0),
      subset(Is1,Is0)
     ),
     (scc(F,S,C),
      member(A,C),   wn([[nvp,Is0,A],[scc,S,C]]),
      forall(lcc([J,S,R],A,L), wn([ok,[preference,J,S,R],[lcc,A,L]]))
     )
    ).
no_veto_power(F,Is,ristricted):-
    subset_of_agents(Is,_),
    scc(F),
    range(F,Bs),
    forall((agent(J),member(J,Is),
          alternative(A),scc_defined_state(S,F)),
      (
        ¥+veto_outcome(A,J,S,Is,F);
        (
```

```
          member(B,Bs),
          ¥+is_prefer_to(J,S,A,B)
        )
      )
    ).
% another code for NVP which is equivalent to nvp
no_veto_power0(F,Is,full):-
    subset_of_agents(Is,_),
    scc(F),
    forall((agent(J),member(J,Is),
          alternative(A),scc_defined_state(S,F)),
      (
       ¥+veto_outcome(A,J,S,Is,F)
      )
    ).
% NVP <--> no agent has veto outcome <--> all agent are weak.
% an agent is "weak" if it has no veto outcome.
% note: in this case ess(F,J,X,X) always hold.
no_veto_power1(F,Is,full):-
    subset_of_agents(Is,_),
    scc(F),
    forall((agent(J),member(J,Is)),weak(J,Is,F)).
weak(J,Is,F):-
    subset_of_agents(Is,_),
    scc(F),
    agent(J),
    ¥+veto_outcome(_A,J,_S,Is,F).
% agent J is weak <--> J blocks only [].
weak_b(J,Is,F):-
    subset_of_agents(Is,_),
    scc(F),
    agent(J),
    ¥+ (alternative(A),
    is_blocked_by(A,J,[_S,_R],Is,F)).
%lemma: weak-> lcc(X)=X, nvp->ess(X)=X.
test_lemma_weak(J,Is,F,R):-
```

```
   member(R,[full,restricted]),wn(R),
   forall(
      (
        agent(J),member(J,Is),
        subset_of_alt(X,_),
      true),
      (r_weak(J,Is,F,R)
      ->
        ess(F,J,X,X)
      )
   ).
test_lemma_nvp(F,Is,R):-
   member(R,[full,restricted]),wn(R),
   no_veto_power(F,Is,ristricted),wn(nvp),
   %
   forall(
      (alternative(A),
        agent(J),member(J,Is),
        scc_defined_state(S,F)
      ),
      (set_C_star(X,_,[[J,S,R],A,_Bstar],Is,F)
      ->
        lcc([J,S,R],A,X)
      )
   ).
% ------------------------------------------------------   %
%   bad_outcome, nonempty_lower_intersection
% ------------------------------------------------------   %
% reference: Moore and Repulo(1990), p.1090.
bad_outcome(Z,F,Is):-
     scc(F),
     subset_of_agents(Is,_),
     alternative(Z),
     range(F,B),
     forall(
        (
```

```
        member(X,B),
        member(J,Is),
        state(S)
      ),
    is_strict_prefer_to(J,S,X,Z)
  ).
/* no_empty_lower_intersection (neli) for n=2. */
neli(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    forall(
      neli(F,Is,[_,_,_,SL12]),
      SL12 ¥= []
    ).
neli(F,[J1,J2],[[R1,R2],[X,Y],[SL1,SL2],SL12]):-
    scc(F),
    subset_of_agents([J1,J2],_),
    scc(F,S1,C1),
    scc(F,S2,C2),
    member(X,C1),
    member(Y,C2),
    slcc([J1,S1,R1],X,SL1),
    slcc([J2,S2,R2],Y,SL2),
    intersection(SL1,SL2,SL12).
% ---------------------------------------------------------  %
% decisiveness and minimal liberalism(ML).
% ---------------------------------------------------------  %
% reference: Peleg(1998), p.76.
is_decisive(J,for(X),over(Y),F):-
    scc(F),
    agent(J),
    alternative(X),
    alternative(Y),Y¥=X,
    forall(
      ( preference(J,S,R),
        subset_of_alt(_B,_) ),
```

```
        ( forall(
            ( slcc([J,S,R],X,SL),
              member(Y,SL),
              scc(F,S,C),
              ¥+member(X,C)
            ),
            ¥+member(Y,C))
        )
    ).
/*   Sen's weakest condition of liberalism (ML). */
mlib(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    subset_of([J1,J2],2,Is),
  % member(J1,Is),member(J2,Is),J2¥=J1,
    decisive_pairs([J1,_X,_Y],[J2,_Z,_W],F).
decisive_pairs([J1,X,Y],[J2,Z,W],F):-
    agent(J1),
    agent(J2),
    alternative(X),
    alternative(Y),
    alternative(Z), Z¥=X,
    alternative(W), W¥=Y,
    is_decisive(J1,for(X),over(Y),F),
    is_decisive(J2,for(Z),over(W),F).
% ---------------------------------------------------------   %
%   neutrality of scc wrt permutation of the alternatives
% ---------------------------------------------------------   %
%   scc(permutated_preference)=permutation(scc).
/*   permutation of outcomes (citation from earlier part)   */
poa(P,APs):-
    alternatives(A),
    permutation_of(A,P,APs).
% <-----   omitted ------>
permutate_of_order(PoA,[Q->R]):-
    poa(_P,PoA),
```

```
    alternatives(A),
    length(A,M),
    ordering(Q,A,M),%wn(Q),
    ordering(R,A,M),%wn(R),
    permutation(Q,PoA,R).
 % <-----   omitted ------>
is_neutral(F,Is):-
    scc(F),
    subset_of_agents(Is,_),
    forall( % -----forall (1)---------
      (
       poa(_,PoA),
       permutated_prefer_profile(Is,PoA,Ps->Rs)
      ),
      (
       forall( % -----forall (2)---------
          (
           state(S),prefer_profile(Is,S,Ps),
           scc(F,S,C),member(C0,C)
          ),
          (
           state(S1),prefer_profile(Is,S1,Rs),
           scc(F,S1,C1),
           member(C0->X,PoA),
           member(X,C1)
          )
       )% ----end forall (1)------
      )% -----end forall (2)---------
    ).
permutated_prefer_profile(Is,PoA,Ps->Rs):-
    subset_of_agents(Is,_),
    alternatives(A),
    state(S),
    prefer_profile(Is,S,Ps),
    ordering(P,A,_M),
    poa(P,PoA),!,%wn(PoA),
```

```
    bagof(R,
       K^J^Q^(
          nth1(K,Is,J),
          nth1(K,Ps,Q),
          permutate_of_order(PoA,[Q->R])
       ),
      Rs).
/* conditions and algorithms for the Moore-Repullo mechanism */
/*
% ---------------------------------------------------------   %
%     Condition  μ: i,ii,iii    μ2: i～iv                    %
% ---------------------------------------------------------   %
*/
% reference: Moore-Repullo(1990) , Dutta-Sen(1991) (for n=2).
% Sjostrom(1991), pp.334-5.
%
condition_mju(F,Is,Bx,[GoBx,GoCx],Mju):-
    subset_of(Mju,_,[i,ii,iii,iv]),     wn([bx,Bx,mju,Mju]),
    scc(F),
    subset_of_agents(Is,_),
    (GoBx=yes -> set_B_star(Bx,_,Is,F);true),
    subset_of_alt(Bx,_),    % <-- note! it should be done after set_B_star/4.
    forall(
       (
         agent(J), member(J,Is), state(S),
         preference(J,S,R),
         alternative(A), scc(F,S,V),member(A,V)
         ,tab(1),wn([[agent_state_pref,J,S,R],A])
       ),
       (
         (GoCx=yes -> set_C_star(Cx,_,[[J,S,R],A,Bx],Is,F);true),
         subset_of_alt(Cx,_), member(A,Cx),
         (
           lcc([J,S,R],A,Lcc),
           subset(Cx,Lcc),
           subset(Cx,Bx)
```

```
        ),
        tab(2),wn([cx_A,Cx,A,lcc,Lcc,bx,Bx]),
        % sub-conditions of mju and mju2 in Moore-Repullo(1990)
        forall(
          (member(M,Mju),M¥=iv),
          (
            sub_condition_of_mju(M,F,Is,[J,A,Bx,Cx])
            ,tab(3),wn([ok_mju,M])
          )
        ),
        (member(iv,Mju)->
          (sub_condition_of_mju(iv,F,Is,[J,A,Bx,GoCx])
          ,tab(3),wn([ok_mju,iv])
          );true
        )
      )
    ).
condition_mju2(F,[J1,J2],Bx,[GoBx,GoCx]):-
    condition_mju(F,[J1,J2],Bx,[GoBx,GoCx],[i,ii,iii,iv]).
/*
% ---------------------------------------------------------- %
%    subsidiary conditions for μ and μ2     %
% ---------------------------------------------------------- %
*/
sub_condition_of_mju(i,F,Is,[J,A,_Bx,Cx]):-
% this corresponds to Moores-Repullo's condition, mju (i), monotonicity.
    scc(F),
    subset_of_agents(Is,_),
    agent(J),member(J,Is),
    alternative(A),
%    subset_of_alt(Bx,_),
    subset_of_alt(Cx,_),
    forall(
      (
        scc_defined_state(S,F),
        prefer_profile(Is,S,Rs),
```

```
        tab(4),wn([profile,Rs,S]),
        forall(
            member(J1,Is),
          (
            maximal(A,Cx,[J1,S,_R1])
            ,tab(5),
            wn([A,is_maximal_in_set_Cx,Cx,for_agent,J1,under_state,S])
          )
        )
       ),
       (
        scc(F,S,V), tab(6),write([scc,V]),member(A,V),wn([A,in_scc])
       )
    ).
sub_condition_of_mju(ii,F,Is,[J,_A,Bx,Cx]):-
% this corresponds to Moores-Repullo's condition, mju (ii), rvp.
    scc(F),
    subset_of_agents(Is,_),
    agent(J),member(J,Is),
    subset_of_alt(Bx,_),
    subset_of_alt(Cx,_),
    forall(
     (
      scc_defined_state(S,F),
      prefer_profile(Is,S,Rs),
      tab(4),wn([profile,Rs,S]),
     %----
      member(C0,Cx),
      preference(J,S,R),
      lcc([J,S,R],C0,Lcc), subset(Cx,Lcc),
      tab(5),wn([C0,J,S,lcc,Lcc,includesCx]),
     %----
      forall((member(J1,Is), J1 ¥= J),
        (
          preference(J1,S,R1),
          lcc([J1,S,R1],C0,Lcc1),subset(Bx,Lcc1)
```

```
          ,tab(6),wn([C0,J1,S,lcc,Lcc1,includesBx])
        )
      )
     ),
     (
       scc(F,S,V),member(C0,V),tab(6),wn([C0,in_scc])
     )
    ).
sub_condition_of_mju(iii,F,Is,[_J,_A,Bx,_Cx]):-
% this corresponds to Moores-Repullo's condition, mju (iii), unanimity.
    scc(F),
    subset_of_agents(Is,_),
    subset_of_alt(Bx,_),
    forall(
     (
      scc_defined_state(S,F),
      prefer_profile(Is,S,Rs),
      tab(4),wn([profile,Rs,S]),
     %----
      member(C0,Bx),
      forall(
        member(J1,Is),
        (
          preference(J1,S,R1),
          lcc([J1,S,R1],C0,Lcc),
          subset(Bx,Lcc)
        )
      ),
      tab(5),wn([C0,J1,S,lcc,Lcc,includesBx])
     ),
     (
      scc(F,S,V),member(C0,V),tab(6),wn([C0,in_scc])
     )
    ).
sub_condition_of_mju(iv,F,[J1,J2],[_J,_A,Bx,GoCx]):-
% this corresponds to Moores-Repullo's condition, mju (iv),for N=2.
```

```
    forall(
     (
       alternative(A),
       alternative(B),
     %you may not use instead 'subset_of_alt([A,B],2)'.
       preference(J1,S1,R1),
       preference(J2,S2,R2),
       scc(F,S1,V1),member(A,V1),
       scc(F,S2,V2),member(B,V2)
       ,tab(4),wn([for_scc_pair,[J1,S1,A],[J2,S2,B]])
     ),
     (
       % ----- the 'self-selection constraint' in the term of Dutta-Sen(1991).
       make_set_Cx_for([[J1,S1,R1],A],Cx1,Bx,[J1,J2],F,GoCx),
       make_set_Cx_for([[J2,S2,R2],B],Cx2,Bx,[J1,J2],F,GoCx),
       intersection(Cx1,Cx2,MR),
       % -----
       member(Phi,MR),tab(5),wn([mr,Phi,in_C1xC2,MR]),
       forall(
          (
            lcc([J1,Sp,_R1p],Phi,Lp1),
            lcc([J2,Sp,_R2p],Phi,Lp2),
            subset(Cx1,Lp1),
            subset(Cx2,Lp2),tab(6),write([for_lcc_pair,Lp1,Lp2])
          ),
          (scc(F,Sp,Vp),member(Phi,Vp),tab(2),wn(in_scc))
       )
     )
    ).
make_set_Cx_for([[J,S,R],A],Cx,Bx,[J1,J2],F,GoCx):-
       member(J,[J1,J2]),
       (GoCx=yes -> (
            set_C_star(Cx,_,[[J,S,R],A,Bx],[J1,J2],F)
       );true),   %wn([cx,Cx,bx,Bx]),
       subset_of_alt(Cx,_),member(A,Cx),     %wn(A),
       lcc([J,S,R],A,Lcc),       %write([lcc,Lcc]),
```

```
        subset(Cx,Lcc),
        subset(Cx,Bx).
% ------------------------------------------------------------  %
% Two sets B_star and C_star    and
% Condition M, M2 and iterative elimination
% for awkward outcomes : %
% ------------------------------------------------------------  %
% reference: Sjostrom(1991), p.336.
% Maskin and Sjostrom(2002),Suh(1996).
/*
set_B_star(Bstar,Kmin,Is,F):-
    (scc(F)->true;(!,fail)),
    subset_of_agents(Is,_),
    % the first fixed point, CB, of the sequece of set operations
    set_Bk([Kmin|_],Bk-Bk,Is,F),
    Bstar=Bk,!.
set_Bk([1|[]],B0-[],Is,F):-
    (scc(F)->true;(!,fail)),
      subset_of_agents(Is,_),
      alternatives(As),
      B0=As.
set_Bk([K|[K1|H1]],Bk-Bk1,Is,F):-
      set_Bk([K1|H1],Bk1-_,Is,F),
      K is K1 + 1,
      findall(A,
        (
          member(A,Bk1),
          (
            scc_defined_state(S,F),
            scc(F,S,V),
            (
              forall(member(J,Is),maximal(A,Bk1,[J,S,_R]))
              -> member(A,V);true
            )
          )
        ),
```

```
      B0),sort(B0,Bk).
%
set_C_star(Cstar,Kmin,[[J,S,R],A,Bstar],Is,F):-
    (scc(F)->true;(!,fail)),
    scc(F),
    subset_of_agents(Is,_),
     member(J,Is),
     preference(J,S,R),
     scc(F,S,V),
     member(A,V),
    set_Ck([Kmin|_],Ck-Ck,[[J,S,R],A,Bstar],Is,F),
   % subset_of_alt(Bstar,_),    <--- never before set_Ck because of using !.
   % subset_of_alt(Ck,_),    <--- never before set_Ck because of using !.
    Cstar=Ck,!.
set_Ck([1|[]],C1-[],[[J,S,R],A,Bstar],Is,F):-
    (scc(F)->true;(!,fail)),
     subset_of_agents(Is,_),%wn([F,Is]),
     member(J,Is),%wn(J),
     preference(J,S,R),%wn([J,S,R]),
     scc(F,S,V),%wn([F,S,V]),
     member(A,V),
     lcc([J,S,R],A,Lcc),%wn([A,Lcc]),
     set_B_star(Bstar,_,Is,F),%wn(Bstar),
     intersection(Lcc,Bstar,C1).
set_Ck([K|[K1|H1]],Ck-Ck1,[[J,S,R],A,Bstar],Is,F):-
     length([K|[K1|H1]],K),    % <-miso
     set_Ck([K1|H1],Ck1-_,[[J,S,R],A,Bstar],Is,F),
     K is K1 + 1,
     findall(B,
       (
        member(B,Ck1),
        (
         scc_defined_state(S,F),
         scc(F,S,V),
         (
          (maximal(B,Ck1,[J,S,R]),
```

```
            forall((member(J1,Is),J1¥=J),
               maximal(B,Bstar,[J1,S,R])
            )
           )
           -> member(B,V);true
          )
         )
        ),
     C0),sort(C0,Ck)
/*
     %%%   MR outcomes in M-R mechanism   %%%
*/
% this is used in the mechanism gMR bellow.
mre(Y,F,[[X1,X2],[J1,J2],[S1,S2],[R1,R2]],[Cx1,Cx2],T):-
     (T=yes
        ->(tab(1),wn([mre_start,[X1,X2],[J1,S1,R1],[J2,S2,R2],F]))
       ;true
       ),
      preference(J1,S1,R11),
      scc(F,S1,V1),
      member(X1,V1),
     (T=yes->(tab(2),wn([[J1,S1,R11],scc,V1,X1]));true),
      preference(J2,S2,R21),
      scc(F,S2,V2),
      member(X2,V2),
     (T=yes->(tab(2),wn([[J2,S2,R21],scc,V2,X2]),
     tab(4),wn([for_scc_pair,[J1,S1,X1],[J2,S2,X2]]),read(_U));true),
      set_C_star(Cx1,_,[[J1,S1,R1],X1,Bx],[J1,J2],F),
      set_C_star(Cx2,_,[[J2,S2,R2],X2,Bx],[J1,J2],F),
     % make_set_Cx_for([[J1,S1,R1],X1],Cx1,Bx,[J1,J2],F,yes),
     % make_set_Cx_for([[J2,S2,R2],X2],Cx2,Bx,[J1,J2],F,yes),
      intersection(Cx1,Cx2,MR),
     (T=yes->(tab(6),wn([cx1,Cx1,cx2,Cx2,mre,MR]));true),
      alternative(Y),
      member(Y,MR),
     (T=yes->(tab(8),wn([mr_end,Y,'in_Cx1*Cx2']));true).
```

```
/*
% ------------------------------------------------------   %
%%%      Conditions M and   M2     %%%
% ------------------------------------------------------   %
      % see Sjostrom(1991), p.335.
*/
condition_M(F,Is):-
    subset_of_agents(Is,_),
    scc(F),
    forall(
       (
        alternative(A),
        state(S),
        member(J,Is),
        preference(J,S,R),
        scc(F,S,C),member(A,C),    wn([[J,S,R],A])
       ),
       (
        set_C_star(Cx,_,[[J,S,R],A,Bx],Is,F),
        member(A,Cx),
        condition_mju(F,Is,Bx,[yes,yes],[i])
       )
    ).
condition_M2(F,[J1,J2]):-
    subset_of_agents([J1,J2],2),
    scc(F),
    forall(
       (
        alternative(A),
        state(S),
        member(J,[J1,J2]),
        preference(J,S,R),
        scc(F,S,C),member(A,C),    wn([[J,S,R],A])
       ),
       (
        set_C_star(Cx,_,[[J,S,R],A,Bx],[J1,J2],F),wn(Cx),
```

```
        member(A,Cx),
        condition_mju(F,[J1,J2],Bx,[yes,yes],[i,iv])
      )
    ).
/*
% ------------------------------------------------------------  %
     %%%      awkward outcomes     %%%
% ------------------------------------------------------------  %
     % reference: Maskin and Sjostrom(2002).
*/
is_awkward(C,[[J,S,R],A],[S1,C,Lcc1],Is,F):-
     alternatives(As),
     subset_of_agents(Is,_),
     member(J,Is),
     scc_defined_state(S,F),
     alternative(A),
     scc(F,S,V),member(A,V),
     preference(J,S,R),
     lcc([J,S,R],A,Lcc),
     alternative(C),member(C,Lcc),
     scc_defined_state(S1,F),
     scc(F,S1,V1),¥+member(C,V1),
    %
     preference(J,S1,R1),
     lcc([J,S1,R1],C,Lcc1),
     subset(Lcc,Lcc1),
     wn([[scc,V,S],[agent,J,[lcc,A,Lcc]]]),
     write('->'),wn([[scc,V1,S1],[agent,J,[lcc1,C,Lcc1]]]),
     forall(
       (nth1(_K,Is,I),I¥=J),
       (lcc([I,S1,_Rk],C,As))
     ).
% C_star has been excluded all higher order awkward outcomes.
awk(W,[[J,S,R],A],Is,F):-
    alternative(A),
    state(S),
```

```
    agent(J),
    set_C_star(C,_K,[[J,S,R],A,_B],Is,F),
    lcc([J,S,R],A,L),
    (subset(C,L)->wn(ok);wn(ng)),
    subtract(L,C,W).
```

Appendix C. Mechanisms

```
/*    the mechanisms part    */
% ------------------------------------------------------   %
% common part %
% ------------------------------------------------------   %
mechanisms([gM,gD,gMR,gMR2]).
mechanism(G,E,Msg,Z):-
      E =.. [environment,[Is,_Ss,_As],[_N,_K,_L],_Rs],
      G =.. [GF,_P,Scc],
      setup(G,Scc,E),
      mtest(GF, Scc,Msg,Is),
      game_form(G,E,Msg,Z),!.
setup(G,Scc,E):-
      scc(Scc),
      mechanisms(GFs),
      member(GF,GFs),
      game_forms(GF,Ps),
      member(P,Ps),
      G =.. [GF,P,Scc],
      subset_of_agents(Is,N),
      states(Ss),
      alternatives(As),
      E = environment([Is,Ss,As],[N,_K,_L],_Rs),
      E.
% ------------------------------------------------------   %
%   message spaces
% ------------------------------------------------------   %
/*   message space for gM, the Maskin-Vind mechanism.   */
mtest(gM, Scc,[],[],Is):-
      scc(Scc),
```

```
        subset_of_agents(Is,_N),!.
mtest(gM, Scc, [M|Msg],[I|Is],Agents):-
        mtest(gM, Scc,Msg,Is,Agents),
        agent(I), ¥+member(I,Is),
        G=..[gM,_,Scc],
        message(G,Agents,I,M,true).
mtest(gM, Scc,Msg,Is):-mtest(gM, Scc,Msg,Is,Is).
/* the message space for gD, the Danilov mechanism. */
mtest(gD,Scc,Msg,[J1,J2]):-
        scc(Scc),
        two_person([J1,J2]),
        Msg = [(X1,Z1),(X2,Z2)],
        subset_of_alt(X1,_),     member(Z1,[0,1]),
        subset_of_alt(X2,_),     member(Z2,[0,1]).
/* message profile for 2 or N agents in mechanism gMR2, gMR.
% reference: Moore-Repullo(1990).    */
mtest(gMR, F,M,Is):-
        scc(F),
        subset_of_agents(Is,_N),
        mr_profile(Is,F,M).
mtest(gMR2, F,M,Is):-
        scc(F),
        subset_of_agents(Is,2),
        mr2_profile(Is,F,M).
% ----------------------------------------------------------    %
% message profiles in gM
% ----------------------------------------------------------    %
maskin(F,Is,S,J,(R,A,Z)):-
        scc(F),
        subset_of_agents(Is,N),
        agent(J), member(J,Is),
        alternative(A),
        state(S),
        prefer_profile(Is,S,R),%wn([Is,S,R]),
        scc(F,S,V),%wn([scc,Obj]),
        member(A,V),
```

```
      asc_nnseq(Aq,N),%wn([Aq,N]),
      member(Z,Aq).
message(gM(_,F),Is,I,(Rs,A,_Z),Consistency):-
    environment([Is,_Ss,_As],[N,_K,_L],_Rks),
    subset_of_agents(Is,N),
    member(I,Is),
    scc(F),
     (
        Consistency = true
       -> (
             state(S),
             prefer_profile(Is,S,Rs),
             scc(F,S,V),
             member(A,V)
           )
       ;   prefer_profile(Is,Rs)
     ),
    alternative(A).
    % for delayed execution, the integer part to be open yet.
    %,asc_nnseq(Aseq,N),member(Z,Aseq).
messages(G,Is,I,Ms,(Con,Show)):-
    subset_of_agents(Is,_N),
    member(I,Is),
    member(Con,[true,_]),
    member(Show,[yes,_]),
    findall(M,message(G,_,I,M,Con),Ms),
    (Show = yes -> writeMessages(Ms,I);true).
writeMessages(Ms,I):-
      length(Ms,L),
      tab(10),write('There are '),
      write(L),write(' strategies for agent '),
      write(I),write('.'), nl,
      prompt(_,' Display those variables? (y/n)> '),
      read(Y), (Y == 'y' -> true; fail).
mr_msg(Is,F,J,M):-
      M=(R,A,B,_Z),
```

```
      subset_of_agents(Is,_N),
      member(J,Is),
      scc_defined_state(S,F),
    %   <-- if you restrict agent-set, shoud'nt use "state(S)" instead.
      prefer_profile(Is,S,R),
      scc(F,S,V),
      alternative(A),member(A,V),
      alternative(B).
mr_profile(Is,F,Prf):-
      subset_of_agents(Is,_N),
      scc(F),
      my_maplist(mr_msg(Is,F),Is,Prf).
mr2_msg([J1,J2],F,J,M2):-
      M2=(R,A,B,Z,Obj),
      M=(R,A,B,Z),
      mr_msg([J1,J2],F,J,M),
      member(Obj,[0,1]).
mr2_profile([J1,J2],F,Prf):-
      subset_of_agents([J1,J2],2),
      scc(F),
      my_maplist(mr2_msg([J1,J2],F),[J1,J2],Prf).
/*
    %%%%%%%%%%%%%%%%%%%%%%%%
    %%  the  game  forms  %%
    %%%%%%%%%%%%%%%%%%%%%%%%
*/
%   available game forms in this system:
game_forms(gM,[1,2,3]).
game_forms(gD,[1,2,3]).
game_forms(gMR,[1,2,3]).
game_forms(gMR2,[1,2,3,4,5,6]).
% ------------------------------------------------------   %
% dictatorship     (almost same as maximal )
% ------------------------------------------------------   %
dictatorship(J,S,X,D):-
  % J: dictator
```

```
   % S: state
   % X: range of choice
   % D: choiced outcome
    agent(J),
    subset_of_alt(X,_),
    true_state(T),
    (state(T)-> S=T;
       state(S),
       %wn(['warnig: state',S, 'is not the updated state.']),
       true
    ),
    maximal(D,X,[J,S,_RJ]).
% ------------------------------------------------------   %
% the roulette of modulo N
% ------------------------------------------------------   %
roulette(Dictator,[Z1,Z2],[J1,J2]):-
      subset_of_agents([J1,J2],2),
      member(Z1,[0,1]),
      member(Z2,[0,1]),
      SumZ is Z1 +   Z2,
      K is SumZ mod 2 + 1,
      nth1(K,[J1,J2],Dictator).%wn(Dictator),read(Y),
% for n-person
roulette(Dictator,Zs,Is):-
      subset_of_agents(Is,N),
      N >2,
      subset_of_agents(Is,N),
      asc_nnseq(Aseqn,N),
      bag0(Zs,Aseqn,N),
      sum(Zs,SumZ),%wn(SumZ),
      K is SumZ mod N + 1,%wn(K),
      nth1(K,Is,Dictator).
% ------------------------------------------------------   %
% The canonical Mechanism for 3 or N agents.
% (Maskin-Vind mechanism, modified by using Ess):
% ------------------------------------------------------   %
```

```
game_form(gM(1,Scc),E,Msg,[C]):-
      E = environment([Is,_Ss,_As],[N,_K,_L],_Rs),
      length(Msg,N),
   %    Msg = [(R,C,_Z1),(R,C,_Z2),(R,C,_Z3)],
   % extended for N >= 3
      setof((Rx,X,0),Z^member((Rx,X,Z),Msg),[(R,C,0)]),!,
      state(S),
      prefer_profile(Is,S,R),
      scc(Scc,S,Obj),
      alternative(C),
      member(C,Obj).
game_form(gM(2,Scc),E,Msg,[C]):-
         E = environment([Is,_Ss,_As],[N,_K,_L],_Rs),
         length(Msg,N),
   % extended for N >= 3. if N=3,
   %    (Msg = [(R1,C1,Z1),(R2,C2,Z2),(R2,C2,Z3)] -> (J is 1,true);
   %    Msg = [(R2,C2,Z1),(R1,C1,Z2),(R2,C2,Z3)] -> (J is 2,true);
   %    Msg = [(R2,C2,Z1),(R2,C2,Z2),(R1,C1,Z3)] -> (J is 3,true)),
   % the agreed preference, R2J, of the agent J who has the unique,
   % and distinctive opinion. estimated same by other two simultaneously.
         setof((R,X,0),Z^(member((R,X,Z),Msg)),MD),
         length(MD,2),
      % find a single deviator's message (by sure thing reasoning).
         member(MJ,Msg),
         counter(1,MJ,Msg),    % <--MJ has to be specified.
         nth1(K,Msg,MJ),
         nth1(K,Is,J),
      % which message is that sent from the deviator.
         MJ = (R1,C1,_Z1),
         member(MJ1,MD),
         MJ1 = (R1,C1,0), % match with MJ partially (if ignore integer).
         member(MJ2,MD),
         ¥+ MJ2 = MJ1,
         MJ2 = (R2,C2,0),!,
        %
         state(S),
```

```
        prefer_profile(Is,S,R2),
        nth1(K,R2,R2J),
        preference(J,S,R2J),
        lcc([J,S,R2J],C2,Lcc),
        ess(Scc,J,Lcc,Ess),
        (member(C1,Ess)-> C = C1; C = C2),
        range(Scc,Range),member(C,Range).
game_form(gM(3,_Scc),E,Msg,[C]):-
     E = environment([_Is,_Ss,_As],[N,_K,_L],_Rs),
     length(Msg,N),
     setof((R,X,0),Z^member((R,X,Z),Msg),Mx),%wn([m,Mx]),
     length(Mx,Lm),   Lm > 2,
     !,
    % Msg = [(_,C1,Z1),(_,C2,Z2),(_,C3,Z3)],!,
    % SumZ is (Z1 + Z2 + Z3),
     roulette(Jd,_Zs,Is),      % delayed execution
     nth1(K,Is,Jd),
     nth1(K,Msg,(_R,C,_Z)).
%%%% end of rules.
%% the Mechanism part for two-person cases:   %%
% ------------------------------------------------------   %
% Danilov's mechanism  μ(F),
% which assumes unristricted domain.
% ------------------------------------------------------   %
game_form(gD(1,Scc),E,Msg,[C]):-%wn([r1]),
     E = environment([[J1,J2],_Ss,_As],[2,_K,_L],_Rs),
     two_person([J1,J2]),
     mtest(gD, Scc,Msg,[J1,J2]),
     Msg=[(X1,Z1),(X2,Z2)],
     ¥+is_blocked_by(X2,J1,[J1,J2],Scc),
     ¥+is_blocked_by(X1,J2,[J1,J2],Scc),!,
     %intersection(X1,X2,Y1),
     /*   caution: the MR set is of the pair [X2,X1]. */
     is_MR_elements(Y,[X2,X1],[J1,J2],Scc),
     roulette(Jd,[Z1,Z2],[J1,J2]),
     member(C,Y),
```

```
    dictatorship(Jd,_S,Y,C).
game_form(gD(2,Scc),E,Msg,[C]):-%write([r2]),
    E = environment([[J1,J2],_Ss,_As],[2,_K,_L],_Rs),
    E,
    two_person([J1,J2]),
    mtest(gD, Scc,Msg,[J1,J2]),
    Msg=[(X1,_),(X2,_)],%wn([X1,X2,[J1,J2]]),
    (
     (
      is_blocked_by(X2,J1,[J1,J2],Scc),
      ¥+is_blocked_by(X1,J2,[J1,J2],Scc),
      X0=X1, J0=J2
     );
     (
      ¥+is_blocked_by(X2,J1,[J1,J2],Scc),
      is_blocked_by(X1,J2,[J1,J2],Scc),
      X0=X2, J0=J1
     )
    ),!,
    ess(Scc,J0,X0,Ess0),%wn([J0,X0,Ess0]),
    member(C,Ess0),
    dictatorship(J0,_S,Ess0,C).
game_form(gD(3,Scc),E,Msg,[C]):-%wn([r3]),
    E = environment([[J1,J2],_Ss,_As],[2,_K,_L],_Rs),
    two_person([J1,J2]),
    mtest(gD, Scc,Msg,[J1,J2]),
    Msg=[(X1,Z1),(X2,Z2)],
    is_blocked_by(X2,J1,[J1,J2],Scc),
    is_blocked_by(X1,J2,[J1,J2],Scc),
    !,
     %wn(Msg),read(Y),
    range(Scc,Range),
    roulette(Jd,[Z1,Z2],[J1,J2]),
    member(C,Range),
    dictatorship(Jd,_S,Range,C).
%%%% end of rules.
```

```
%% the Mechanism for 2 (or more) person cases    %%
% ---------------------------------------------------------    %
% the mechanisms of Moore-Repullo, Dutta-Sen:
% Augmented by Sjostrom's algorithm
% ---------------------------------------------------------    %
game_form(gMR(1,_Scc),E,Msg,[C]):-
        E = environment([_Is,_Ss,_As],[N,_K,_L],_Rs),
        length(Msg,N),N>2,
        setof((R,X),Y^Z^member((R,X,Y,Z),Msg),[(_,C)]),!.
game_form(gMR(2,Scc),E,Msg,[C]):-
        E = environment([Is,_Ss,_As],[N,_K,_L],_Rs),
        length(Msg,N),N>2,
        setof((R,X),Y^Z^(member((R,X,Y,Z),Msg)),[M1,M2]),
        member((R1,A1),[M1,M2]),
        MJ = (R1,A1,B1,_O1),    % disagreed message.
        counter(1,MJ,Msg),      % a unilateral deviator.
        % the semi-agreed component.
        member((R,A),[M1,M2]),(R1,A1)¥=(R,A),
        bagof((R,A),Y^Z^(member((R,X,Y,Z),Msg)),Agree),
        length(Agree,N1),
        N1 is N - 1,
       !,%
        nth1(K,Msg,MJ),
        nth1(K,R,RJ),
        nth1(K,Is,J),      % the deviator has specified.
        preference(J,S,RJ),    % estimated state from agreed profile.
        set_C_star(Cx1,_,[[J,S,RJ],A,_Bx],Is,Scc),
        (member(B1,Cx1)-> C = B1; C = A).
game_form(gMR(3,_Scc),E,Msg,[C]):-
      E = environment([_Is,_Ss,_As],[N,_K,_L],_Rs),
      length(Msg,N),
      setof((R,X),Y^Z^member((R,X,Y,Z),Msg),Mx),%wn([m,Mx]),
      length(Mx,Lm),    Lm > 2,
      !,
      roulette(Jd,_Zs,Is),       % delayed execution
    % lpom(Msg,[gMR,Scc,Is],Zs),
```

```
      nth1(K,Is,Jd),
      nth1(K,Msg,(_R,_A,C,_Z)).
%%%% end of rules.
% ---------------------------------------------------------   %
%%%% Moore-Repullo mechanism gMR2 for N=2. %%%
% ---------------------------------------------------------   %
game_form(gMR2(1,_Scc),E,Msg,[C]):-
         E = environment([_Is,_Ss,_As],[2,_K,_L],_Rs),
         Msg = [(R,C,_,_,_),(R,C,_,_,_)],!.
game_form(gMR2(P,Scc),E,Msg,[C]):-
         E = environment([[J1,J2],_Ss,_As],[2,_K,_L],_Rs),
         Msg=[MJ1,MJ2],
         MJ1 = (R1,A1, B1,Z1,O1),
         MJ2 = (R2,A2, B2,Z2,O2),
         [R1,A1] ¥= [R2,A2],
        % ----------------------- %
         member(P,[2,3,4,5,6]),
        % ----------------------- %
         ((O1=0,O2=0) -> P=2    ;true),
         ((O1>0,O2=0) -> P=3    ;true),
         ((O1=0,O2>0) -> P=4    ;true),
           Sum is Z1 + Z2,
           Mod is Sum mod 2,
         ((O1>0,O2>0,Mod = 0) -> P=5; true),
         ((O1>0,O2>0,Mod = 1) -> P=6; true),!,
        % ----------------------- %
   % note(1): If you handle P5, as follows, as the case 2, [s2,c] is in scc mr.
      % ((O1=1,O2=1) -> P=2    ;true),!,
   % note(2): the original Moore-Repullo construction uses shouting game.
      % ((O1>=O2,O2>0) -> P=5 ; true),
      % ((O1>0,O2>O1) ->    P=6 ; true), %originally shouting integer.
        alternative(C),
        R1 = [_R11,R12], prefer_profile([J1,J2],S1,R1),
        R2 = [R21,_R22], prefer_profile([J1,J2],S2,R2),
        (
          mre(MR,Scc,[[A2,A1],[J1,J2],[S2,S1],[R21,R12]],[Cx1,Cx2],no)
```

```
        ->true;MR=non
        ),
        ((P=2) -> C = MR ;true),
        ((P=3) -> (member(B1,Cx1)-> C = B1; C=MR);true),
        ((P=4) -> (member(B2,Cx2)-> C = B2; C=MR);true),
        ((P=5) -> C = B1 ; true),
        ((P=6) -> C = B2 ; true).
%%%% end of gMR2 rules.
% consistency check for any game form.
game_form(_,E,Msg,[]):-
        E = environment([Is,_,_],[N,_,_],_),
        length(Is,N1),
        length(Msg,N2),
        ¥+sort([N,N1,N2],[N]),
        wn(this_message_is_inconsistent_and_cannnot_processed),!.
```

Appendix D. Getting start with our system---A sample execution.

```
Welcome to SWI-Prolog (Version 1.9.0 June 1994)
Copyright (c) 1993,1994 University of Amsterdam.    All rights reserved.

1 ?- [impl09].

 %*******************************************************%
 %*       Nash implementation theory on SWI-prolog.       *
 %*                    version impl09                     *
 %*******************************************************%
impl09 compiled, 0.11 sec, 155,472 bytes.

Yes

2 ?- setup_domain(jp).

preference domain:
[agent,state,preference,difference]
    [1,s1,[x,z,y,w],[s,s,s,end]]
```

[1,s2,[x,z,y,w],[s,s,s,end]]
[2,s1,[y,z,x,w],[s,s,s,end]]
[2,s2,[y,z,x,w],[s,s,s,end]]
[3,s1,[z,x,w,y],[s,s,s,end]]
[3,s2,[z,x,y,w],[s,s,s,end]]

The domain model has changed.
Will you update the message file? (y/n) | y.
game_form? | gM.
scc? | f.
agents? | [1,2,3].
reduced test by 0-integers? (y/n) | y.
delayed resolution for 0-integers? (y/n) | n.
ok
output_file_open
ファイルに書出しています。

Yes
3 ?- mechanism(A,[M1,M2,M3],C).

No
4 ?- [messages].
messages compiled, 0.00 sec, 12,320 bytes.

Yes
5 ?- mechanism(A,[M1,M2,M3],C).
M2 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
A = gM(1, f)
C = [x]
M3 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
M1 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0

Yes
6 ?- nash(full,x,s1,[M1,M2,M3],gM(1,f),h0,E,Is,NE).

Now checking whether the message profile.

```
For state s1,outcome=x   [in,f],rule=1
and message profile is
       [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
       [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
       [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
 no output file stream has opened.
wait..wait..wait..
 agent=1,P1s=[2],Czs=[x,z],Lcc=[w,x,y,z] <is_best_response>
 agent=2,P1s=[2],Czs=[x],Lcc=[w,x] <is_best_response>
 agent=3,P1s=[2],Czs=[x],Lcc=[w,x,y] <is_best_response>
Br_Members=[1,2,3]

This action profile is a Nash equilibrium.

NE = yes
E = environment([[1,2,3],[s1,s2],[w,x,y,z]],
 [3,2,4], [[[x,z,y,w],[y,z,x,w],[z,x,w,y]],[[x,z,y,w],[y,z,x,w],[z,x,y,w]]])
Is = [1,2,3]
M1 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
M2 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0
M3 = [[x,z,y,w],[y,z,x,w],[z,x,w,y]], x, 0

Yes
7 ?- tests_for_scc(f,[1,2,3],O).

O = [no,no,no,no,bad(w),nvp,rvp,unan,no,no,no,neli,mlib]

Yes
8 ?- tests_for_scc(g,[1,2,3],O).

O = [mm,no,no,no,no,nvp,rvp,unan,no,no,no,neli,mlib]

Yes
9 ?- tests_for_scc(g1,[1,2,3],O).
```

O = [mm,em,no,no,no,nvp,rvp,unan,no,no,no,neli,mlib]

Yes

References

Danilov, V. (1992). Implementation via Nash equilibria. Econometrica 60(1): 43-56.

Dutta, B. and A. Sen(1991). A necessary and sufficient condition for two-person Nash implementation. Review of Economic Studies 58:121-8.

Jackson, M.O. and T.R. Palfrey(2001). Voluntary implementation. Journal of Economic Theory 98:1-25.

Kraus, S.(2001). Strategic Negotiation in Multiagent Environment. MIT Press.

Maskin, E. (1999). Nash equilibrium and welfare optimality. Review of Economic Studies 66:23-38.

Maskin, E. and T. Sjostrom(2002). Implementation theory. In K. Arrow, A. Sen, and K. Suzumura (eds.), Handbook of Social Choice and Welfare, Vol. 1, North-Holland, pp.237-288.

Moore, J. (1993). Implementation, contracts, and renegotiation in environments with complete information. In J.-J. Laffont (ed.), Advances in Economic Theory, Cambridge University Press. pp.182-282.

Moore, J. and R. Repullo(1990). Nash implementation: A full charaterization. Econometrica 58(5):1083-99.

Peleg, B. (1998). Effective functions, game forms, games, and rights. Social Choice and Welfare15:67-80.

Saijo, T. (1988). Strategy space reduction in Maskins theorem: Sufficient conditions for Nash implementation. Econometrica 56(3): 693-700.

Shoham, Y. (1994). Artificial Intelligence Teqniques in Prolog. Morgan Kaufmann.

Sjostrom, T. (1991). On the necessary and sufficient conditions for Nash implementation. Social Choice and Welfare 8:333-40.

Su, S.-C. (1996). An algorithm for checking strong Nash implementability. Journal of Mathematical Economics 25: 109-22.

Wielemaker, J. (1994). SWI-Prolog 1.6 Reference Manual. In Kantrowitz(ed.), CMU AI Repository. Prime Time Freeware for AI, Issue 1-1. (software and manual is downloadable from URL http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/lang/prolog/impl/prolog/swi_pl/0.html)

Worldridge, M.(2000). Reasoning about Rational Agent. MIT Press.

Yamato, T. (1992). On Nash implementation of social choice correspondences. Games and Economic Behavior 4:484-92.