Using *Theorema* in the Formalization of Theoretical Economics*

Manfred Kerber[1], Colin Rowat[2], and Wolfgang Windsteiger[3]

[1] Computer Science, University of Birmingham, Birmingham B15 2TT, England
[2] Economics, University of Birmingham, Birmingham B15 2TT, England
[3] RISC, JKU Linz, 4232 Hagenberg, Austria

Abstract. Theoretical economics makes use of strict mathematical methods. For instance, games as introduced by von Neumann and Morgenstern allow for formal mathematical proofs for certain axiomatized economical situations. Such proofs can—at least in principle—also be carried through in formal systems such as *Theorema*. In this paper we describe experiments carried through using the *Theorema* system to prove theorems about a particular form of games called pillage games. Each pillage game formalizes a particular understanding of power. Analysis then attempts to derive the properties of solution sets (in particular, the core and stable set), asking about existence, uniqueness and characterization. Concretely we use *Theorema* to show properties previously proved on paper by two of the co-authors for pillage games with three agents. Of particular interest is some pseudo-code which summarizes the results previously shown. Since the computation involves infinite sets the pseudo-code is in several ways non-computational. However, in the presence of appropriate lemmas, the pseudo-code has sufficient computational content that *Theorema* can compute stable sets (which are always finite). We have concretely demonstrated this for three different important power functions.

1 Introduction

Theoretical economics may be regarded as a branch of applied mathematics, drawing on a wide range of mathematics to explore and prove properties of stylized economic environments. Since the Second World War, one particularly important body of theory has been game theory, as introduced by von Neumann and Morgenstern [17] and successfully developed by John Nash.

The game theory stemming from von Neumann and Morgenstern has become known as cooperative game theory; it allows abstraction from the details of how agents might interact, instead focusing directly on how final outcomes may or may not dominate each other. As dominance is a binary relation, cooperative game theory has lent itself naturally to axiomatic analyses. The game theory

* The first author would like to thank the 'Bridging The Gap' initiative under EPSRC grant EP/F033087/1 for supporting this work. The first and second author would like to thank the JKU Linz for its hospitality.

stemming from Nash has become known as non-cooperative game theory; it is explicitly constructive, requiring specification of a game form that details the set of permissible moves available to agents. Solutions to cooperative games have been difficult to calculate relative to non-cooperative games, contributing to the latter body of theory's current preeminence within game theory.[4]

In the current work, we use *Theorema* [19] to formalize a particular cooperative game form, called pillage games, and to prove formally certain properties of them. Pillage games, introduced in [8], form an uncountable set of cooperative games, taking the two best known classes of cooperative games (those in characteristic and partition function form) as boundary points. At the same time, even though each is defined over an uncountable domain, their structure has thus far been sufficient to avoid Deng and Papadimitriou's [5] pessimistic conclusions that whether a solution even exists may be undecidable. Thus, pillage games provide a class of games for analysis that is simultaneously rich and tractable. To our knowledge, this represents the first attempt to formally prove properties of a cooperative game. Related is work on other axiomatic proofs within economic theory which have been formalized. Arrow's theorem in social choice has attracted the most attention, including studies by Wiedijk [18] using *Mizar*, Nipkow [15] using *HOL*, and Grandi and Endriss [6] using *Prover9*. Non-cooperative game theory has also received attention, including by Vestergaard and co-authors [16] with *Coq*.

The formalization of a particular game form in economics can be interesting to computer science for at least two reasons, and to economics for at least one. For computer science, economics is a relatively new area for automated theorem proving and therefore presents a new set of canonical examples and problems. Secondly, it is an area which typically involves new mathematics in the sense that axioms particular to economics are postulated. Further, in the case of pillage games, the concepts involved are of a level that an undergraduate mathematics student can understand easily. That is, the mathematics is of a level that should be much more amenable for formalizations than research level mathematics.

For economics, as in any other mathematical discipline [11], establishing new results is typically an error-prone process, even for the most respected researchers. We cite but two examples from cooperative game theory:

1. In founding game theory, von Neumann and Morgenstern [17] assumed that one of the key concepts of the field, the so-called stable set (by them just called the "solution"), always existed in games in characteristic function form. This was subsequently demonstrated by Lucas [12] to be incorrect.
2. Nobel Prize winning economist and game theorist Maskin [13] claimed that certain properties of a game in partition function form extended from $n = 3$ to $n > 3$. However, de Clippel and Serrano [4] found counterexamples with $n > 3$.

It is understandable that such problems occur since typically for any new axiom set humans have initially no or only limited intuition. This way it is easy to as-

[4] For example, Gambit [14] solves a broad class of non-cooperative games.

sume false theorems and to overlook cases in proofs. Proofs found in mathematics in general and theoretical economics in particular, can be viewed from a logical point of view more like proof plans. That is, not all details are given, hidden assumptions may be overlooked, proof steps may be incorrect, generalizations may not hold. Thus, any mathematical discipline, including theoretical economics, can benefit from formalizing proofs since this will make proofs much more reliable. However, there are other potential benefits. For instance, in experimenting with axiomatizations it is much easier to reuse proof efforts. Furthermore the dependencies of assertions can be accessed more easily and experiments with the computational content of theorems becomes possible which without computer support would be time consuming and error-prone.

In this work we report on our experiments with the *Theorema* system which we conducted to formalize pillage games, to prove certain properties of them, and to exploit computational features in them. Full verification of the formal statements is an important goal of these experiments. However, we also look at less labour extensive ways of making use of *Theorema* and will discuss this.

The paper has the following structure. In the next section we give a brief introduction to pillage games. In Section 3 we present the representations in *Theorema* and the proofs of some formal statements which we formally proved in *Theorema*. A focus of the presentation as presented in Subsection 3.4 is the pseudo-code, which summarizes the results of the paper we formalize. This pseudo-code is non-computational in several ways. However, a mixture of proof and computation makes it possible to evaluate it in concrete cases. In Section 4 we evaluate the approach taken.

2 A Brief Introduction to Pillage Games

The class of games used for our experiments are called pillage games, a particular form of cooperative games, introduced by Jordan in [8]. In a nutshell, pillage games describe how *agents* (n in total) can form coalitions in order to redistribute all or part of the possessions of other coalitions among themselves. The possessions are described by a so-called *allocation*, a vector of n non-negative numbers which sum to one. Given two such vectors, x and y, three coalitions are induced: the *win set* of a transition from x to y is $\{i \,|\, y_i > x_i\}$; the *lose set* is $\{i \,|\, x_i > y_i\}$; the remaining agents are indifferent between x and y. Pillage is possible if and only if the win set is more powerful at x than is the lose set; if this is so, it is said that y *dominates* x.

The power of a coalition is determined by a so-called *power function*, π, a function which depends exclusively on the coalition members and the holdings of all agents. A power function must satisfy three monotonicity axioms: firstly *weak coalition monotonicity (WC)*, whereby taking a new member into a coalition does not decrease the coalition's power; secondly *weak resource monotonicity (WR)*, whereby weakly increasing the holdings of a coalition's members does not decrease the coalition's power; and thirdly, *strong resource monotonicity (SR)*, whereby strictly increasing the holdings of a coalition's members strictly

increases the coalition's power. This setting given, two sets are of interest. Firstly, the *core*, the set of undominated allocations, and secondly the *stable set*, a set of allocations such that none dominates another (called *internal stability*) but at least one dominates each non-member allocation (called *external stability*). The core always exists, and is unique, but may be empty. A stable set may not exist since it has to satisfy two conflicting properties, on the one hand it must contain sufficiently many elements so that any element not in it is dominated by an element in it (e.g. the empty set is not externally stable), on the other hand it must not contain so many elements that none dominates another (e.g. the full set of all allocations is not internally stable in a pillage game). If a stable set exists, it is finite and contains the core; uniqueness has been established for some pillage games, and no counterexamples have been found as yet. If agents are forward looking, expecting that y dominating x may allow a subsequent comparison between z and y. Jordan [8] proved that a stable set is a *core in expectation*, a set of undominated allocations given some such future expectation (and consequent comparison of x and z, rather than x and y).

Jordan [8] proved general properties about pillage games (such as the finiteness of the stable set) and investigated the possibilities of three particular power functions for arbitrarily many agents. Kerber and Rowat [10] studied an infinite class of power functions for three agents. In particular they gave a complete characterization of the stable set for arbitrary power functions (with three agents) which satisfy three additional axioms. The first axiom is *continuity* in the resources; although definable with an ϵ-δ-statement, the intermediate value theorem is actually used. The second axiom, *responsiveness*, requires that a coalition gaining a member with some power as a singleton strictly increases the coalition's power. The third axiom, *anonymity*, requires that power, dominance, and—consequently—the stable set are invariant under permutations of the agents; thus, an agent's identity is irrelevant to a coalition's power, merely its presence or absence, and its holdings.

3 Formalizations in *Theorema*

Within this case study, we fully proved the first three lemmas from [10]. Furthermore, we give a formalization of the pseudo-code that summarizes the results derived in that paper. In this section, we briefly give the main assertions, which we proved in *Theorema* input syntax.[5] We begin with the definition of a power

[5] The *Theorema* language syntax comes very close to how mathematicians are used to write up things. In particular, two-dimensional notation can be used both in input and output through *Mathematica*'s notebook technology, so that a *Theorema* formalization can easily be read and understood by a mathematician. In this presentation, we typeset all *Theorema* expressions in LaTeX with the aim to mimick their appearance in *Theorema* as closely as possible. Formal text blocks (definitions, theorems, lemmas, etc.) in *Theorema*, so-called environments, are of the form '*Env*[*l*, any[*v*], with[*C*], bound[*r*], *form*]', where *Env* is the type of environment, *l* is a string label used to refer to this environment, *v* lists the universal variables in

function. In all what follows, $I[n] := \{1, \ldots, n\}$ stands for the set of n agents and $X[n]$ denotes the set of all allocations for n agents.

With these preliminaries we use *Theorema* to formally define *weak coalition monotonicity (WC)*, *weak resource monotonicity (WR)*, *strong resource monotonicity (SR)*, and *power function* as follows.[6]

Definition["WC", any$[\pi, n]$, bound[allocation$_n[x]$]],

$$\mathrm{WC}[\pi, n] :\Leftrightarrow n \in \mathbb{N} \wedge \forall_{\substack{C1, C2 \\ C1 \subset C2 \wedge C2 \subseteq I[n]}} \forall_{x} \pi[C2, x] \geq \pi[C1, x]]$$

Definition["WR", any$[\pi, n]$, bound[allocation$_n[x]$, allocation$_n[y]$]],

$$\mathrm{WR}[\pi, n] :\Leftrightarrow n \in \mathbb{N} \wedge (\forall_{\substack{C \\ C \subseteq I[n]}} \forall_{x,y} \big((\forall_{i \in C} y_i \geq x_i) \implies \pi[C, y] \geq \pi[C, x] \big))]$$

Definition["SR", any$[\pi, n]$, bound[allocation$_n[x]$, allocation$_n[y]$]],

$$\mathrm{SR}[\pi, n] :\Leftrightarrow n \in \mathbb{N} \wedge (\forall_{\substack{C \\ C \subseteq I[n] \wedge C \neq \emptyset}} \forall_{x,y} \big((\forall_{i \in C} y_i > x_i) \implies \pi[C, y] > \pi[C, x] \big))]$$

Definition["powerfunction", any$[\pi, n]$, powerfunction$[\pi, n] :\Leftrightarrow \wedge \begin{cases} \mathrm{WC}[\pi, n] \\ \mathrm{WR}[\pi, n] \\ \mathrm{SR}[\pi, n] \end{cases}$]

In this formalization, we decided not to put explicit conditions on variables in definitions assuming that defined expressions will only be used "as intended by definition." In theorems and lemmas, of course, we explicitly list all preconditions. Furthermore, we split the definitions into individual environments, although the *Theorema* language would allow to collect several formulae into one environment. We decided to proceed this way because we later want to use the definitions on an individual basis, and the current version of *Theorema* allows to access only whole environments, not single formulae.

3.1 Lemma 1: Representation

The first lemma states that the power of a coalition depends only on the holdings of the members of the coalition, but not on the holdings of the other agents.

Lemma["powerfunction-independent", any$[\pi, n, C, x, y]$,
with[allocation$_n[x] \wedge$ allocation$_n[y] \wedge C \subseteq I[n] \wedge$ powerfunction$[\pi, n]$],

form, C is a formula expressing a condition on v, r specifies ranges for bound variables in *form*, and finally *form* is a single formula or a sequence of formulae. After evaluating a formal text block in a *Theorema* session, it can be referred to (e.g. in a call to a prover) by '*Env*[*l*]'. Concretely, e.g. in (WC), the 'any$[\pi, n]$' makes the definition applicable for all π and all n and the 'bound[allocation$_n[x]$]' makes the 'for all x' to actually range over all allocations x of length n. The real convenience of the 'bound'-construct will be revealed once we collect several definitions into one 'Definition'-environment, e.g. one definition for the axioms (WC), (WR), and (SR), where one 'bound'-statement would suffice for all three axioms. For more details we refer to [1].

[6] For full formalizations in *Theorema* together with proofs see also `http://www.cs.bham.ac.uk/~mmk/economics/theorema`.

$$\underset{i\in C}{\forall}\ (x_i = y_i) \implies (\pi[C,x] = \pi[C,y])\,]$$

In order to prove this, we call the *Theorema* predicate logic prover, see [2], by

Prove[Lemma["powerfunction-independent"],
 using→{Definition["powerfunction"], Definition["WR"], Proposition["ref/as"]},
 by → PredicateProver, SearchDepth → 100],

which uses the definition of power function, the axiom (WR), and the following (trivial) property of the partial ordering $\geq$ on real numbers in its knowledge base:[7]

Proposition["ref/as", any$[a,b]$, $(a = b) \Leftrightarrow (a \geq b \wedge b \geq a)$].

With these settings the lemma is proved fully automatically. It also generates a proof if it is given not only the axioms it needs for a proof but the full theory. *Theorema* generates a human readable proof of the proof search, which is ten pages long for a proof with the full theory, and five pages for the setting used above. This proof can be automatically tidied to a three page proof, which contains only those steps necessary for the final argument, see Fig. 1 to get an impression of how *Theorema* presents a human readable proof (further information is at `http://www.cs.bham.ac.uk/~mmk/economics/theorema`).

3.2 Lemma 2: Domination

The second Lemma states that when the opposing coalitions consist of one element each and the power function is anonymous then the the coalition which wins is the one with bigger holdings. In order to formalize Lemma 2 we need in addition to formalize the win set of a transition from an allocation x to an allocation y (those agents which benefit from the transition) and the lose set (those agents to whose detriment the transition is) as well as the notions of domination (x domintates y if the coalition consisting of the win set is, in x, more powerful than the coalition consisting of the lose set) and anonymity (invariance under permutations of agents). See also Section 2.

Definition["WinLose", any$[n,x,y]$,
 $W[n,x,y] := \{\underset{i\in I[n]}{i} \mid y_i > x_i\}$ "W"
 $L[n,x,y] := \{\underset{i\in I[n]}{i} \mid x_i > y_i\}$ "L"]

Definition["domination", any$[\pi,n,x,y]$,
 dominates$[y,x,\pi,n] :\Leftrightarrow n \in \mathbb{N} \wedge \pi[W[n,x,y],x] > \pi[L[n,x,y],x]$]

Definition["anonymity", any$[\pi,n]$, bound[allocation$_n[x]$, allocation$_n[y]$]],

$$\text{anonymous}[\pi,n] :\Leftrightarrow n \in \mathbb{N} \wedge \underset{\substack{\sigma\\ \text{permutation}[\sigma,I[n]]}}{\forall}\ \underset{\substack{Cx,Cy,x,y\\ Cx\subseteq I[n]\wedge Cy\subseteq I[n]}}{\forall}$$
$$\underset{i}{\forall}\ (((i \in Cx) \iff (\sigma[i] \in Cy)) \wedge (x_i = y_{\sigma[i]})) \implies (\pi[Cx,x] = \pi[Cy,y])]$$

[7] This proposition can in turn be proven fully automatically using reflexivity of $\geq$ for the part from left to right and anti-symmetry of $\geq$ for the opposite direction.

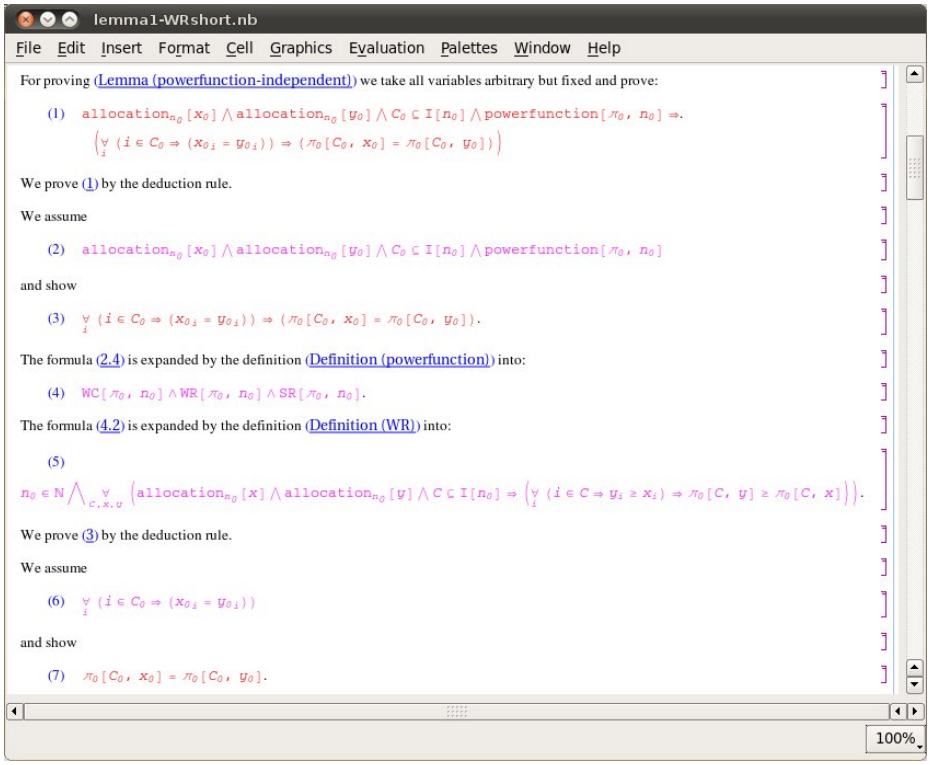

Fig. 1. A human readable proof in *Theorema*.

Lemma 2 can then be formulated as follows:

Lemma["ANdominates", any$[n, x, y]$, with$[n \in \mathbb{N} \wedge n \geq 2 \wedge$ allocation$_n[x] \wedge$ allocation$_n[y] \wedge (W[n, x, y] = \{1\}) \wedge (L[n, x, y] = \{2\})]$,

$$\forall_{\substack{\pi \\ \text{anonymous}[\pi,n] \wedge \text{powerfunction}[\pi,n]}} (\text{dominates}[y, x, \pi, n] \Leftrightarrow x_1 > x_2)\Big]$$

While this is apparently intuitively correct, a formal argument is not completely trivial. Since anonymity involves permutations of the agents, a formal proof requires auxiliary knowledge about permutations. In the concrete case, we provide a lemma about permuted tuples, namely that when swapping the first two elements in a tuple the second element in the new tuple is equal to the first of the original:

Lemma["perm swap", any$[x]$, perm$[x, \sigma_{1,2}]_2 = x_1$],

where perm$[x, \sigma]$ stands for the tuple x permuted by σ, and $\sigma_{1,2}$ is the permutation swapping the first and second component while leaving the rest unchanged. Furthermore, we use a lemma saying that the power does not change when swapping agents 1 and 2, i.e.

Lemma["ANswap", any$[n, \pi, x]$, with[anonymous$[\pi, n] \wedge$ allocation$_n[x]$],
$\pi[\{1\}, x] = \pi[\{2\}, \text{perm}[x, \sigma_{1,2}]]$].

Lemma["perm swap"] can be verified without effort based on the definitions only, whereas Lemma["ANswap"] needs the definitions of the concepts involved plus idempotency of swapping, which is trivial but still automatically verified based on the definition of $\sigma_{1,2}$. These proofs, and all that will follow, have been generated by the *Theorema* set theory prover, see [20]. Equipped with Lemma["perm swap"] and Lemma["ANswap"] in the knowledge base, a three page human-understandable proof of the two directions of Lemma 2 is obtained. Note, however that the introduction of the auxiliary lemma is a eureka step, since finding the right permutation is key to the proof of our main lemma; it is difficult to see how *Theorema*—with its currently integrated theorem provers—could automatically find this permutation. This shows that the main idea of the proof requires a fairly intuitive understanding of permutations which needed to be made more explicit when instructing *Theorema*.

The crucial question is, of course, how to obtain appropriate intermediate lemmas. One approach in this kind of *theory exploration* is to always prove all properties of interactions between available concepts before introducing a new concept. This approach is advocated for instance in [3]. In the concrete case, it would require proving many properties of permuted vectors and swaps before talking about anonymity. *Theorema*, however, also supports a much more goal-oriented approach: in the case of a failing proof attempt, it displays the partial proof, allowing a human check of where resources are used, and whether *Theorema* can be better guided. In most cases, formulating an appropriate lemma is then a straightforward exercise, so that even this can be automated. There is a literature on lemma speculation from failing proofs (see e.g. [7]); some mechanisms have been implemented in the *Theorema* system as well (see [1]). In this case study, these tools have not been employed.

3.3 Lemma 3: The Core

The *dominion* $D[Y, \pi, n]$ denotes the set of all allocations that are dominated by an allocation in Y. Using this, the *core*, i.e. the set of undominated allocations, can simply be defined as $K[\pi, n] := X[n] \setminus D[X[n], \pi, n]$. The third lemma specializes to the case of three agents and consists of two parts. Firstly, if the core is empty then the tyrannical elements (one agent possesses everything) are dominated by the half splits (e.g., $\langle 1/2, 1/2, 0\rangle$ dominates $\langle 0, 0, 1\rangle$):

Lemma["Core,n=3,a", any$[\pi]$,

$$(K[\pi, 3] = \emptyset) \implies \underset{i,j,k \in I[3]}{\forall} (\text{distinct}[i, j, k] \implies t[i, 3] \in D[\{s[j, k, 3]\}, \pi, 3]) \,].$$

On the other hand, for anonymous power functions, the half way splits are never dominated by the tyrannical elements (e.g., $\langle 0, 0, 1\rangle$ does not dominate $\langle 1/2, 1/2, 0\rangle$), written in *Theorema* as follows:

Lemma["Core,n=3,b", any$[\pi]$, with[anonymous$[\pi, 3] \wedge$ powerfunction$[\pi, 3]$],

$$\underset{i,j,k \in I[3]}{\forall} (\text{distinct}[i, j, k] \implies s[j, k, 3] \notin D[\{t[i, 3]\}, \pi, 3]) \,].$$

It is clear by the definitions of D and domination, that the win and lose sets under tyrannical elements and the half splits, e.g. $W[3, t[i, 3], s[j, k, 3]]$, will play an essential role in the proofs. We then use the computational capabilities of *Theorema* in order to get some intuition about these entities. Using the built-in computational semantics of the *Theorema* language, one can do some experiments like computing $W[3, t[1, 3], s[2, 3, 3]]$ and $L[3, t[1, 3], s[2, 3, 3]]$ resulting in $\{2, 3\}$ and $\{1\}$, respectively. After some calculations of this kind, the following generalization can be conjectured as a lemma:

Lemma["s dominates t", any$[i, j, k \in I[3]]$, with[distinct$[i, j, k]$],

$$\begin{array}{l} W[3, t[i, 3], s[j, k, 3]] = I[3] \setminus \{i\} \\ L[3, t[i, 3], s[j, k, 3]] = \{i\} \end{array}],$$

The proof of this Lemma can be done by pure computation, since the universal quantifier over i, j, k boils down to just testing finitely many cases, namely the six distinct choices of $i, j, k \in I[3]$. The neat integration of proving and computing in the *Theorema* system is of great value in this kind of investigation, because the statements need no reformulation when switching between proving and computing.

For proving the first part of Lemma 3, instead of going back to the definition of the core, we use a theorem of Jordan that connects the core and the power of tyrannical allocations, see [8]. Together with Lemma["s dominates t"], this proof goes through without further complications. For the second part, *Theorema* comes up with an indirect proof using a variant of Lemma["s dominates t"] with the roles of s and t interchanged and specialized versions of the axioms (WC) and (SR) and of Lemma["ANswap"] (introduced in the context of Lemma 2), for the case $n = 3$.

3.4 Pseudo-Code and its Computational Content

The main result of [10] is a classification of the possibilities which a stable set can have in three agent pillage games with continuous, responsive, and anonymous power functions. No stable set may exist but—if one does—it may have up to 15 elements. How these elements are determined (in dependency of π) can be summarized in form of some pseudo-code. This pseudo-code can be represented as an algorithm in *Theorema* as shown in Fig. 2.

The algorithm in Fig. 2 makes use of several sets and conditions which we explain only partly in the following, since not all details are of importance in this context. Important is that certain sets (such as dyadicSet$[1, 3]$) are computational, whereas others (such as $R[1, \pi]$) are not. For details of these constructs see [10].

This algorithm is non computational in several ways. For $n = 3$, however, it is a significant improvement on the Roth-Jordan [9] algorithm to determine stable sets, since the latter is non-computational in even more ways.[8] In this section we will discuss how the algorithm, which is written in the first place to

[8] First, the Roth-Jordan algorithm starts with the core without giving an effective means for computing it. Second, for an empty core it provides no clue for finding

Algorithm[“StableSet2”, any[π],

stableSet[π] :=

$$\left\{\begin{array}{l}\left\{\begin{array}{ll}\text{“no stable”} & \Leftarrow \text{empty}[R[1,\pi]]\\ \text{where}[\mathcal{S} = \text{dyadicSet}[0,3] \cup \bigcup_{i=1,\ldots,3} S[i,\pi], & \\ \left\{\begin{array}{ll}\mathcal{S} \cup P[\pi] & \Leftarrow \neg\text{fullSet}[\mathcal{S} \cup D[\mathcal{S},\pi,3]]\\ \mathcal{S} & \Leftarrow \text{fullSet}[\mathcal{S} \cup D[\mathcal{S},\pi,3]]\\ \text{“unknown X”} & \Leftarrow \text{otherwise}\end{array}\right. & \Leftarrow \neg\text{empty}[R[1,\pi]] \Leftarrow (*)\\ \text{“unknown R”} & \Leftarrow \text{otherwise}\end{array}\right.\\ \text{dyadicSet}[1,3]\backslash\text{dyadicSet}[0,3] \Leftarrow \text{otherwise}\end{array}\right.$$

with $(*)$ to be replaced by $\pi[\{1\}, t[1,3]] \geq \pi[\{2,3\}, t[1,3]]$.

Fig. 2. Algorithm to compute a stable set written in *Theorema* notation.

summarize the results in a concise form, can be used to determine stable sets by a mixture of proving and computing.

We have tested the implementation for the three specific power functions introduced by Jordan [8], *strength in numbers* (SIN), *Cobb-Douglas* (CD), and *wealth is power* (WIP).

If a concrete power function π is given, the algorithm has first to test condition $(*)$. Since we assume that π is computable, both $\pi[\{1\}, t[1,3]] \in [0,1]$ and $\pi[\{2,3\}, t[1,3]] \in [0,1]$, hence $\pi[\{1\}, t[1,3]] \geq \pi[\{2,3\}, t[1,3]]$ can be decided. If the condition is false the stable set is given by the last line of the algorithm.[9] It is computed by *Theorema* as the set $\{\langle 1/2, 1/2, 0\rangle, \langle 1/2, 0, 1/2\rangle, \langle 0, 1/2, 1/2\rangle\}$. This case was concretely tested for the ‘strength in numbers’ power function, defined as

$$\text{SIN}\pi_\nu[C,x] := \sum_{i \in C} (x_i + \nu)$$

(with $\nu > 1$) for the concrete value of $\nu = 2$.

If the condition is true, however, the algorithm must check whether a particular set $R[1,\pi]$ is empty or not. For finite $R[1,\pi]$, the ad-hoc method to test $R[1,\pi] = \emptyset$ is to compute $R[1,\pi]$ (by enumerating its elements) and then comparing it to the empty set $\emptyset$, which can all be done in a *Theorema* computation. Unfortunately, the set $R[1,\pi]$ is defined in terms of the set $M[1,\pi]$, which in turn is defined in form of the set $B[1,\pi]$ of all triples, where agent 1 on its own

---

an initial iterate. Third, the iteration step is not computational since it involves the computation of undominated sets, which are typically infinite. Finally, it is not clear whether terminating at a set $\mathcal{S}$ which is not externally stable means that no stable set exists, or merely that further steps must be taken independently of the algorithm. For details, see [9] or [10].

[9] The allocations in which each agent has either nothing or $\left(\frac{1}{2}\right)^j$ for some integer $j \leq i$ form a so-called *dyadic set*, represented as dyadicSet$[i,n]$ (for n agents). For three agents the previously mentioned tyrannic allocations and half-splits as well as the ones of type $\langle 1/2, 1/4, 1/4\rangle$ play an important role in determining the stable set.

is equally powerful as agents 2 and 3 together, i.e.

$$B[1,\pi] := \{x \in X[3] \mid \pi[\{1\},x] = \pi[I[3]\backslash\{1\},x]\}\,.$$

The set $M[1,\pi]$ is then a subset of $B[1,\pi]$ such that the x_1 are maximal, and $R[1,\pi]$ those elements in $M[1,\pi]$ which are maximal from the viewpoint of agents 2 and 3. Since $B[1,\pi]$ is typically infinite, testing $R[1,\pi] = \emptyset$ by computation as described above would fail. Without any further knowledge on $R[1,\pi]$, the algorithm returns "unknown R", which indicates that it has insufficient knowledge on the set R and for this reason insufficient knowledge to determine the stable set.

If, however, for the concrete power function π there is additional knowledge that allows us to decide $R[1,\pi] = \emptyset$ (without actually computing $R[1,\pi]$), the algorithm may make use of this knowledge and continue. Concretely in the algorithm above, we provide the following lemma:
Lemma["emptyR,Cobb Douglas", any[ν], empty[$R[1,\mathrm{CD}\pi_\nu]$]],
where $\mathrm{CD}\pi_\nu$ is the 'Cobb-Douglas' power function defined as

$$\mathrm{CD}\pi_\nu[C,x] := |C|^\nu \Big(\sum_{i\in C} x_i\Big)^{1-\nu} \quad \text{for } 0 \le \nu \le 1.$$

In this case, according to the algorithm, no stable set exists. Note, however, that the knowledge must be formulated appropriately. For instance, in the algorithm above it was not possible to use the usual *Theorema* notation $R[1,\pi] \neq \emptyset$ instead of $\neg$empty$[R[1,\pi]]$: if $R[1,\pi] \neq \emptyset$ was given as an auxiliary lemma, the computation engine would have had to process negated equalities in an appropriate manner, which the current version of *Theorema* is not capable of.

If $R[1,\pi]$ is known not to be empty, further knowledge is necessary. Most importantly, does the current iteration of the computed candidate stable set together, with the allocations dominated by it, include all possible allocations? Formally, is fullSet$[\mathcal{S}\cup D[\mathcal{S},\pi,3]]$ true or false? This will typically not be computational, since $D[\mathcal{S},\pi,3]$ is infinite. Hence for any particular given power function π a corresponding lemma is necessary, which states whether the property holds or not. In case of the 'wealth is power' power function, defined as

$$\mathrm{WIP}\pi[C,x] := \sum_{i\in C} x_i,$$

the property does not hold since there are three points which are not dominated by the allocations computed so far. In this case, the stable set is computed by *Theorema* to $\mathcal{S}\cup P[\pi]$, which is evaluated directly, since $P[\pi]$ is a set of at most three elements explicitly defined in [10].

The algorithm presented above does not check whether a power function satisfies the additional axioms (continuity, responsiveness, and anonymity), or even whether the functions supplied are actually power functions (satisfying axioms WC, WR, and SR). As a consequence, the algorithm may give wrong answers if applied inappropriately. This can be remedied by adding further conditions

to the functions. While this makes the application safer on the one hand, it increases the proof obligations on the other hand.

Note that when applying the algorithm to concrete examples, part of the knowledge may typically be computed, certain information about sets derived from $R[1,\pi]$ must be given in form of lemmas. That is, the algorithm consists of a mixture of proving and computing. It is conceivable that *Theorema* could be extended to make use of the underlying *Mathematica* system to compute the sets $B[i,\pi]$ for particular power functions π.

4 Added Value—Price Paid

In this section we summarize the added value from the point of view of a researcher in theoretical economics. The added value is partly due to the formalization effort and could have been achieved with any formal system, partly, however, it is specifically due to *Theorema* and its features. Furthermore we discuss the effort necessary to do such a formalization.

An obvious point to mention is the greater precision that a formal system requires. This is an advantage (enhanced clarity, greater reliability) and at times a disadvantage (greater effort) at the same time. A concrete example where the formal precision played a role was the characterization of the core as $K = \{x \in X \,|x_i > 0 \Leftrightarrow \pi(\{i\}, x) \geq \pi(I \setminus \{i\}, x)\}$. In this definition a free index i is used. Two standard interpretations are possible, an existentially quantified one ('there is an i such that'), or a universally quantified one ('for all i holds'). When translating into *Theorema* first the existentially quantified translation was chosen. However, this was later found to be the wrong one when it was used.

Another obvious advantage is that we can have much higher confidence in lemmas and theorems which are formally proved. In addition, the system clearly states all the knowledge used for proving a particular assertion and any hidden assumptions have to be made explicit. On the other hand, an automated theorem prover will typically be inundated with too much information so that the knowledge used to prove an assertion is typically minimal. This is useful knowledge since it allows us to generalize statements.

There are also particular advantages of the computational aspects of *Theorema*. They allow computation and checking particular structures for specific examples. For instance, for a given power function π it is possible to compute particular sets (such as $R[i,\pi]$ or $P[\pi]$) and to see whether these correspond to the intuition. If they do, this gives confidence that the formalization accurately mirrors the intuition. If they do not, then either the intuition needs to be changed or the formalization does not reflect what actually should have been formalized and needs to be changed. Of course, also incorrect statements may be discovered this way. For instance, it led to an adjustment of the algorithm in Fig. 2 in which the last case was incorrectly copied from the corresponding

lemma into it. That is, a mistake in the algorithm was detected, although no attempt was made to verify it.[10]

A particular advantage of using *Theorema* is due to the fact that some reasoners—in particular the set theory prover used mostly for our study—use an interface to the computation engine, so that proving and computing are well-integrated in the *Theorema* system. In this case study, this feature turned out to be useful when the whole proof of Lemma["s dominates t"] was shifted to just one simple computation on finite sets. In our concrete application example, we also make use of the computational parts of the algorithm in Fig. 2 to determine the stable set. The algorithm contains algorithmic parts but needs at two steps an oracle which can be given in form of lemmas. These can in turn be formally proved in *Theorema*. The possibility of *Theorema* working in a 'compute' mode makes it relatively painless to combine reasoning and computation. This makes it possible to determine the stable set for concrete power functions by a mixture of reasoning and computation.

As mentioned in the previous section it seems feasible to allow *Theorema* to move more tasks from its reasoning part into its computation part (for specialized power functions). One way to achieve this is to represent infinite sets finitely. More work in this direction is necessary.

Obviously using a system such as *Theorema* has a price. The formalization is more labour intense than formalizing the knowledge just on paper or using LaTeX. However, just writing down the definitions, lemmas, and theorems in *Theorema* is—after an introduction phase of a day—almost as painless as using LaTeX. Formalizing the knowledge is relatively easy, formally proving the lemmas and theorems, however, is typically labour intense and requires knowledge which cannot be acquired so quickly. Certain formalizations need to be changed in order to avoid certain pitfalls of the reasoners. Additionally it is necessary to know which integrated prover to use best and to adjust it by setting suitable options. This requires expert knowledge. Furthermore, it may be necessary to introduce suitable auxiliary lemmas to prove statements. This makes it currently unlikely that there will be a big uptake in using such systems, although in balance the extra work required—at least for the formal properties proved in this work—does not look too huge with one to two days of work for a formal statement. This effort may go up as the statements get more complicated as the paper proceeds, which is partly due to the fact that typically authors acquire some intuition about the concepts introduced in earlier sections, and this intuition is rarely made explicit by additional lemmas or corollaries. That is, this increased effort could be considered as a price paid by the author and an added value on the side of the reader of a formalized paper. It should be noted that just using *Theorema* for representing knowledge and making use of this knowledge, e.g. by evaluating the algorithm for concrete power functions can uncover mistakes and thereby improve the reliability of the results.

[10] As the algorithm summarizes the central results in [10], we are still in the process of verifying the proofs in paper—informally and formally.

5 Conclusion

The true benefit of a formalization can only be obtained if it is used. Typically you either prove something with it, or you use it in some computation. The latter is only possible in finite domains. In the examples, for instance, it meant that once a set is reduced to a finite set of concrete values, lemmas need not be formally proved as their truth status can be shown by exhaustive computation of all possible cases. This dual feature of computation and proving also allowed us to use the algorithm for computing the stable set for power functions for which certain features had been established (or claimed) by lemmas. Note that *Theorema* does not insist on proving assertions claimed. This makes it very easy to cite external theorems (concretely, in this example, theorems established by Jordan previously) without reproving them. While this is very convenient, it has a downside, namely it is easy to base a theorem on a lemma which is not proved or is even wrong.

A very useful feature of *Theorema* is the possibility to generate incomplete proofs. This allows both detection of potential problems with the formalizations (e.g. inconsistent argument orderings) and monitoring of whether the integrated *Theorema* prover used is making useful steps towards a solution. Moreover, the study of an incomplete proof typically helps greatly in the formulation of lemmas that then help the prover to succeed in a subsequent run, which is quite helpful since the integrated provers of *Theorema* are typically not interactive. This means, that, if they do not find a proof fully automatically, then the only possibility to guide the search is to adjust suitable options (which requires good knowledge of the inner workings of such a prover), or to introduce additional auxiliary lemmas.

As with any other theorem proving system, applying *Theorema* requires good knowledge of the system. On the positive side, the *Theorema* input language is very flexible and allows—after a very brief introduction to the system—for a natural representation which is close to a paper representation. Care should, however, be taken, since writing down statements in *Theorema* does not mean that they are correct, or can be directly used in the form they have been inputted. The input language is untyped which makes it easy to write things down. However, it also means that it is easy to introduce mistakes, e.g., *Theorema* would not complain if you defined a power function with the argument order as in powerfunction$[\pi, n]$ and later used it in form of powerfunction$[n, \pi]$. However, proofs may not be established any more in such a case.

While proving assertions formally using the *Theorema* system requires typically good knowledge of the proof in the first place, the formalization effort pays off in several ways. Above all it is possible to gain greater confidence in the correctness of the assertions, and it is possible to make experiments—concretely with specific instances of power functions—which otherwise are labour intense and error-prone.

References

1. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger, The Theorema Project: A Progress Report. In *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, M. Kerber and M. Kohlhase (eds.), pp. 98–113, A.K. Peters, Natick, Massachusetts, 2000.
2. B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, W. Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic* 4(4):470-504. 2006.
3. B. Buchberger, Theory Exploration Versus Theorem Proving. In *Proceedings of the Calculemus '99 Workshop, Electronic Notes in Theoretical Computer Science*, 23/3, Elsevier, 1999.
4. G. de Clippel and Roberto Serrano. Bargaining, coalitions and externalities: A comment on Maskin. `http://ssm.com/abstract=1304712`.
5. X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, May 1994.
6. U. Grandi, and U. Endriss. First-Order Logic Formalisation of Arrow's Theorem. *LORI'09 Proceedings of the 2nd international conference on Logic, rationality and interaction*, 2009.
7. A. Ireland, A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning* 16(1):79–111, 1995.
8. J. S. Jordan. Pillage and property. *Journal of Economic Theory*, 131(1):26–44, 2006.
9. J. S. Jordan and David Obadia. Stable Sets in Majority Pillage Games. November 2004, mimeo.
10. M. Kerber and C. Rowat. Stable Sets in Three Agent Pillage Games. June 2009, Department of Economics Discussion Paper, University of Birmingham, 09-07, `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1429326`.
11. I. Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976.
12. W. F. Lucas. A game with no solution. *Bulletin of the American Mathematical Society*, 74(2):237–239, March 1968.
13. E. Maskin, Bargaining, Coalitions and Externalities. mimeo, June 2003.
14. R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software Tools for Game Theory, Version 0.2010.09.01. mimeo, 2010 `http://www.gambit-project.org`
15. T. Nipkow. Social Choice Theory in HOL: Arrow and Gibbard-Satterthwaite. *Journal of Automated Reasoning*, 43:289–304, 2009.
16. R. Vestergaard, P. Lescanne, and H. Ono. The Inductive and Modal Proof Theory of Aumann's Theorem on Rationality. mimeo, 2006
17. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1st edition, 1944.
18. F. Wiedijk. Formalizing Arrow's Theorem. *Sādhanā*, 34(1):193–220, 2009.
19. W. Windsteiger, B. Buchberger, and M. Rosenkranz, Theorema. In *The Seventeen Provers of the World*, Freek Wiedijk (ed.), LNAI 3600:96–107, Springer, 2006.
20. W. Windsteiger, An Automated Prover for Zermelo-Fraenkel Set Theory in Theorema. *JSC* 41(3-4):435-470. Elsevier, 2006.