Implementing Nash Implementation Theory on Prolog: A Logic Programming Approach

Kenryo Indo
Department of Management, Faculty of Economics, Kanto Gakuen University

Oct 14, 2002

Abstract. Implementation theory explore the mathematical conditions and mechanisms which exactly predict the possibility and impossiblity by using rational autonomous agents. The necessary and sufficient conditions for Nash-implementability has explicated in early 1990's. In this paper, we argued a modeling and simulation of Nash implementation theory using Prolog. It was straightforward to devise such models via logic programming of Prolog. As for very small unrestricted domain with two agents and two outcomes, an impossibility result (of Hurwicz-Schmeidler) has reproduced in our experiments. But our naive simulation method shed light on the insight problem of intelligent experts who design the implementability conditions and mechanisms.

Keywords: Nash implementation, game theory, Prolog, Logic programming

1. Introduction

The necessary and sufficient condition for implementability in Nash equilibria had finally explicated in early 1990's (Moore and Repullo, 1990; Dutta and Sen, 1991) since the problem has set up in 1977 by Eric Maskin (Maskin, 1999).

In this paper, we argue a modeling and simulation for Nash implementation theory using Prolog. By logic programming of Prolog, it is straightforward to devise the preference domain models, the SCCs, and the game theoretical mechanisms. And also we can generate all implementable SCCs for very small domain models. Via computational experimentation we can observe some basic results in this field summarized in section 5 has reproduced.

The rest of the paper is organized as follows: In section 2 we explain the motivation behind the current paper informally. Section 3 is a brief introduction to the logic programming with Prolog and its application for the rational choice. In section 4 we model a preference domain, a social choice correspondence (SCC), and a Nash equilibrium. In section 5 we summarizes the known results of Nash implementation theory. The section 6 we model a mechanism and demonstrate some naive simulations with it. A mechanism consists of a message space, best responses of agents, and sets of Nash equilibrium. In the last section,

by exploiting the backtracking of *Prolog*, we discuss a method of auto-generation-and-test for the possible SCCs given a domain model. As for 'ud22' a very small unrestricted domain with two agents and two outcomes, we observe that an impossibility result by Hurwicz-Schmeidler has reproduced in our experiments.

2. Why Simulate Rational Agents with Mechanisms

Simulation and modeling of a society as a multi-agent system, i.e., a system of artificial agents, is new stream of social science. It has the technological foundations from (distributed) artificial intelligence, the theories of computationally implemented intelligent programs such as logical or non-logical knowledge representation, automated-reasoning, 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)).

The new approach set out a tuple of questions, however which is classical in social science, to both engineers and researchers who are interested in multi-agent systems.

- How the cooperation of distributed autonomous agents can be attained? (*attainability*)
- How the social objective and the each individual's objective are balanced?(*incentive compatibility*)
- Is there any desirable conditions for the institutive / administrative rules for such agent society? (*desirability*)

And the new stream weights more on the computational aspect of these classical questions.

- 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? (*design complexity*)
- 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 positively? (*experimentation*)
- And that the theory with validated proof in mathematical language is also computationally tractable if appropriately implemented on the computer? (*computational simulation*)

Whereas there are also candidates for the theoretical foundations of a multi-agent systems from traditional veins of mathematical theory of rational decision making. For example, Kraus(2001) analyses multi-agent negotiation systems by the game theoretical framework. The current paper will focus on Nash implementation theory (Maskin, 1999; Moore, 1993; Maskin and Sjostrom, 2002), an applied field of game theory. The theory has tried to answer these questions mathematically except for the last two. Recently the researchers of experimental economics have been challenging mainly the second last questioin. The current paper aims to contribute to both of them.

Mathematically, the goal of implementation theory is stated as that the correspondence of the equilibrium outcomes of the noncooperative game must coincide with a social choice correspondence (SCC) $f : R \rightarrow 2^A$, normally excluded empty set, for all agents' admissible orderings over alternatives (i.e., a model of preference domain). Of course this is not always possible.

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 preferences (what the agents mutually know) is not known to the planner.

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 "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.

Thus (Nash-)implementation theory provides mathematical conditions and mechanisms which exactly predict, at least in principle, what is possible (or impossible) by rational autonomous agents. Definitions and theorems are summarized in section 5.

Whereas it is the most basic part of mechanism design and many other researchers advanced this field in last two decades (Maskin and Sjöstr̈om, 2002), as for the necessary and sufficient condition for Nash implementation including $n = 2$, it was many years later since the problem has set out. The necessary condition is monotonicity introduced by E. Maskin in 1970's. Maskin proved with his mechanism it is also sufficient if $n \geq 3$ and NVP is satisfied. But NVP is not necessary (Maskin, 1999).

Finally the two groups of game theorists independently in early 1990's (Moore and Repullo, 1990; Dutta and Sen, 1991). Moore and Repullo introduced *Condition* μ for $n \geq 3$, and *condition* $\mu 2$ (i.e.,

in addition to the condition, self-selection with maximal elements) for $n = 2$. 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).

As for unrestricted domain (i.e., all possible combination of orderings) or its strict part, the condition reduced to essential monotonicity for $N \geq 3$ (Danilov, 1992; Yamato, 1992), 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.

Whereas I have been interested in cognitive science as well as management science, I think that implementation theory is the very mathematical analogue of a social aspect of human intelligence what governs his / her own future behavior by rules or artificials once designed and agreed.

Prolog-based, or more generally logic programming-base modeling and simulation provides us useful tool to visualize the abstract objects and events in the theory. The task is of straightforward. However, the computational intractability we faced in naive simulation method shed light on an aspect of "insight" of intelligent human experts of this field to design the theoretical conditions and mechanisms.

3. Automated deduction by `Prolog`

In this section we briefly review the `Prolog`, a computer language which is familiar to the AI researchers (Shoham, 1994), but is not so the economic scientists. `Prolog` is a one of the famous computer language to develop an artificial intelligent reasoner or a problem solver 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.

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 J.R.Robinson. For example the term `X`, a variable, is unified with `a`, `b`, `f(a,Z)`, and so on.

3.1. Logical representation for knowledge

Resolution is an outcome of syllogism 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, a knowledge base that consists of 2 facts and 1 rule:

```
bird(a).
fly(b).
fly(X):- bird(X).
```

is 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."

Each clause in `Prolog` DB represents a horn-clause, i.e., a finite disjunctive of literals (or 'terms') with at most a single positive literal which must end with a period '.'. For example "$L_h \vee \neg L_{b1} \vee \neg L_{b2} \vee \ldots \vee \neg L_{bn}$.", or equally, "$L_h \leftarrow L_{b1} \wedge L_{b2} \wedge \ldots \wedge L_{bn}$.", is a clause with a positive literal L_h. Any literal is a term or term with its negation. The standard syntax of `Prolog` is '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 a delimiter, a function of them using functor, or a list. Each of `bird` or `fly` is the functor of these terms.

The symbol ':-', called *neck* of a rule, is special functor which separates a head of rule, ex., `fly(X)` of above rule, 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 '$\leftarrow$' (*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).`" $\leftrightarrow$ "`bird(X):- true.`".

Each of `true /0` and `fail /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 '->' 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.

3.2. 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. For example, "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).
```

Here is an example of a rule using member /2.

```
bird(X):-member(X, [swallow, hawk, penguin]).
clans_of_econ(X):-member(X, [macro, micro, o_meters]).
```

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.

3.3. 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 model of rational choice in Figure 1.

```
  < program >
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 >
?- rational_choice(1,s1,X).
X = a
Yes
?- rational_choice(1,S,b).
S = s2
Yes
```

Figure 1. A simple model of rational choice.

4. Rational preference, social choice, and game

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

4.1. Rational preference

Traditionally in this field, rational choice of such agent is represented by the "preference", $R_i \subseteq A \times A$, a complete, transitive, and reflective binary relation, i.e., a 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 a fact that an alternative 'a' is preferred to an alternative 'b' by the agent j. Indifference relation, denoted by $a\ I_j\ b$, is defined as $a\ R_j\ b \wedge b\ R_j\ a$. We write its strict part of R_j, a linear order, as P_j. It is defined as $a\ P_j\ b \wedge \neg\ a\ I_j\ b$.

In Figure 2 we display two examples of (restricted) preference domain as Prolog programs. The first example is named 'ks', a domain of 2 agents, 3 outcomes, and 2 states. And the second is named 'jp', a domain of 3 agent, 4 outcomes, and 2 states.

In our Prolog-based modeling, a predicate preference is most basic element for any other preference-relevant objects, even though a binary *is_prefer_to* is a derivative from it (see Figure 4). Since the preference is

```
/*  samples of the preference domain  */
% domain ks: the King Solomom example.
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]).

% domain jp: a voting example in Jackson and Palfrey (2001)
preference(1,S,[x,z,y,w]):-member(S,[s1,s2]).
preference(2,S,[y,z,x,w]):-member(S,[s1,s2]).
preference(3,s1,[z,x,w,y]).
preference(1,s2,[x,z,y,w]).
preference(2,s2,[y,z,x,w]).
preference(3,s2,[z,x,y,w]).
difference(_,_,[s,s,s,end]).
```

Figure 2. Samples of preference domain models.

```
/* 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).
```

Figure 3. Summary of domain variables.

complete, transitive, and reflective, any ordering can be represented by a (flatten) 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).

Prolog based models of R_j, I_j, and P_j are displayed in from Figure 4. With a predicate `prefer_profiles` in Figure 5 we models $\mathcal{R}$ a collection of all possible preference profiles. And also in Figure 5, with a predicate `environment` we summarize all of alternatives, agents, preferences, and states.

```
/*  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).
```

Figure 4. Binary 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).
```

Figure 5. Preference profiles.

4.2. Social choice correspondence

We can extend the rational choice domain to the social choice environment easily. The code of Prolog is displayed in Figure 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

```
/* 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(1990).
scc(mr,s1,[c]).
scc(mr,s2,[d]).
scc(mr,s3,[c]).
scc(mr,s4,[c]).
```

Figure 6. Examples of social choice correspondence (SCC).

correspondence (SCC) is SCR which maps a subset of the outcomes (Figure 6). Social choice function is single-valued SCR.

4.3. Games and Nash equilibrium

The elements of a *(standard-form) game* (of the standard form under complete information) in the game theory is, $N = 1, 2, ..., n$, the set of *agents*(i.e., the *players* of the game), $S_i = s_{i1}, s_{i2}, ...$ for each $i \in N$, the set of *strategies* (i.e., the *messages* for a mechanism) of each player, $A = a, b, c, ...$, the set of *game outcomes* (i.e., the *alternatives*), $g : S \rightarrow A$, $S = \prod_{i \in N} S_i$ the *outcome function*, and R_i for each $i \in N$,the set of *preference* (or utility function) of each agent defined over the set of outcomes and $\mathcal{R} := \prod_{i \in N} R_i$.

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 $L(a, R_j) := \{x \in A \mid aR_jx\}$, and the attainable set is $C(a, R_j) := \{x \in A \mid (\exists s)a \in g(s), x \in g(s/s'_j), s/s'_j \in S\}$ where $g(\cdot)$ is the outcome function and $s/'_j$ represents unilateral mutation of j's strategy.

A *Nash equilibrium* (NE) of a game $\Gamma = \langle N, S, A, g, \mathcal{R} \rangle$ is defined as $b \in A$ such that $C(b, R_i) \subseteq L(b, R_i)$ for all $i \in N$, which represents the best response profile. We can redefine it more explicitly by using the maximal elements of the set $X \subseteq A, M(X, R_j) := \{a \in A \mid a \in X \subseteq L(a, R_j)\}$, a NE as $b \in A$ such that $b \in M(C(b, R_i), R_i)$ for all i.

```
/*  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).
```

Figure 7. Lower contour set and maximal elements.

5. Nash implementation: theoretical results

In this section, we review a model of Nash implementation and theoretical results uncovered in the literature. And we will refer these results in Prolog based computational experiments in succeeding sections.

In the succeeding subsections, we display definitions and theorems in the literature without proof. Prolog modeling for "essential monotonicity" is displayed in appendix.

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

Definition 1. (Monotonicity) An SCC f is monotone *if* for all R, $R' \in \mathcal{R}$, $a \in f(R)$, $L(a, R_j) \subseteq L(a, R'_j)$ for all $j \to a \in f(R')$.

```
/* 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), subset_of_agents(Is,N),member(I,Is),
   alternative(C),
   game_forms(GF,Ps), mtest(GF, SCC,Msg,Is),
   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).

 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]), 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).
```

Figure 8. Programs for best response and Nash equilibrium.

Definition 2. (No Veto Power) An SCC f satisfies NVP (No Veto Power) *if* for all $R \in \mathcal{R}$, $a \in M(A, R_j)$ for all $j \neq i \rightarrow a \in f(R)$.

Definition 3. (Unanimity) f satifies unanimity *if* for all $R \in \mathcal{R}$, $a \in M(A, R_j)$ for all $j \rightarrow a \in f(R)$.

Theorem 1. (Nash implementation (Maskin,1977/98)) An SCC f is Nash implementable *only if* f is monotone. And it is also sufficient *if* f satisfies NVP and $n \geq 3$.

5.2. Hurwicz-Schmeidler impossibility with two-agent unrestricted domain

Theorem 2. (cf., Maskin(1998), Moore and Repullo(1990)) Let $n = 2$, $\mathcal{R} = \mathcal{P}$ is a strict part of unrestricted domain. $f : \mathcal{R} \rightarrow 2^A/\emptyset$ is Pareto efficient, *then* f is Nash implementable $\leftrightarrow$ f is dictatorial.

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

Let an SCC $f : R \rightarrow 2^A$, permitted an empty set, $\mathcal{R}$ is an unristricted (or universal) domain that consists of all possible preference profiles, or $\mathcal{P}$, the strict part of $\mathcal{R}$.

Definition 4. (Essential elements) For an SCC $f, j \in N, a \in X \subseteq A$ is essential for $j \in X$ if $a \in f(R)$ for some $R \in \mathcal{R}$, such that $L(a, R_j) \subseteq X$. The set of essential elements is denoted by $Ess(f, j, X)$.

Definition 5. (Essential monotonicity) An SCC f is essentially monotone *if* for all $R, R' \in \mathcal{R}, a \in f(R), Ess(f, j, L(a, R_j)) \subseteq L(a, R'_j)$ for all $j \rightarrow a \in f(R')$.

Definition 6. (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 $R \in \mathcal{R}$ such that $X \cap f(R'/R_j) = \emptyset$ for all $R' \in \mathcal{R}$.

Definition 7. (Individual rationality) An SCC f is individually rational if for all $R \in \mathcal{R}, a \in f(R) \rightarrow$ There is no $j \in N$ who blocks $L(a, R_j)$.

Proposition 1. (A part of proposition 3 of Danilov(1992) and a statement in p.49) If f is an essentially monotone SCC, then $B(f, j, X) \leftrightarrow Ess(f, j, X) \neq \emptyset$. And f satisfies NVP $\leftrightarrow$ every agent blocks only an empty set via f.

Definition 8. (MR-elements) Let $N = 1, 2$. For a pair of subsets of alternatives $X_1, X_2 \subseteq A$, a set of MR-elements is defined as
$MR(f, X_1, X_2) := \{b \in X_1 \cap X_2 \mid X_1 = L(b, R_1), X_2 = L(b, R_2)\}$.

Definition 9. (MR-property) Let $N = 1, 2$. An SCC f that has MR-property *if* for any unblocked pair (X_1, X_2) such that $X_1, X_2 \subseteq A$, $\neg B(f, 1, X_1)$, and $\neg B(f, 2, X_2)$, there is $a \in A$, an element.

Theorem 3. (Nash implementability in unrestricted domain and an empty outcome permitted (Danilov,1992)) If $n \geq 3$, an SCC $f : \mathcal{R} \to 2^A$ is Nash implementable $\leftrightarrow$ f is essentially monotone. If $n = 2$, f is Nash implementable $\leftrightarrow$ in addition f is individually rational and f has MR-property.

5.4. approximatedly universal domain(Yamato(1992). Also see Danilov(1992) footnote 2.)

Definition 10. (Condition D) $\mathcal{R}$ satisfies condition D *if* there is no triple $(a, R, j), a \in A, R \in \mathcal{R}, j \in N$, such that $X = L(a, R_j)$,
$\{b \in A \mid (\exists\, R' \in \mathcal{R}) L(b, R'_j) = X, L(b, R'_k) = A \ (\forall\, k \neq j)\} = \emptyset$.

Theorem 4. (Theorem 1 and 2 of Yamato(1992)) If admissible preferences $\mathcal{R}$ satisfies condition D, an SCC $f : \mathcal{R} \to 2^A/\emptyset$ is Nash implementable *only if* f is essentially monotone. And if $n \geq 3$, it is also sufficient.

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

Definition 11. (Condition μ) $\mathcal{R}$ is an admissible preference profiles. There exsit $B \subseteq A$ and $C(a, R_j) \subseteq A$ for all $R \in \mathcal{R}, a \in f(R), j \in N$, such that for all $R, R' \in \mathcal{R}$, (i) $a \in M(C(a, R_j), R'_j)$ for all $j \to a \in f(R')$,
(ii) $c \in M(C(a, R_j), R'_j)$ and $c \in M(B, R'_k)$ for all $k \neq j \to c \in f(R')$,
(iii) $d \in M(B, R'_k)$ for all $k \in N \to d \in f(R')$.

Definition 12. (Condition μ 2) Let $N = i, j$. In addition to condition μ, for all $R, R' \in \mathcal{R}$ such that $a \in f(R)$ and $b \in f(R')$, there exists $e \in C(a, R_i) \cap C(b, R_j)$, such that for all $R'' \in \mathcal{R}$,
(iv) $e \in M(C(a, R_i), R''_i) \cap M(C(b, R'_j), R''_j) \to e \in f(R'')$.

Theorem 5. (Full characterization (Moore-Repullo,1990)) If $n \geq 3$ (or $n = 2$), an SCC $f : \mathcal{R} \to 2^A/\emptyset$ is Nash implementable *iff* condition μ(*or* μ 2) is satisfied.

6. Mechanisms and simulated Nash implementation

In this section, the Prolog based modeling and simulation for a Nash implementation theory with game theoretic mechanisms has demonstrated. A mechanism consists of a 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.

6.1. MECHANISMS UNDER COMPLETE INFORMATION

Story telling in the context of an implementation theory is such that a social planner as a game (or 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.

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.

Roughly to say, game theorists will try to predict that the autonomously maximizing behavior of agents lead to some tuple of best responses, i.e., Nash equilibrium (or its modifications). 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 as a profile of best responses.

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 I summarizes several 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

- gDict. A simple mechanism for a dictatorial rule.
- gM. Maskin-Vind's cononical mechanism for $n \geq 3$.
- gD. Danilov's mechanism for $n = 2$.
- gMR/gMR2. Moore-Repullo's mechanism for $n \geq 3/n = 2$ (augmented by Sjöström's algorithm).

Table I. The mechanisms for Nash implementation

author	equil.	n	messages	domain	proof
E. Maskin('77/99)	Nash	≥ 3	$\mathcal{R} \times A \times Z$	restricted	if
T. Saijo('88)	Nash	≥ 3	$\mathcal{R}_{k,k+1} \times A \times Z$	restricted	if
Dutta&Sen('91a)	Nash	$= 2$	$\mathcal{R} \times A \times B \times Z$	restricted	iff
Moore&Repullo('90)	Nash	≥ 2	$\mathcal{R} \times A \times A \times Z$	restricted	iff
V. Danilov('92)	Nash	$= 2$	$2^A \times Z$	universal	iff
Dutta&Sen('91b)	Strong	≥ 2	$\mathcal{R} \times A \times B \times Z$	restricted	iff
S.-C. Suh('96)	Strong	≥ 2	$\mathcal{R} \times A \times B \times Z$	restricted	iff

A part of these Prolog code is in appendix. We postpone the discussion of Prolog simulation of these mechanisms to next subsection.

Almost commonly in those mechanisms (in Table I) 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 $B = \{0, 1\}$), 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.

6.2. Naive simulation of implementation 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 Prolog modeling of these mechanisms are displayed in appendix. In Figures 9 and 10 we demonstrate an simulation example of non-monotone SCC on 3-person retricted domain 'jp' in Figure.1 in section 4, using gM the simulated Maskin mechanism. This sample is a complete result of a test run, but only 0-normalized messages have checked for best response. An out-of SCC outcome [s2,z] became Nash equilibrium, because the monotonicity is needed to prevent a joint-lie. In Figures 11 and 12 we display the pair of results for a non dictatorial and non constant SCC on 'ud22', a small two-person unrestricted domain, using mechanism gD, a simulated Danilov mechanism. This SCC 'urd2' is one of the implementable SCCs found by the auto-checking described in the next section.

```
 **** the model specification ******
game form: gM
members of society: [1,2,3]
Scc:
f(s1) = [x,z]   f(s2) = [x]   f(s3) = [x,z]   f(s4) = [x]
is not Maskin-monotone.
is not Essentially-monotone.
checking the on-SCC patterns:
[s1,x][s1,z][s2,x]

For state s1,outcome=x  [in,f],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=[x,z],Lcc=[w,x,y,z] <best_response>
 agent=2,P1s=[2],Czs=[x],Lcc=[w,x] <best_response>
 agent=3,P1s=[2],Czs=[x],Lcc=[w,x,y] <best_response>
Br_Members=[1,2,3]
This action profile is a Nash equilibrium.

For state s1,outcome=z  [in,f],rule=1
message profile is
     [[x,z,y,w],[y,z,x,w],[z,x,w,y]], z, 0
     [[x,z,y,w],[y,z,x,w],[z,x,w,y]], z, 0
     [[x,z,y,w],[y,z,x,w],[z,x,w,y]], z, 0
 agent=1,P1s=[2],Czs=[z],Lcc=[w,y,z] <best_response>
 agent=2,P1s=[2],Czs=[x],Lcc=[w,x,z] <best_response>
 agent=3,P1s=[2],Czs=[x],Lcc=[w,x,y,z] <best_response>
Br_Members=[1,2,3]
This action profile is a Nash equilibrium.

For state s2,outcome=x  [in,f],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=[x,z],Lcc=[w,x,y,z] <best_response>
 agent=2,P1s=[2],Czs=[x],Lcc=[w,x] <best_response>
 agent=3,P1s=[2],Czs=[x],Lcc=[w,x,y] <best_response>
Br_Members=[1,2,3]
This action profile is a Nash equilibrium.
```

Figure 9. A naive simulation of Nash implementation via canonical mechanism gM for a voting problem in Jackson and Palfrey (2001): the On-SCC part

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

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

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

This action profile is a Nash equilibrium.


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

Statistics
cputime= [2.500000,increased from,72.000005]
inferences= [869792,increased from,857188]
```

Figure 10. A naive simulation of Nash implementation via canonical mechanism gM for a voting problem in Jackson and Palfrey (2001): the Off-SCC part.

7. Checking implementability on a small unrestricted domain

In this section, we demonstrate a screening method for the implemetable SCCs on given domain. Using the backtracking incorporated in Prolog, we can generating possible SCCs recursively and automatically verifies the conditions of Nash-implementability under the given domain model (in Fig.12). Since this program for auto-generating SCCs in Fig.13

```
 **** 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] <best_response>
 agent=2,P1s=[1,2],Czs=[x,y],Lcc=[x,y] <best_response>
Br_Members=[1,2]

This action profile is a Nash equilibrium.

<----  omitted  ---->
```

Figure 11. A Nash implementation via Danilov mechanism gD for a non constant, non dictatorial SCC for agent 1 on ud22:the On-SCC part.

```
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] <best_response>
 agent=2,P1s=[1,2],Czs=[x,y],Lcc=[x] <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] <not_best_response>
 agent=2,P1s=[2,3],Czs=[x,y],Lcc=[x] <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]
```

Figure 12. A Nash implementation via Danilov mechanism gD for a non constant, non dictatorial SCC for agent 1 on ud22: the Off-SCC part.

```
tests_for_scc(F,Is,Goals):-
  Goals=[G1,G2,G3,G4,BO,G5,G6,G7,G8,G9,G10],
  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).
```

Figure 13. Automated testing program for possible SCCs given domain model.

is iterative mutation based on already settled domain model, we can use it for any model which have made to test at least neglecting the computational load argued bellow.

In Tables II–IV, we display summary results in which the data from auto-generating simulation of all possible SCCs on 'jp' and 'ud22'. In Table II, we demonstrate Nash-implementable SCCs on a domain 'jp' by checking whether the SCC satisfies *em*, which is a sufficient condition for $N \geq 3$ as for restricted domain (Yamato, 1992).

A model 'ud22' is strict part of an 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 Table III.

The computational load of this simulation can be evaluated as the size of SCC space enumerated. Since in ud22 the number of states (i.e., all possible preference profiles) is 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 this domain 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.) We found 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).

There are 7 SCCs that are consistent, that is each of them has no empty set in each state, including the constant (2) and the dictatorial (2). The other 6 SCCs have empty output in some state or no output at all.

Fortunately, we know the theorems for unrestricted domain for $n = 2$ and $N \geq 3$(Nash implementation theorems (Danilov, 1992) and famous impossibility result for $n = 2$. We can observe a Hurwicz-Schmeidler impossiblity result when focused on Pareto efficient (par=1) non-empty (emptiness=0) SCCs in Table IV. But if we try "ud23", the computational complexity of this approximation to be

$$((2^3)^{3!})^2 = 325 \times 10^{39}.$$

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. 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.

Appendix

```
% ------------------------------------------------------ %
%  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
```

Table II. Automatically generated SCCs which satisfy *em* on 'jp'.

sc1	s2	satisfied conditions
[z]	[z]	[mm,em,no,no,bad(w),nvp,rvp,unan,po,no,dict]).
[y]	[y]	[mm,em,no,no,no,nvp,rvp,unan,po,no,dict]).
[w]	[y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,y]	[y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[y,z]	[y,z]	[mm,em,no,no,no,nvp,rvp,unan,po,no,no]).
[w,z]	[y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,y,z]	[y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[x]	[x]	[mm,em,no,no,bad(w),nvp,rvp,unan,po,no,dict]).
[x,z]	[x,z]	[mm,em,no,no,bad(w),nvp,rvp,unan,po,no,no]).
[x,y]	[x,y]	[mm,em,no,no,no,nvp,rvp,unan,po,no,no]).
[w,x]	[x,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,y]	[x,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[x,y,z]	[x,y,z]	[mm,em,no,no,no,nvp,rvp,unan,po,no,no]).
[w,x,z]	[x,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,y,z]	[x,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w]	[w]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,z]	[w,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w]	[w,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,y]	[w,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,z]	[w,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,y,z]	[w,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x]	[w,x]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,z]	[w,x,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x]	[w,x,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,y]	[w,x,y]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,z]	[w,x,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).
[w,x,y,z]	[w,x,y,z]	[mm,em,no,no,no,nvp,rvp,unan,no,no,no]).

```
% 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)
```

Table III. Automatically generated SCCs which satisfy $em + ir + mr$ on 'ud22'.

emptyness	s1	s2	s3	s4	satisfied conditions
4	[ ]	[ ]	[ ]	[ ]	[mm,em,mr,ir,no,no,no,no,po,co,no]
2	[x]	[ ]	[ ]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no]
1	[x]	[y]	[ ]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no]
1	[x]	[x]	[ ]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no]
1	[x]	[ ]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no]
0	[y]	[y]	[y]	[y]	[mm,em,mr,ir,no,no,no,no,no,no,no]
0	[x,y]	[y]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,no,no,no,]
0	[x]	[x]	[y]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,dict]
1	[x]	[ ]	[x]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,no]
0	[x]	[y]	[x]	[y]	[mm,em,mr,ir,no,no,rvp,unan,po,no,dict]
0	[x]	[x]	[x]	[x]	[mm,em,mr,ir,no,no,no,no,no,no,no]
0	[x]	[x]	[x]	[x,y]	[mm,em,mr,ir,no,no,rvp,unan,no,no,no]

Table IV. A Hurwicz-Schmeidler impossibility for Pareto correspondence.

s1	s2	s3	s4	mm	em	mr	ir	bad	nvp	rvp	unan	po	dict
[x]	[y]	[y]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[x]	[y]	[y]	1	1	1	1	0	0	1	1	1	1
[x]	[x,y]	[y]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[y]	[x]	[y]	1	1	1	1	0	0	1	1	1	1
[x]	[x]	[x]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[x,y]	[x]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[y]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[x]	[x,y]	[y]	1	1	0	1	0	0	1	1	1	0
[x]	[x,y]	[x,y]	[y]	1	1	0	1	0	1	1	1	1	0

```
    ),
    (
     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).

% ------------------------------------------------------  %
/*   the mechanisms part   */
% ------------------------------------------------------  %
% common part %
mechanisms([gDict,gM,gD,gMR,gMR2]).
mechanism(G,E,Msg,Z):-
    E =.. [environment,[Is,_Ss,_As],[_N,_K,_L],_Rs],
    G =.. [GF,_P,Scc],
    mtest(GF, Scc,Msg,Is),
    game_form(G,E,Msg,Z),!.
% ------------------------------------------------------  %
%  message spaces
% ------------------------------------------------------  %
/* message space for gDict, a dictatorial mechanism.
mtest(gDict,Scc,Msg,[J1,J2]):-
    scc(Scc),
    range(Scc,B),
    two_person([J1,J2]),
    Msg = [X1,X2],
    alternative(X1),
    alternative(X2),
    subset([X1,X2],B).
mtest(gDict,Scc,Msg,Js):-
    scc(Scc),
    range(Scc,B),
    subset_of_agents(Js,N),N>2,
```

```
    bag0(Msg,B,N).
/*  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]).

/* <------ omitted --------> */

   %%%%%%%%%%%%%%%%%%%%%%%%
   %%  the  game  forms  %%
   %%%%%%%%%%%%%%%%%%%%%%%%
%  available game forms in this system:
game_forms(gDict,[1]).
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).

% ------------------------------------------------------  %
% gDict, a very simple dictatorial mechanism.
% ------------------------------------------------------  %
game_form(gDict(Dict,_Scc),E,Msg,[C]):-
     E = environment([Is,_Ss,_As],[N,_K,_L],_Rs),
     length(Msg,N),
     nth1(K,Is,Dict),
     nth1(K,Msg,C).
% ------------------------------------------------------  %
% 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)),
  %  R2J, the semi-agreed preference of the agent J who said
  %  a distinctive opinion. estimated same by semi-agreed.
      setof((R,X,0),Z^(member((R,X,Z),Msg)),MD),
      length(MD,2),
    % find a single deviator's message.
      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), % partial match (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
% 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),
    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),!,
    range(Scc,Range),
    roulette(Jd,[Z1,Z2],[J1,J2]),
    member(C,Range),
    dictatorship(Jd,_S,Range,C).
%%%% end of rules.
```

Acknowledgements

We have benefited greatly by a free copy of SWI-Prolog delivered by Jan Wielmaker and Amsterdam University to build and test our experimental system. We also grateful to the attendances at the 6th conference of experimental economics in Chiba for useful discussions.

References

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

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

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

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

Maskin, E. Nash equilibrium and welfare optimality. *Review of Economic Studies*, 66:23–38, 1999.

Maskin, E. and T. Sjöström. Implementation theory. In K. Arrow, A. Sen, and K. Suzumura, editors, *Handbook of Social Choice and Welfare*, pages 237–88. North-Holland, 2002.

Moore, J. Implementation, contracts, and renegotiation in environments with complete information. In J. -J. Laffont, editor, *Advances in Economic Theory*, pages 182–282. Cambridge University Press, 1993.

Moore, J. and R. Repullo Implementation via Nash equilibria. *Econometrica*, 58(5):1083–99, 1990.

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

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

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

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

Wielemaker, J. SWI-Prolog 1.6 Reference Manual. In Kantrowitz, editor, *CMU AI Repository*, URL http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/, 1994.

Worldridge, M. *Reasoning about Rational Agent*, MIT Press, 2000.

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