A Library for Algorithmic Game Theory in Ssreflect/Coq

ALEXANDER BAGNALL
SAMUEL MERTEN
and
GORDON STEWART
School of Electrical Engineering and Computer Science, Ohio University
Athens, Ohio, USA
{ab667712,sm137907,gstewart}@ohio.edu

We report on the formalization in Ssreflect/Coq of a number of concepts and results from algorithmic game theory, including potential games, smooth games, solution concepts such as Pure and Mixed Nash Equilibria, Coarse Correlated Equilibria, ϵ-approximate equilibria, and behavioral models of games such as better-response dynamics. We apply the formalization to prove Price of Stability bounds for, and convergence under better-response dynamics of, the Atomic Routing game, which has applications in computer networking. Our second application proves that Affine Congestion games are $(5/3, 1/3)$-smooth, and therefore have Price of Anarchy 5/2. Our formalization is available online.

1. INTRODUCTION

Game theory studies the interactions of self-interested parties in situations in which the actions of one party may interfere with those of another. Algorithmic game theory [25] studies games through the lenses of algorithms and theoretical computer science. For various classes of games, how tractable are the traditional solution concepts, e.g. Nash equilibria? Can we approximate such equilibria to make them more tractable? How does the cost of the worst equilibrium state compare with that of an optimal state (the Price of Anarchy for the game)? Are there subclasses of games that have bounded Price of Anarchy?

Game theory itself has proved widely relevant since Bachelier, Borel, and Zermelo in Europe and von Neumann, Nash, and Morgenstern in the United States first promulgated it in the first part of the 20th century [2, 7, 37, 24, 23, 22]. Algorithmic game theory is less venerable but seeks answers to questions that are no less relevant, especially to the application of game-theoretic models. For example, if calculating the equilibria of some game is PPAD-complete, can we expect such equilibria to be good models of an underlying game-like phenomenon?

In this paper, we report on the formalization of some recent (and not so recent) results in game theory and algorithmic game theory. These include, all in Ssreflect/Coq:

—multiplayer games;

—solution concepts such as Pure Nash Equilibria, Mixed Nash Equilibria, Coarse Correlated Equilibria and their ϵ-approximate variations;

—subclasses of games such as potential games and smooth games;

—a formalization of better-response dynamics;

—a proof that potential games converge to Pure Nash Equilibria (PNE);

—a bound on the Price of Stability of the PNE of potential games;

—a bound on the Price of Anarchy of smooth games;

—a proof that the Atomic Routing game converges under better response dynamics;

—a proof that Affine Congestion games are $(5/3, 1/3)$-smooth.

We formalized these results for two reasons. First, they are relevant – especially the results on potential and smooth games, and on Price of Stability and Anarchy – to recent developments in Algorithmic Game Theory; our formalization provides tools with which researchers could validate new results. Second, the authors are working in parallel on applications of some results from this paper to the design and proof of game-theoretic models of distributed systems, e.g. distributed network routers, which we reported on recently in a brief announcement at PODC [3]. We believe that the results in this paper – which focuses on formalization-related aspects of the underlying Ssreflect/Coq libraries – are of independent interest from the work described briefly in [3].

Our formalization is available online at:

`https://github.com/gstew5/games.`

2. RELATED WORK

Games in Formal Cryptography. Barthe and colleagues have published extensively on formal verification (in CertiCrypt) of cryptographic protocols such as encryption [5] and signature schemes [36]. In the cryptographic setting, such protocols can be expressed as games against a (typically computationally bounded) adversary. The CertiCrypt model deeply embeds games via a probabilistic programming language, pWHILE, with an associated relational Hoare logic. This deep embedding facilitates the definition of program refinements, which are used to prove bounds on, e.g., an adversary's advantage against a particular encryption scheme. Other researchers, such as Nowak [26], have used shallow embeddings of games to formalize similar cryptographic proofs to those in CertiCrypt. The shallow-embedding style more closely matches the definitions we use in this paper.

Formalized Mechanism Design. Perhaps more relevant to this article are recent results in the formalization of protocols from mechanism design, a field closely related to algorithmic game theory. Barthe and his co-authors have done pioneering work in this area as well, e.g., [4]. One of the main goals of such work is to formally prove that mechanisms such as those used in auction design incentivize participants to faithfully report their preferences (so-called *truthfulness* properties). For example, in [4], Barthe et al. verify the truthfulness of the random sampling auction of Goldberg et al. [10]. A secondary contribution of [4] was to formally prove the correctness of a mechanism for computing approximate Nash equilibria of aggregative games [9]. This mechanism plays a role similar to that of the better-response dynamics we formalize in Section 7. Our work is complementary in the sense that we provide a unified library for proving results in algorithmic game theory which could be used to prove additional results in mechanism design. As we outline in Section 1, we also prove a number of results that – to the best of our knowledge – have not yet been mechanized, such as the facts that Atomic Routing games converge under better-response dynamics, and that Affine Congestion games are (5/3, 1/3)-smooth.

Game Theory Formalized. A few researchers have previously mechanized results from game theory in theorem provers such as Coq and Isabelle [16, 18, 17, 34, 32, 15]. Lescanne [16], for example, reports on a library of extensive games in Coq, in which (potentially infinite) games are represented as a Coq co-inductive type. Our library is limited to finite games (the set of player strategies is finite) but includes a number of results from algorithmic game theory that do not appear in [16]. In 2006, Vestergaard [34] reported on an earlier mechanization of game theory in Coq in which he proved via backward induction that finite sequential games (also represented in extensive form, this time as an inductive rather than co-inductive type) have Nash equilibria. More recently, Le Roux [14, 15] generalized Vestergaard's result, which was limited to binary games with natural-valued payoff functions, to arbitrary finite games with acyclic preference relations over abstract outcomes.

3. BACKGROUND

3.1 Game Theory

Game theory studies the design and analysis of systems of mutually competitive actors: a set of N players, each attempting to minimize their individual costs wrt. some cost function C over an action space A.[1] The type A might be indexed by the player number $i \in [0, N)$ (as in A_i) to allow each player to specialize its action space to a particular type.

The overall state after a round of multiplayer play is an N-tuple of actions $(a_1, a_2, \cdots, a_N)$, where the action of player i is drawn from A_i. The cost C_i to player i is calculated wrt. the tuple $(a_1, \cdots, a_i, \cdots, a_N)$ and can be understood as the cost to i of its chosen action (a_i) wrt. the actions of the other $N-1$ players. A state $(a_1, \cdots, a_i, \cdots, a_N)$ is a *Pure Nash Equilibrium (PNE)* [23] if for every i and potential deviant action a_i',

$$C_i(a_1, \cdots, a_i, \cdots, a_N) \leq C_i(a_1, \cdots, a_i', \cdots, a_N)$$

The notion of PNE generalizes to situations in which players may randomize over their actions (Mixed Nash Equilibrium) and to situations in which the players'

[1] We use a cost-minimization formulation of games. However, everything we present in this paper could be suitably dualized to formulate games in payoff-maximization style.

distributions over actions may be correlated (Correlated Equilibrium). An even broader generalization, called Coarse Correlated Equilibrium (CCE), classifies those distributions σ over states a such that

$$\mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, a_i, \cdots, a_N)] \leq \mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, a'_i, \cdots, a_N)]$$

for all i and a'_i. The expected cost to player i in σ of $a = (a_1, \cdots, a_i, \cdots, a_N)$ is less than or equal to the expected cost of $(a_1, \cdots, a'_i, \cdots, a_N)$.

All the equilibrium notions above have approximate counterparts. For example, ϵ-approximate CCEs are those distributions σ such that

$$\mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, a_i, \cdots, a_N)] \leq \mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, a'_i, \cdots, a_N)] + \epsilon$$

Player i can gain (in expectation) at most ϵ by deviating to a'_i.

3.2 Algorithmic Game Theory

Algorithmic game theory (AGT) [25] applies traditional computer science techniques such as algorithm analysis to the study of games. A number of recent AGT results [28, 6, 1, 19, 29, 31, 27] have sought to bound the degree to which the solutions of particular games approximate socially optimal solutions to problems such as network routing, the so-called *Price of Anarchy (POA)* [13] of the game. By socially optimal, we mean states of the game that minimize some objective function such as the sum of all player costs. The POA of a game is the ratio of the cost of the worst equilibrium state to the cost of a socially optimal solution.

More informally, POA quantifies the loss of efficiency one pays by allowing mutually competitive agents to selfishly calculate an equilibrium or solution state for the game, wrt. an optimal (perhaps centrally coordinated) solution. The POA for some classes of games can be quite small. For example, affine congestion games, which can be used to model network routing, have POA 5/2 [8]. Other classes of games (e.g., facility location [35]) also have low POA.

Related to POA is *Price of Stability (POS)*, the ratio of the cost of the *best* equilibrium state to that of an optimal state. POA and POS are equal in games with only one equilibrium state.

3.3 Game Dynamics

By a game *dynamics*, we mean a model of the strategy used by the players of the game to choose their actions over the course of iterated play.

One such strategy is *better response*: In each round, a player may move from current action a to new action a' only if the cost of a', wrt. the actions of other players, is less than the cost of a (each move by player i reduces player i's cost). For certain classes of games, e.g. potential games [21], better-response dynamics leads naturally to Nash equilibria, as we prove formally in Section 7. Other strategies, such as no-regret dynamics, drive all games to the wider solution class of ϵ-approximate CCEs. [30, Chapter 17]

3.4 Ssreflect/Coq

We use Ssreflect [11] libraries throughout our formalization. For readers more familiar with standard Coq or with another theorem prover, we briefly summarize some of the definitions and notation we use most heavily:

Finite Types. Ssreflect models finite types (notation A : finType) as pairs of the type A and an enumerator enum : list A. The enumerator satisfies the property:

$\forall a : A.$ count a enum $= 1$.

In the enumeration of the values of type A, every element is included exactly once.

Finite Functions. Ssreflect models functions with finite domain:

{ffun $A \to B$}

as tuples of values of type B, of size $|A|$, where $|A|$ is the cardinality of the finite type A. The cardinality of a finite type is naturally defined as the length of its enumeration, which works because the enumeration is defined to include each element in the type exactly once.

Bounded Naturals. One useful finite type which we use widely is the set of naturals $[0 \dots N)$ between 0 and N exclusive, for a particular bound N. Ssreflect's syntax for this type is 'I_N. To clarify in code listings, we often replace 'I_N with the slightly less cumbersome syntax $[N]$. For example, the type of finite functions mapping integers in the range $[0 \dots N)$ to values of type A has type:

{ffun 'I_$N \to A$}

or in the notation which we use in this paper:

{ffun $[N] \to A$}.

4. GAMES IN SSREFLECT/COQ

Ssreflect uses packed classes and canonical structures [20] to construct type hierarchies. We use Ssreflect's numeric hierarchy and other Ssreflect types in packed-class form, but depart from packed classes to operational type classes [33] when defining new types (aside from the discrete distributions of Section 5). Operational type classes facilitate parameter sharing, e.g., in the definition of combinators.[2] As an example of one such typeclass hierarchy, consider our encoding of games:

Class CostClass (N : nat) ($\mathbb{R}$: realFieldType) (A : finType) $\triangleq$
  cost_fun : $[N] \to$ {ffun $[N] \to A$} $\to \mathbb{R}$.
Notation “'cost'” $\triangleq$ (@cost_fun _ _ _) (at level 30).
Class CostAxiomClass N $\mathbb{R}$ A ‘(CostClass N $\mathbb{R}$ A) $\triangleq$
  cost_axiom (i : $[N]$) (f : {ffun $[N] \to A$}) : $0 \leq$ cost i f.
Section costLemmas. Context {N $\mathbb{R}$ A} ‘(CostAxiomClass N $\mathbb{R}$ A).
  Lemma cost_nonneg i f : $0 \leq$ cost i f. Proof. apply: cost_axiom. $\mathbb{Q}$ed.
End costLemmas.
Class MovesClass (N : nat) (A : finType) $\triangleq$ moves_fun : $[N] \to$ rel A.
Notation “'moves'” $\triangleq$ (@moves_fun _ _) (at level 50).

[2]While we do not use such combinators in this article, they are nevertheless quite useful when defining, e.g., domain-specific languages for the combinatorial construction of games, as we have done in the work described in [3].

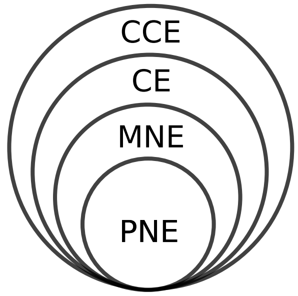

Fig. 1. Equilibrium Refinements. Illustration modeled on [30, Figure 13.1].

Class game (A : finType) (N : nat) ($\mathbb{R}$: realFieldType)
'(costAxiomClass : CostAxiomClass N $\mathbb{R}$ A)
(movesClass : MovesClass N T) : Type $\triangleq$ {}.

The operational type class CostClass N $\mathbb{R}$ A – in a game with N players and action space A – asserts the existence of an $\mathbb{R}$-valued cost function cost_fun (notation cost) mapping a player index of type $[N]$ and state of type $\{$ffun $[N] \to A\}$ (assigning an action of type A to each player) to a real-valued cost of type $\mathbb{R}$. The class costAxiomClass ensures that cost is nonnegative. Games in our formulation are finite, a constraint enforced by the fact that the type of game states A is itself a finType. We often use the phrase "game A" to refer metonymously to the entire game over type A (including its other defining components such as the cost function).

The game typeclass packages the cost function with a second class, MovesClass, that defines the game's allowable moves. For example, although a game operates over a single action type A, we can implement indexed action types A_i (in which each player has its own action space) through a combination of

—dependent pairs $\Sigma_i : [N].\ A_i$, for some function A : $\{$ffun $[N] \to$ Type$\}$, and
—an auxiliary moves relation $\lambda a\, a'.\ \pi_1\ a = \pi_2\ a'$ enforcing that players leave unchanged the i that indexes the type of the second part of each player's strategy.

We use this construction in Section 8.2 to index the player actions in an Atomic Routing game by their respective network sources and sinks.

An alternative to the single-type-with-Move-constraints strategy is to allow each player in the game to specify its own type A_i, dependent on the player's index i. This formulation of games, as generalized in [32] to support abstract agents and observations, builds the dependency of strategy types on players into the definition of games itself. It therefore does not require an auxiliary Moves relation as we impose above. But it is also less straightforward, under the dependently-typed formulation, to construct a typeclass hierarchy indexed by the single type of strategies A associated with a particular game, as we do in [3] to build a library of combinators over smooth games.

5. SOLUTION CONCEPTS

Game theory is concerned as much with the definition of equilibrium notions – the solutions of games – as it is with the games themselves. In this section, we present our formalization of key equilibrium notions from the equilibrium hierarchy in Figure 1, following Roughgarden [30, Chapter 13]. To simplify some of the definitions, we use the following short-hand in a **Section** context parameterized by the number of players N and the action type A:

Definition state N $A \triangleq$ ({ffun $[N] \to A$})%**type**.

We first consider Pure Nash Equilibria, the smallest equilibrium notion in Figure 1.

Definition PNE (t : state N A) : Prop $\triangleq$
  $\forall (i : [N])\ (t'_i : A)$,
  moves i $(t\ i)$ $t'_i \to$ cost i $t \leq$ cost i (upd i t t'_i).

In the definition of PNE, the relation moves i $(t\ i)$ t'_i asserts that player i's move is valid with respect to the current game's MovesClass. The function upd i t t'_i returns the new state t' for which $t'\ i = t'_i$ and $t'\ j = t\ j$ for all $i \neq j$.

Because the event space A is finite, the predicate PNE is decidable, which we express by defining the following boolean version of PNE:

Definition PNEb (t : state N A) : bool $\triangleq$
[$\forall$ i : $[N]$,
  [$\forall$ t' : state N A, Move i t t' $\Longrightarrow$ (cost i $t \leq$ cost i t')]].

along with the following reflection lemma:

Lemma PNEP t : reflect (PNE t) (PNEb t).

In the definition of PNEb above, we use features from Ssreflect such as the boolean-valued enumeration $[\forall i : [N], \ldots]$ of a boolean-valued predicate over a finite domain, as well as Ssreflect's boolean implication ($\Longrightarrow$) and equality ($==$) operators.

Discrete Distributions. The larger equilibrium classes of Figure 1 (MNE, CE, and CCE) are probabilistic rather than deterministic. To define these equilibria, we first define discrete distributions (over finite event spaces A) as follows:

Section Dist.
  Variable A : finType.
  Variable $\mathbb{R}$: realFieldType.
  Definition dist_axiom (f : {ffun $A \to \mathbb{R}$}) : bool $\triangleq$
    [&& $\sum_a (f\ a) == 1$
      & [$\forall$ a : A, f $a \geq 0$]].
  Record dist : Type $\triangleq$
    mkDist { pmf :> {ffun $A \to \mathbb{R}$}; dist_ax : dist_axiom pmf }.
  (∗ ... canonical projections ... ∗)
End Dist.

We represent discrete distributions as finite probability mass functions pmf (type {ffun $A \to \mathbb{R}$}) that map values in the event space A to their weights in $\mathbb{R}$. To ensure that pmfs are well-formed probability distributions, we impose axioms (dist_axiom) asserting that the pmf (1) sum to 1 and (2) be nonnegative. Elided are a few canonical projections which ensure that distributions inherit structures (e.g., decidable equality) from the pmf projection.[3]

Standard definitions like the expected value of a discrete random variable are easily given with respect to the formulation of distributions above. We do so within a section that is parameterized by A, $\mathbb{R}$, and a particular distribution d.

Section expectedValue.
  Variable A : finType.
  Variable $\mathbb{R}$: numDomainType.
  Variable d : dist A $\mathbb{R}$.

We define expected value as the specialization of an auxiliary function, expectedCondValue, to the constant predicate predT $= (\lambda\_ \Rightarrow$ true$)$

  Definition expectedCondValue (f : $A \to \mathbb{R}$) (p : pred A) $\triangleq$
    $(\sum_{\_}(t : A \mid \mathsf{p}\ t)\ (\mathsf{d}\ t * \mathsf{f}\ t))\ /\ (\sum_{\_}(t : A \mid \mathsf{p}\ t)\ \mathsf{d}\ t)$.
  Definition expectedValue (f : $A \to \mathbb{R}$) $\triangleq$ expectedCondValue f predT.

where expectedCondValue takes the sum over only those values of A that satisfy the predicate p, divided by the probability in d that p occurs. In our development, we prove some useful facts about expectedValue such as:

  Lemma expectedValue_linear f g :
    expectedValue $(\lambda\ t \Rightarrow \mathsf{f}\ t + \mathsf{g}\ t)$ = expectedValue f + expectedValue g.
  Lemma expectedValue_mull f c :
    expectedValue $(\lambda\ t \Rightarrow \mathsf{c} * \mathsf{f}\ t)$ = c $*$ expectedValue f.
  Lemma expectedValue_const c : expectedValue $(\lambda\ \_ \Rightarrow \mathsf{c})$ = c.
  Lemma expectedValue_range f :
    $(\forall\ t : A,\ 0 \leq \mathsf{f}\ t \leq 1) \to 0 \leq$ expectedValue f ≤ 1.
  (* . . . *)
End expectedValue.

Formulating basic and derived distributions is also straightforward. For example, here is the uniform distribution, which we define within a section parameterized by the event type A and an element t_0 of type A.

Section uniform.
  Variable A : finType.
  Variable t_0 : A.
  Definition uniform_dist : {ffun $A \to$ rat} $\triangleq$
    finfun $(\lambda\ \_ \Rightarrow 1\ /\ \#|A|\%:R)$.
  Lemma uniform_normalized : dist_axiom uniform_dist.
  Definition uniformDist : dist A [numDomainType **of** rat] $\triangleq$

[3]There are alternative ways to model distributions within a theorem prover. For example, one could formalize the theory of measurable spaces, on top of which probability spaces are measurable spaces with measure 1. The measure-theory formulation would extend to continuous distributions but introduces needless complexity wrt. our discrete (decidable) distributions over finite-strategy games. Computable distributions, as applied to programming language semantics by Huang and Morrisett [12], are perhaps a promising middle ground for future consideration.

```
    mkDist uniform_normalized.
  Lemma expectedValue_uniform (f : A → rat) :
    expectedValue uniformDist f = (∑_(t : A) (f t)) / #|A|%:R.
End uniform.
```

The element $t_0 : A$ ensures that the cardinality $\#|A|$ of type A is greater than 0, a fact necessary to prove that the division 1 / #|A|%:R is well defined.

We define product distributions, which are used to define Mixed Nash Equilibria, as follows:

```
Section product.
  Variable A : finType.
  Variable ℝ : numDomainType.
  Variable N : nat.
  Variable f : {ffun [N] → dist A ℝ}.
  Notation type ≜ ({ffun [N] → A}).
  Definition prod_pmf : {ffun type → ℝ} ≜
    finfun (λ p : type ⇒ ∏_(i : [N]) f i (p i)).
  Lemma prod_pmf_dist : dist_axiom (A ≜ [finType of type]) (rty ≜ ℝ) prod_pmf.
  Definition prod_dist : dist [finType of type] ℝ ≜
    mkDist prod_pmf prod_pmf_dist.
End product.
```

We assume N distributions, given by the finite function f : {ffun $[N]$ → dist A $\mathbb{R}$} mapping indices in the range 0 to $N-1$ to distributions over A. The event space of the product distribution is the type of N-tuples over A, which we represent as finite functions of **type** ≜ ({ffun $[N]$ → A}).

MNEs, CEs, CCEs, and Approximations. We work backward to build MNEs, CEs, and CCEs (the most general class in Figure 1), by first defining ϵ-approximate CCEs, and then specializing $\epsilon = 0$ to yield nonapproximate CCEs. CEs are a refinement of ϵ-approximate CEs, which are themselves a subset of ϵ-approximate CCEs. MNEs specialize CEs to the case in which the distribution over actions is a product distribution (the players' mixed strategies are independent).

We define the most general class, ϵ-CCEs, as follows (N, A, and $\mathbb{R}$ are section parameters):

```
Definition eCCE (ϵ : ℝ) (d : dist [finType of state N A] ℝ) : Prop ≜
  ∀(i : [N]) (t'_i : A),
    (∀ t : state N A, t ∈ support d → moves i (t i) t'_i) →
    expectedCost i d ≤ expectedUnilateralCost i d t'_i + ϵ.
```

Player i can gain at most ϵ by making a unilateral move from distribution d to action t'_i. The support of d is the set of values $t : A$ with nonzero probability (0 < d t). The expectedCost to player i in distribution d is simply the expected value in d of the cost to player i:

```
Definition expectedCost (i : [N]) (d : dist [finType of state N A] ℝ) ≜
    expectedValue d (cost i).
```

Here cost is the cost function associated with the game over type A.

The function expectedUnilateralCost gives the expected value, to player i, of a unilateral move by i to action t'_i:

Definition expectedUnilateralCost
$(i : [N])$ $(d :$ dist [finType **of** state N A] $\mathbb{R})$ $(t'_i : A) \triangleq$
expectedValue d $(\lambda$ t : state N $A \Rightarrow$ cost i (upd i t t'_i)).

The function upd is the same as that used to define PNE above. Nonapproximate CCEs specialize ϵ-CCEs to $\epsilon = 0$:

Definition CCE $(d :$ dist [finType **of** state N A] $\mathbb{R})$: Prop $\triangleq$ eCCE 0 d.

and thus are trivially also eCCEs.

Correlated equilibria are distributions σ over states a such that

$$\mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, b_i, \cdots, a_N)] \le \mathbb{E}_{a\sim\sigma}[C_i(a_1, \cdots, b'_i, \cdots, a_N)]$$

for all i, b_i, and b'_i. That is, player i's calculation is conditioned on the fact $a_i = b_i$ (the realization of player i's action in state a drawn from σ is known). For completeness, we formalize CEs in our development but do not use them much, except to define Mixed Nash Equilibria (MNEs) as those CEs in which σ is a product distribution over the players' strategies and to prove that every CE is a CCE. Thus every MNE is a CCE as well, validating two more of the inclusion relationships in Figure 1.

5.1 Efficiency of Equilibria

Equilibria are most useful if it is possible to quantify – for a given game or class of games – the quality of that class of game's equilibria with respect to some objective function. Two commonly used measures, as we outlined in Section 3, are Price of Anarchy (POA) and Price of Stability (POS). POA calculates the ratio of the cost of the worst equilibrium state, with respect to an objective function (typically the sum of player costs), to that of an optimal state. POS calculates the ratio of the cost of the best equilibrium state to that of an optimal state. POA helps to quantify the quality of the equilibria of a game – which is especially useful in combination with procedures that calculate such equilibria. POS bounds, while weaker than POA bounds, can be useful when games have just a single equilibrium state (in which case POS and POA coincide) or in, e.g., network routing games, in which a central authority may propose the best rather than worst equilibrium network route plan (cf. [25, Chapter 17]).

Price of Anarchy. In our formal development, we define POA as:

Definition POA : $\mathbb{R} \triangleq$
Cost (arg_max PNEb Cost t_0) / Cost (arg_min predT Cost t_0).

This definition's main ingredients are:

—The objective function

Definition Cost $(t :$ state N $A) : \mathbb{R} \triangleq \sum\_i$ (cost i t).

which sums the per-player costs cost i t of state t in the context of game A;

—An optimal state of game A, defined as:

arg_min predT Cost t_0

satisfies optimality as given by the following predicate over states:

Definition optimal : pred (state N A) $\triangleq \lambda t \Rightarrow$ [$\forall$ t', Cost $t \leq$ Cost t'].

The function arg_min (P : pred I) ($F : I \to \mathbb{R}$) ($i_0 : I$) – with respect to some finite type I, a predicate P over I, and a valuation function $F : I \to \mathbb{R}$ – returns an $i : I$ that minimizes F restricted to P. We supply a default value t_0 : state N A to arg_min to ensure that state N A is inhabited, and arg_min predT is therefore total (because all t_0 satisfy the top predicate predT).

—The maximum-cost Pure Nash Equilibrium state

arg_max PNEb Cost t_0,

a state of type state N A that maximizes the cost function Cost restricted to PNEb.

To define POA as a computable (boolean- rather than Prop-valued) function, it is important that the auxiliary predicates used above – optimal and PNEb – are themselves computable. To use Ssreflect's boolean quantification (e.g., [$\forall t'$, Cost $t \leq$ Cost t'], returning a boolean), we must also know that states state N A are finite. To prove, e.g., that the state arg_max PNEb Cost t_0 is a Pure Nash Equilibrium, it is necessary to show that game A has at least one PNE (for example, by proving that the default state t_0 is a PNE).

Price of Stability. Our formal definition of Price of Stability (POS) is quite similar to POA:

Definition POS : $\mathbb{R}$ $\triangleq$
Cost (arg_min PNEb Cost t_0) / Cost (arg_min predT Cost t_0).

the main difference being that the numerator of the ratio is now the cost of the minimum-cost PNE rather than the maximum-cost PNE.

As one might expect, it is straightforward to prove that for every game with nonnegative cost functions, POS is always less than or equal than POA:

Lemma POS_le_POA
(has_PNE : PNEb t_0) : POS $\leq$ POA.

In order for POS and POA to be defined, we must assume that game A has at least one PNE (has_PNE : PNEb t_0). The cost function for the game must also be nonnegative, a constraint satisfied by the game's CostAxiomClass instance.

All the definitions in this section easily dualize to a payoff-maximization formulation of games (for example, by requiring negative cost functions and by switching the directions of various inequalities).

6. GAME SUBCLASSES

Some games, such as the potential and smooth games that we formalize in this section, have bounds on either POS or POA or both. Such bounds are most useful in connection with models of the dynamics of games (Section 7), which define the conditions under which a particular game converges to equilibrium (assuming equilibria exist).

6.1 Potential Games

Potential games are those for which there exists a potential function Φ – mapping game states to $\mathbb{R}$ – such that

$$\forall i\ t\ t'_i.\ \ \mathsf{let}\ t' \triangleq \mathsf{upd}\ i\ t\ t'_i\ \mathsf{in}$$
$$\Phi(t') - \Phi(t) = \mathsf{cost}_i(t') - \mathsf{cost}_i(t).$$

For any unilateral deviation by some player i from action $t\ i$ to t'_i, the potential function Φ exactly captures the cost difference incurred by i from the deviation (t' is the state that updates player i's strategy from t_i to t'_i but is otherwise equal t). Potential games are guaranteed to have at least one PNE (a state that minimizes the potential function Φ) and are guaranteed to converge to equilibrium under better-response dynamics, a fact we prove formally in Section 7.

We formalize potential games using operational type classes, just as we did the (unqualified) games of Section 4. We first define an operational type class for the potential function itself:

```
Class PhiClass (N : nat) (ℝ : realFieldType) (A : finType)
    `(costAxiomClass : CostAxiomClass N ℝ A)
    (movesClass : MovesClass N A) : Type ≜
  Phi : state N A → ℝ.
```

and then a type class for the potential axiom:

```
Class PhiAxiomClass (N : nat) (ℝ : realFieldType) (A : finType)
    `(costAxiomClass : CostAxiomClass N ℝ A)
    (movesClass : MovesClass N A)
    (phiClass : PhiClass costAxiomClass movesClass) : Type ≜
  PhiAxiom :
    ∀(i : [N]) (t : state [N] A) (t'_i : A),
      moves i (t i) t'_i →
      let t' ≜ upd i t t'_i in
      Phi t' − Phi t = cost i t' − cost i t.
```

The main difference in PhiAxiom from the mathematical definition of potential games above is that we assume, additionally, that players are limited to moves allowed by the game's moves relation: moves $i\ (t\ i)\ t'_i$. Stated another way, the Φ function need be exact only with respect to action updates permitted by moves. The moves hypothesis can always be made vacuous by constructing a game in which moves is the constant relation $\lambda\_\ \_ \Rightarrow$ true (in which case we get the standard definition of potential games).

In a context in which we assume the type A together with its associated cost and moves functions define a potential game, we then prove a number of facts, such as:

```
Theorem exists_PNE (t_0 : state N A) : ∃t : state N A, PNE t.
```

Every potential game with at least one action (or equivalently, at least one state) has at least one Pure Nash Equilibrium.

The structure of this proof is as follows. First, call minimal those states that minimize the potential function Φ:

Definition minimal : pred (state N A) $\triangleq$
[pred t : state N A | [$\forall$ t' : state N A, Phi $t \leq$ Phi t']].

Any state minimal wrt. the potential function is a PNE, because Φ exactly tracks the cost function of the game:

Lemma minimal_PNE (t : state N A) : minimal $t \to$ PNE t.

We formalize this intuition in the following lemma about the relation of Φ and game A's cost function:

Lemma Phi_cost_le (t : state N A) i (t'_i : A) :
moves i (t i) $t'_i \to$
let $t' \triangleq$ upd i t t'_i **in**
Phi $t \leq$ Phi $t' \to$ cost i $t \leq$ cost i t'.

If Φ increases after a unilateral move by player i, than so does the cost to player i.

From **Lemma** minimal_PNE, it is straightforward to prove that at least one PNE exists, by exhibiting a state t that minimizes Φ:

Definition Phi_minimizer (t_0 : state N A) : state N $A \triangleq$ arg_min predT Phi t_0.

In order to build Phi_minimizer, we must first ensure that at least one state exists (t_0 : state N A).

6.1.0.1 *Price of Stability Bound.* Every potential game has bounded Price of Stability assuming there exist values α and β such that α is greater than 0:

Hypothesis (HAgt0 : 0 $< \alpha$)

and for any state t, Φ t is bounded below by $1/\alpha$ times the cost of t and above by β times the cost of t:

Hypothesis AB_bound_Phi :
$\forall t$: state N A, Cost t / $\alpha \leq$ Phi $t \leq \beta *$ Cost t.

The proof that POS is bounded by $\alpha\beta$:

Lemma POS_bounded (t : state N A) (PNE_t : PNE t) : POS $t \leq \alpha * \beta$.

then follows from the following series of inequalities, letting t^* be a state with optimal cost and t^Φ a state that minimizes the potential function:

$$\begin{aligned}
\text{Cost (arg_min PNEb Cost } t_0) &\leq \text{Cost } t^\Phi && (1)\\
&\leq \alpha \cdot \text{Phi } t^\Phi && (2)\\
&\leq \alpha \cdot \text{Phi } t^* && (3)\\
&\leq \alpha\beta \cdot \text{Cost } t^* && (4)
\end{aligned}$$

where inequalities (2) and (4) follow from AB_bound_Phi, (3) follows from the fact that t^Φ minimizes Phi, and (1) from **Lemma** minimal_PNE.

6.2 Smooth Games

Potential games are guaranteed to have Pure Nash Equilibria. However, the existence of a potential function does not in itself imply any particular bounds on the cost of such PNEs with respect to the optimal cost of the game.

Smooth games – a class of games first described by Roughgarden [28] that is distinct from potential games – are guaranteed, by contrast, to have POA bounds that quantify the quality of the equilibria of the games. How good such POA bounds are depends on two technical parameters, called λ and μ: as Rougharden shows in [28, Section 2.1], a game that is (λ,μ)-smooth has POA $\frac{\lambda}{1-\mu}$, by a simple generic argument.

Smoothness was motivated in part by Roughgarden's desire to encapsulate in a single condition the essence of proofs of POA for disparate games such as routing and location games (cf. [30, Section 14.3]). However, games that are smooth also exhibit a number of nice properties, such as POA bounds that extend not just to PNEs but even to CCEs, the largest equilibrium class of Figure 1.

We say a game is (λ,μ)-smooth if for every two states t and t^*, the following condition holds:

$\sum_{i=1}^{N} \mathsf{cost}\ i\ (t_1,\ \ldots,\ t_i^*,\ \ldots,\ t_N) \leq \lambda \cdot \mathsf{Cost}\ t^* + \mu \cdot \mathsf{Cost}\ t$

The overall cost of the "mixed" state in which we consider the cost to each player i of a unilateral deviation from t_i to t_i^* is bounded above by λ times the cost of the new state t^* plus μ times the cost of the previous state t. For intuition, think of t^* as a possible optimal state to which the game might move from t. The λ parameter, which is typically greater than or equal to 1, relates the mixed state on the left to the "optimal" deviation t^*. The μ parameter, which should be greater than or equal to 0 and strictly less than 1, relates the mixed state to the previous state t before any player has moved to t^*.

Assuming that game A is (λ,μ)-smooth, one can show (cf. [28, Section 2.1] or [30, Section 14.4.1]) that the game's Pure Nash Equilibria have POA $\frac{\lambda}{1-\mu}$ by the following derivation:

$$\begin{aligned}
\mathsf{Cost}\ t &= \sum_{i=1}^{N} \mathsf{cost}\ i\ t && (5)\\
&\leq \sum_{i=1}^{N} \mathsf{cost}\ i\ (t_1, \ldots, t_i^*, \ldots, t_N) && (6)\\
&\leq \lambda \cdot \mathsf{Cost}\ t^* \ +\ \mu \cdot \mathsf{Cost}\ t && (7)\\
&\leq \frac{\lambda}{1-\mu} \cdot \mathsf{Cost}\ t^* && (8)
\end{aligned}$$

Inequality 6 follows from the fact that t is assumed a PNE. Inequality 7 follows from the smoothness condition. Inequality 8, which establishes the POA bound, follows from 7 by rearranging terms.

We formalize smooth games just as we did potential games, via a series of type class declarations:

```
(* Operational type class for λ *)
Class LambdaClass (A : finType) (ℝ : realFieldType) : Type ≜ lambda_val : ℝ.
Notation ‘’’lambda’ ’of’ A’’ ≜ (@lambda_val A _ _) (at level 30).
Class LambdaAxiomClass (A : finType) (ℝ : realFieldType) ‘(LambdaClass A ℝ)
  : Type ≜ lambda_axiom : 0 ≤ lambda of A.

(* Operational type class for μ *)
Class MuClass (A : finType) (ℝ : realFieldType) : Type ≜ mu_val : ℝ.
Notation ‘’’mu’ ’of’ A’’ ≜ (@mu_val A _ _) (at level 30).
Class MuAxiomClass (A : finType) (ℝ : realFieldType) ‘(MuClass A ℝ)
  : Type ≜ mu_axiom : 0 ≤ mu of A < 1.

(* Smooth games *)
Class SmoothnessAxiomClass (N : nat) (ℝ : realFieldType) (A : finType)
  ‘(costAxiomInstance : CostAxiomClass N ℝ A)
  (movesInstance : MovesClass N A)
  (gameInstance : game costAxiomInstance movesInstance)
  ‘(lambdaAxiomInstance : LambdaAxiomClass A ℝ)
  ‘(muAxiomInstance : MuAxiomClass A ℝ) : Type ≜
  SmoothnessAxiom :
    ∀t t′ : {ffun [N] → A},
    valid_Move t t′ →
    ∑_(i : [N]) cost i (upd i t (t′ i)) ≤
    lambda of A * Cost t′ + mu of A * Cost t.
Notation ‘’’smooth_ax’’’ ≜
  (@SmoothnessAxiom _ _ _ _ _ _ _ _ _).
```

As was true for potential games above, our formalized smoothness condition matches the mathematical definition of smoothness except for the additional moves-related condition valid_Move t t', which is defined as $\forall i$: $[N]$, moves i $(t\ i)$ $(t'\ i)$ (each per-player move from t to t' is valid wrt. the moves relation of the game).

For a game to be (λ,μ)-smooth, it must also be the case that $0 \le \lambda$ (LambdaAxiomClass) and $0 \le \mu < 1$ (MuAxiomClass).

Generically for all smooth games, we have proved results such as:

```
Lemma smooth_PNE_POA (t t′ : {ffun [N] → A}) :
  PNE t →
  valid_Move t t′ →
  Cost t ≤ (lambda of A / (1 − mu of A)) * Cost t′.
```

By instantiating t' to the maximum-cost PNE of game A, smooth_PNE_POA implies the $\frac{\lambda}{1-\mu}$ POA bound we derived above.

One advantage of smooth games, however, is that such POA bounds are robust: they extend even to Coarse Correlated Equilibria (as well as to Correlated Equilibria and Mixed Nash Equilibria). For instance, the following lemma proves that any distribution d that is a CCE for game A has expected cost less than $\frac{\lambda}{1-\mu}$ times that of any (potentially optimal state) t':

```
Lemma smooth_CCE (d : dist [finType of state N A] ℝ) (t' : state N A) :
  CCE d →
  dist_valid_Move d t' →
  ExpectedCost d ≤ (lambda of A / (1 − mu of A)) ∗ Cost t'.
```

The conclusion of this lemma generalizes POA to CCEs, which predicate over state distributions rather than states as do PNEs. ExpectedCost is defined as the sum of the individual player costs $\sum_{\_}(i : [N])$ expectedCost i d, which is equivalent to the expected total cost by linearity of expectation.

7. DYNAMICS

It is fruitless to prove bounds on the quality of equilibria of games if such games never reach equilibrium states in the first place. In this section, we formalize dynamics, or operational semantics, for the games of the previous section – defining the behavior of the games under iterated play by multiple agents. Although our model of game dynamics is modular, we focus here on the specialization to better-response dynamics [25, Section 1.4.3], which require that players take only those moves that either minimize or decrease at each step their individual (expected) costs with respect to the actions chosen by other players. Our general operational semantics for games is parameterized by a step relation, step, and a predicate, halted, that specifies when an execution has safely ended.

```
Section stepDefs.
  Context {A : Type}.
  Variable step : A → A → Prop.
  Variable halted : A → Prop.
  Hypothesis haltedP : ∀t t' : A, halted t → step t t' → False.
  (∗ . . . ∗)
```

Hypothesis haltedP relates halted and step by asserting that halted states cannot execute further. As one instantiation of the step relation, consider the following definition of a version of better-response dynamics:

```
Inductive better_response_step N A : {ffun [N] → A} → {ffun [N] → A} → Prop ≜
| better_response_step_progress t (i : [N]) t'_i :
    moves i (t i) t'_i →
    let t' ≜ upd i t t'_i in
    cost i t' < cost i t →
    better_response_step t t'.
```

which states that a step from state t to t' is allowed only if it strictly reduces some player i's cost (and satisfies the game's moves relation).

We define the reflexive transitive closure of the step relation as the following fixed point on the number of steps n:

```
Fixpoint stepN (n : nat) : A → A → Prop ≜
[λ t t' ⇒
  if n is S n' then
    ∃t'', [∧ step t t'' & stepN n' t'' t']
  else t = t'].
```

This fixpoint definition of the closure of step is equivalent to the more standard inductive characterization, which is also useful:

Inductive step_star : $A \to A \to$ Prop $\triangleq$
| step_refl t : step_star t t
| step_trans t'' t t' :
  step t t'' $\to$
  step_star t'' t' $\to$
  step_star t t'.
Lemma stepN_step_star t t' : ($\exists$ n, stepN n t t') $\leftrightarrow$ step_star t t'.

We say a state $t : A$ is safe, as is standard, if

Definition safe t $\triangleq$
 $\forall t''$, step_star t t'' $\to$
  [$\vee$ $\exists t'$, step t'' t' | halted t''].

Any state we can reach from t can either take a step or is halted. This characterization of safety works both for deterministic and nondeterministic step relations.

A state $t : A$ *everywhere halts* if every state it could possibly reach has at least one path to a halted state:

Definition everywhere_halts (t : A) $\triangleq$
 $\forall t''$, step_star t t'' $\to$
 $\exists t'$, [$\wedge$ step_star t'' t' & halted t'].

By contrast, we say a state t *somewhere halts* if there exists a halting path from state t:

Definition somewhere_halts (t : A) $\triangleq$
 $\exists t'$, [$\wedge$ step_star t t' & halted t'].

It naturally follows that if a state satisfies everywhere_halts then it also satisfies somewhere_halts.

7.1 Termination of Finite Games

In games with finite action spaces A, one can prove termination of a multiplayer dynamics by showing that the dynamics never revisits states. Our Coq library captures such reasoning generically, for any step relation that satisfies certain properties, by mapping step to a new operational semantics hstep, for "step with history", that tracks the history of states visited at each point in an execution.

We build hstep within a section parameterized by a game over actions of type A:

Section history.
 Context $\{A\}$ ‘{gameClass : game A}.
 Notation state $\triangleq$ ({ffun $[N] \to A$}).
 Variable step : state $\to$ state $\to$ Prop.
 ($*\ldots*$)

A state of the hstep semantics is defined as a triple (s, u, t) of type:

Let hstate $\triangleq$ (simpl_pred state $*$ simpl_pred state $*$ state)%**type**

comprising

—a predicate s giving the states visited so far;

—a predicate u giving the states not yet visited; and

—the current state t.

The hstep relation is defined as:

```
Inductive hstep : hstate → hstate → Prop ≜
| hstep_step (s u : simpl_pred state) t t' :
  let: s' ≜ predU1 t' s in
  let: u' ≜ predD1 u t' in
    u t' →
    step t t' →
    hstep (s,u,t) (s',u',t').
```

where predU1 t' s is the predicate corresponding to the set $\{t'\} \cup s$ while predD1 u t' corresponds to the set $u - \{t'\}$. That is, we can take an hstep from (s, u, t) to (s', u', t') as long as t' is unvisited (u t'), step t t' holds in the underlying step relation, and s' and u' mark the unvisited state t' as visited.

To ensure that the predicates s and u consistently cover the entire state space, we impose the following invariant on hstates:

```
Definition inv (sut : hstate) : Prop ≜
  let: (s,u,t) ≜ sut
  in [∧ predU s u =1 λx ⇒ x ∈ (enum state) (*Condition 1*)
      , predI s u =1 λ_ ⇒ false (*Condition 2*)
      & s t (*Condition 3*)].
```

which asserts that (1) the union of s and u is extensionally equivalent to the entire state space (enum state); (2) s and u are disjoint; and (3) the current underlying state t of type state is in s. The initial hstate:

```
Definition init (t : state) ≜ (pred1 t, predD1 predT t, t).
```

parameterized by some underlying initial state t satisfies this invariant, for example. The predicate pred1 t is the singleton set $\{t\}$. An hstate sut is halted:

```
Definition hstep_halted (sut : hstate) ≜
  [∨ halted sut.2 | let: (s,u,t) ≜ sut in #|u| = O]
```

when either the underlying state t is halted (halted $sut.2$) or the size of the unvisited set is 0 (there are no more states to visit).

Some step relations may revisit previously visited states. To rule out such step relations in our termination proof, we require that step be packaged with a predicate, P, over hstates that satisfies the following properties:

```
Variable P : hstate → Prop.
Hypothesis init_P : ∀t, P (init t).
Hypothesis step_P :
  ∀s u t t',
    inv (s,u,t) → P (s,u,t) → step t t' →
    [∧ u t' & P (predU1 t' s, predD1 u t', t')].
```

P must hold of the initial state, for any initial underlying state t. Furthermore, if inv (s,u,t) and P (s,u,t) hold initially, and the system steps from t to some new state t', then t' is a previously unvisited state $(u\ t')$ and P can be reestablished on the new hstate that results from removing t' from u and adding it to s.

The step_P property is sufficient to prove a number of other properties, such as the following lemma about the preservation of P and inv under the reflexive transitive closure of hstep:

Lemma hstep_star_inv sut sut' :
  inv $sut \rightarrow$ P $sut \rightarrow$
  step_star hstep sut $sut' \rightarrow$
  [$\wedge$ inv sut' & P sut'].

The step_P property also implies that steps from states t to t' can be matched by corresponding steps in the history step relation hstep:

Lemma step_hstep su t t' :
  inv $(su,t) \rightarrow$
  P $(su,t) \rightarrow$
  step t $t' \rightarrow$
  $\exists su'$, [$\wedge$ hstep (su,t) (su',t') & P (su',t')].

assuming the initial hstate (su, t) satisfies inv and P.

How is P typically instantiated? For potential games with potential function Φ, we define it as:

Definition P (sut : hstate) : Prop $\triangleq$
  let: $(s,u,t) \triangleq sut$ **in**
  $\forall t_0,\ s\ t_0 \rightarrow \Phi\ t \leq \Phi\ t_0$.

A state (s, u, t) satisfies P only if every previously visited state t_0 ($s\ t_0$: in the "seen" set s) has potential greater than or equal to that of the current state t ($\Phi\ t \leq \Phi\ t_0$). This inequality, together with the condition cost $i\ t' <$ cost $i\ t$ that defines better response in the definition of better_response_step, implies that potential games never revisit states (the step_P condition given above).

To prove termination of games like potential games that satisfy step_P, we first prove a few useful auxiliary lemmas:

—hstep everywhere_halts (assuming a safe initial hstate sut):

Lemma hstep_everywhere_halts_or_stuck sut :
  safe hstep hstep_halted $sut \rightarrow$
  everywhere_halts hstep hstep_halted sut.

—everywhere termination of hstep implies everywhere termination of step from safe initial states t (assuming step_P within a **Section** context):

Lemma everywhere_halts_hstep_step s u t :
  safe step halted $t \rightarrow$
  inv $(s,u,t) \rightarrow$
  P $(s,u,t) \rightarrow$
  everywhere_halts hstep hstep_halted $(s,u,t) \rightarrow$
  everywhere_halts step halted t.

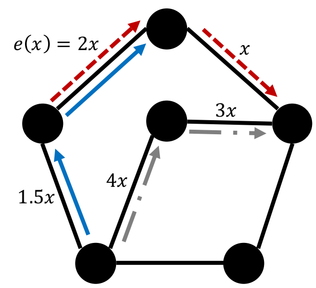

Fig. 2. An Atomic Routing Game With Three Players.

—hstep either everywhere terminates or is stuck (proved by induction on the unvisited set u):

Lemma hstep_everywhere_halts_or_stuck sut :
safe hstep hstep_halted $sut \rightarrow$
everywhere_halts hstep hstep_halted sut.

—for states (s, u, t) satisfying inv and P, safety of step implies safety of hstep:

Lemma safe_step_hstep $s\ u\ t$:
inv $(s,u,t) \rightarrow$
P $(s,u,t) \rightarrow$
safe step halted $t \rightarrow$
safe hstep hstep_halted (s,u,t).

The formal proof that step terminates (assuming step_P):

Theorem step_everywhere_halts_or_stuck t :
safe step halted $t \rightarrow$
everywhere_halts step halted t.

applies lemma safe_step_hstep to the safety hypothesis safe step halted t to prove safety, under hstep, of the initial hstate init $t \triangleq$ (pred1 t, predD1 predT t, t). State init t also satisfies inv (because it is initial) and P (by init_P). By the previously proved lemma everywhere_halts_hstep_step, it suffices to prove that

everywhere_halts hstep hstep_halted (init t),

which itself follows by lemma hstep_everywhere_halts_or_stuck and from safety under hstep of init t.

8. APPLICATIONS

8.1 Atomic Routing Games

As an example of a potential game that converges under better-response dynamics, consider Atomic Routing as depicted in Figure 2. In the general Atomic Routing game, N players each attempt to choose a path from some source to some sink vertex (both of which may differ across players) such that the path chosen minimizes the player's cost. The cost of a path is the sum of the costs of the edges in the path, where the cost of each edge is determined by a function $e(x)$ of the number of players that chose that edge (the traffic x).

For example, in Figure 2, the solid-arrow blue player pays 5.5 (1.5 plus 4 for the shared edge) while the dotted red player pays 5. The half-dotted half-solid gray player pays 7. It would not be profitable for the half-solid half-dotted gray player to follow the red–blue path since then it would pay 10: 3 for the edge shared with blue, 6 for the edge shared with red and blue, and 2 for the edge shared just with red.

We formalize Atomic Routing in a section that declares the type of vertices T, the number of players num_players, the graph g as an adjacency matrix, the codomain of the cost function $\mathbb{R}$, the cost functions ecosts associated with each edge in the graph, and a proof ecosts_pos that the cost functions are positive.

```
Section AtomicRoutingGame.
  Variable T : finType.
  Variable num_players : nat.
  Variable g : 'M[bool]_(#|T|, #|T|).
  Variable ℝ : realFieldType.
  Variable ecosts : ∀x y : [#|T|], nat → ℝ.
  Hypothesis ecosts_pos :
    ∀x y n, (0 ≤ ecosts x y n)%R.
  ...
```

Recall that #|T| is the cardinality of the finite type of vertices T, while [#|T|] is syntax for the dependent type of natural numbers in the range $[0, \#|\mathsf{T}|)$. We represent the graph g : 'M[bool]_(#|T|, #|T|) as an adjacency matrix mapping each pair of vertices to a boolean value indicating whether or not there is an edge between them. The ecosts function takes (the indices of) two vertices as arguments, along with the traffic on that edge (type nat), and returns the cost (type $\mathbb{R}$).

We represent a player in the game as a pair of a source and a sink:

```
Record player : Type ≜
  mkPlayer {
    source : [#|T|];
    sink : [#|T|] }.
```

and the set of all players as a function from player indices [num_players] to player records:

```
Variable players : [num_players] → player.
```

A path of size up to #|T| is defined as a tuple of (indices to) vertices that maps each vertex in the path to that vertex's successor. Paths must additionally satisfy the predicate sspred_pred x y, standing for "source–sink path from vertex x to y":

```
Definition path ≜ ([#|T|]^#|T|)%type.
Fixpoint sspath_rec n (x y : [#|T|]) : pred path ≜
  [pred p : path |
   [|| [&& p x == y & g x (p x)]
     | [&& g x (p x) & if n is n'.+1 then sspath_rec n' (p x) y p else false]]].
Definition sspath_pred (x y : [#|T|]) ≜ sspath_rec #|T| x y.
```

A path satisfies sspath_rec $n\ x\ y$ (and therefore sspath_pred $x\ y$, assuming $n \leq \#|T|$) when either (1) the path p maps vertex x to y and edge (x, y) is an available edge in the graph ([&& $p\ x == y$ & g x ($p\ x$)]), or (2) n is greater than 0 and p maps vertex x to an available vertex $p\ x$ such that sspath_rec $(n-1)$ $(p\ x)$ y p ($p\ x, y$ is recursively a valid path). The type strategy i:

```
Notation src i ≜ (source (players i)).
Notation snk i ≜ (sink (players i)).
Definition strategy (i : [num_players]) ≜
  sig (λ the_path ⇒ sspath_pred (src i) (snk i) the_path).
```

packages together in a sigma type player i's path with a proof that the path satisfies sspath_pred (src i) (snk i) the_path.

The sspath_pred predicate assumes that paths are of size no greater than #|T|, which is sufficient to represent all cycle-free paths. An alternative representation (if cyclical paths of size greater than #|T| are necessary) is to define paths as linked lists together with an inductively-defined predicate in place of the the fixpoint sspath_rec.

We define states of the Atomic Routing game as tuples mapping player indices to dependent pairs of a player index i and a strategy indexed by i:

```
Definition strategy_pkg ≜ {: {i : [num_players] & strategy i}}.
Notation st ≜ ((strategy_pkg^num_players)%type.
```

The amount of traffic over a particular edge in the graph is defined as the number of players that have chosen paths that contain that edge:

```
Definition traffic_edge (s : st) (x y : [#|T|]) : nat ≜
  #|edgePlayers s x y|.
```

where edgePlayers is a predicate that returns true for each player index i for which i's path contains an edge from x to y:

```
Definition edgeOfPlayer i (s : st) (x y : [#|T|]) ≜
  path_of s i x == y.
Definition edgePlayers (s : st) (x y : [#|T|]) : pred [num_players] ≜
  [pred i | edgeOfPlayer i s x y].
```

The predicate edgeOfPlayer $i\ s\ x\ y$ is satisfied only if there is a edge (x, y) in player i's path in state s.

The cost of an edge (x, y) is defined, using the parameterized ecosts, as a function of the traffic over that edge:

```
Definition cost_edge (s : st) (x y : [#|T|]) ≜
  ecosts x y (traffic_edge s x y).
```

The cost to a particular player i is the sum of the costs of each edge in player i's source–sink path:

```
Definition costFun (i : [num_players]) (s : st) : ℝ ≜
  ∑_(x : [#|T|])
  ∑_(y : [#|T|]) if edgeOfPlayer i s x y then cost_edge s x y else 0.
```

```
Instance costInstance : CostClass num_players ℝ [finType of strategy_pkg]
  ≜ costFun.
Program Instance costAxiomInstance
  : CostAxiomClass costInstance. (*proof elided*)
```

The costs given by costFun are all positive, and therefore satisfy the CostAxiomClass of our formalization of games in Section 4.

To construct the overall game instance for Atomic Routing, we define the allowable moves of each player i as those that leave unmodified the first projection of i's strategy_pkg:

```
Definition movesFun (i : [num_players]) : rel strategy_pkg ≜
  [λ p p' : strategy_pkg ⇒ projT1 p == projT1 p'].
Instance movesInstance : MovesClass num_players [finType of strategy_pkg]
  ≜ movesFun.
Instance gameInstance : game costAxiomInstance movesInstance.
```

This definition of movesInstance ensures that players only ever update their strategies, never the values i : [num_players] that index the types of their strategies.

Atomic Routing is a Potential Game. Atomic Routing is a potential game with the following potential function:

```
Definition phiFun (s : st) : ℝ ≜
  ∑_(x : [#|T|])
  ∑_(y : [#|T|])
  ∑_(1 ≤ z < (traffic_edge s x y).+1)
  ecosts x y z.
```

That is, for any state s and any new state $s' \triangleq$ upd $i\ s\ s'_i$ resulting from a unilateral deviation of player i, the following equation holds:

```
phiFun s' − phiFun s = cost i s' − cost i s. (*Potential Equation*)
```

To see why, consider the effect of a player i's unilateral deviation on the traffic at some edge (x, y). Either i's strategy in state s' differs from s at edge (x, y) or it doesn't, leading to four possibilities as encapsulated by the following lemmas. In each case, traffic at edge (x, y) can differ by at most 1:

```
Lemma traffic00 (i : [num_players]) (x y : [#|T|]) s s' : Move i s s' →
  edgeOfPlayer i s' x y = false → edgeOfPlayer i s x y = false →
  traffic_edge s' x y = traffic_edge s x y.
Lemma traffic01 (i : [num_players]) (x y : [#|T|]) s s' : Move i s s' →
  edgeOfPlayer i s' x y = false → edgeOfPlayer i s x y →
  (traffic_edge s' x y).+1 = traffic_edge s x y.
Lemma traffic10 (i : [num_players]) (x y : [#|T|]) s s' : Move i s s' →
  edgeOfPlayer i s' x y → edgeOfPlayer i s x y = false →
  traffic_edge s' x y = (traffic_edge s x y).+1.
Lemma traffic11 (i : [num_players]) (x y : [#|T|]) s s' : Move i s s' →
  edgeOfPlayer i s' x y → edgeOfPlayer i s x y →
  traffic_edge s' x y = traffic_edge s x y.
```

In each of the traffic lemmas above, the Move $i\ s\ s'$ states that s' is a valid unilateral update by player i (i's strategy may differ, as allowed by movesFun above, but the strategies of all other players $j \neq i$ are unchanged).

As a representative case of the proof of Potential Equation above, consider traffic10 in which a player i uses some edge (x, y) in state s' but not in state s. In this case, Potential Equation simplifies to:

$$\begin{aligned}&\textstyle\sum_{\_}(1 \leq z < (\mathsf{traffic\_edge}\ s'\ x\ y).{+}1)\ \mathsf{ecosts}\ x\ y\ z\ - \\ &\textstyle\sum_{\_}(1 \leq z < (\mathsf{traffic\_edge}\ s\ x\ y).{+}1)\ \mathsf{ecosts}\ x\ y\ z \\ = &\ \mathsf{ecosts}\ x\ y\ (\mathsf{traffic\_edge}\ s'\ x\ y) - 0\end{aligned}$$

which by traffic10 can be rewritten in terms of s as:

$$\begin{aligned}&\textstyle\sum_{\_}(1 \leq z < (\mathsf{traffic\_edge}\ s\ x\ y).{+}2)\ \mathsf{ecosts}\ x\ y\ z\ - \\ &\textstyle\sum_{\_}(1 \leq z < (\mathsf{traffic\_edge}\ s\ x\ y).{+}1)\ \mathsf{ecosts}\ x\ y\ z \\ = &\ \mathsf{ecosts}\ x\ y\ (\mathsf{traffic\_edge}\ s\ x\ y).{+}1 - 0\end{aligned}$$

The left-hand side equals:

$$\mathsf{ecosts}\ x\ y\ (\mathsf{traffic\_edge}\ s\ x\ y).{+}1)$$

by the following equality, over functions f and positive m:

$$\textstyle\sum_{\_}(1 \leq i < m.{+}1)\ f\ i - \sum_{\_}(1 \leq i < m)\ f\ i = f\ m$$

thus finishing the proof of this case. The other 3 cases follow in a similar fashion, by application of the appropriate traffic lemma and some arithmetic.

Once the Atomic Routing game is proved a potential game:

Instance PhiAxiomInstance : PhiAxiomClass phiInstance $\triangleq$ (∗. . .∗).
Instance AtomicPotentialInstance : Potential PhiAxiomInstance.

it is straightforward to apply our library results from Section 6.1 to prove both that the Atomic Routing game has a PNE and that Atomic Routing converges to a PNE under better-response dynamics (recall that halted t is defined as PNE t):

Lemma AtomicRouting_exists_PNE (t_0 : st) : $\exists t$: st, PNE t.
Proof. by apply: (exists_PNE t_0). $\mathbb{Q}$ed.
Lemma AtomicRouting_everywhere_halts (t : st) : everywhere_halts step halted t.
Proof. by apply: better_response_everywhere_halts. $\mathbb{Q}$ed.

8.2 Affine Congestion Games

Our second application is to Affine Congestion games (Figure 3), in which N players each must choose a subset of M resources. In the figure, we represent the resources as servers and the players (which might be, e.g., network flows) as circles. The cost incurred by each player is the sum of the costs of the resources chosen by that player, where the cost of each resource is an affine function $(ax + b)$ of the amount of traffic x on that resource (the number of players having chosen that resource). We require that a and b are nonnegative so that costs are nonnegative, as required by the CostAxiomClass.

We model the congestion game in a section parameterized by the finite type of resources T : finType and the number of players num_players : nat.

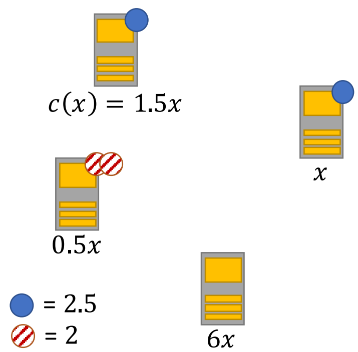

Fig. 3. An Affine Congestion Game Mapping Two Players (Blue–Solid, Red–Hatched) to 4 Servers. Costs incurred by each player are listed in the lower left.

Section CongestionGame.
  Variable T : finType. (∗∗ The **type of** resources ∗)
  Variable num_players : nat. (∗∗ The number **of** players ∗)

The number of resources is therefore #|T|, the cardinality of T. A strategy in this game is a subset of the resources in T, which we represent as finite functions from T to bool.

  Definition strategy ≜ {ffun T → bool}.

Note that with this definition of strategy, a player may choose the empty subset of resources. It is straightforward to update the game type to enforce a particular policy on valid strategies. For example, one might let strategy equal:

  Definition strategy′ ≜ {f : ffun T → bool | ∃t : T. f t = true}

thus enforcing that each player choose at least one server.

To define the cost function for the game, we first model affine functions via the following record:

  Record affineCostFunction : Type ≜
  { aCoeff : rat;
    bCoeff : rat;
    aCoeff_positive : 0 ≤ aCoeff;
    bCoeff_positive : 0 ≤ bCoeff }.

The cost function for each resource is then a parameter of the model:

  Variable costs : {ffun T → affineCostFunction}.
  Definition evalCost (t : T) (x : nat) : rat ≜
    aCoeff (costs t)∗x + bCoeff (costs t).

The function evalCost t x calculates, with respect to costs, the affine function associated with resource t when applied to x traffic.

States of the congestion game are finite functions from player indices to strategies, as abbreviated by the following notation:

Notation st $\triangleq$ ({ffun [num_players] $\rightarrow$ strategy})%**type**.

To define the cost incurred by a player i in state s : st, we first define the *load* on a resource t – or total number of players using that resource – as the cardinality of the set of players using t:

Definition load (s : st) (t : T) : nat $\triangleq$ #|[set i | s i t]|.

The cost to player i of a state s is then just the sum of the costs of all resources in i's strategy:

Definition costFun (i : [num_players]) (s : st) : rat $\triangleq$
$\sum$_t **if** s i t **then** evalCost t (load s t) **else** 0.

In the routing game of the previous section, the movesFun prohibited players from updating the indices of their strategies. Here, the type of player strategies is uniform across indices, making movesFun the trivial relation:

Definition movesFun (i : [num_players]) : rel strategy $\triangleq$ [λ _ _ : strategy $\Rightarrow$ true].

that simply accepts all strategy updates.

Affine Congestion is Smooth. The Affine Congestion game we model above is $(\frac{5}{3}, \frac{1}{3})$-smooth, as was first proved by Roughgarden [28]. Smoothness in turn implies a robust Price of Anarchy guarantee of 5/2 (Section 6.2).

The proof relies on the following arithmetic fact, originally noted by Christodoulou and Koutsoupias in [8]:

$$\forall yz : \mathsf{nat}.\ y * (z+1) \leq \frac{5}{3}y^2 + \frac{1}{3}z^2$$

which we formalize as the lemma:

Lemma christodoulou (y z : nat) :
y%:$\mathbb{Q}$ $*$ (z%:$\mathbb{Q}$ + 1) $\leq$ 5%:$\mathbb{Q}$/3%:$\mathbb{Q}$$*$$y$%:$\mathbb{Q}$^2 + 1%:$\mathbb{Q}$/3%:$\mathbb{Q}$$*$$z$%:$\mathbb{Q}$^2.

In our statement of christodoulou, the ubiquitous %:$\mathbb{Q}$ simply coerces natural numbers to $\mathbb{Q}$. Its proof is by case analysis on y where the only nontrivial case, in which we have $1 < y$, is dispatched by reduction to the inequality of arithmetic and geometric means (AGM inequality). Ssreflect's ssrnum module provides a convenient proof of the AGM inequality via the lemma lerif_AGM2.

Smoothness of the game follows from a consequence of christodoulou and the fact that in a unilateral deviation of any player i, the load at a given resource can increase by at most one. For a fuller exposition of the structure of this proof, see [28, Section 2.3.1].

9. CONCLUSION

In this paper, we report on a library in Ssreflect/Coq for doing algorithmic game theory. Our results include a number of definitions and theorems, including: multiplayer games; solution concepts such as Pure Nash Equilibria, Mixed Nash Equilibria, Coarse Correlated Equilibria and ϵ-approximate variations; subclasses of games such as potential games and smooth games; better-response dynamics; convergence of potential games to Pure Nash Equilibria (PNE); bounds on the Price of Stability of the PNE of potential games; bounds on the Price of Anarchy of smooth games; a proof that the Atomic Routing game converges under better-response dynamics; and a proof that Affine Congestion games are $(5/3, 1/3)$-smooth. As far as we are aware, we are the first to formalize Atomic Routing and Affine Congestion games and to formalize the proofs that (1) Atomic Routing games are potential games and (2) Affine Congestion games are $(5/3, 1/3)$-smooth.

ACKNOWLEDGMENTS

We thank the anonymous referees for their detailed and insightful comments on an earlier version of this paper. This material is based on work supported by the National Science Foundation under Grant No. CCF-1657358.

References

[1] Sebastian Aland, Dominic Dumrauf, Martin Gairing, Burkhard Monien, and Florian Schoppmann. Exact price of anarchy for polynomial congestion games. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 218–229. Springer, 2006.

[2] Louis Bachelier. Théorie mathématique du jeu. In *Annales Scientifiques de l'Ecole Normale Supérieure*, volume 18, pages 143–209. Elsevier, 1901.

[3] Alexander Bagnall, Samuel Merten, and Gordon Stewart. Brief announcement: Certified multiplicative weights update: Verified learning without regret. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 459–461, 2017.

[4] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. *ACM SIGPLAN Notices*, 50(1):55–68, 2015.

[5] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

[6] Avrim Blum, Eyal Even-Dar, and Katrina Ligett. Routing without regret: On convergence to nash equilibria of regret-minimizing algorithms in routing games. In *Proceedings of the twenty-fifth annual ACM Symposium on Principles of Distributed Computing*, pages 45–52. ACM, 2006.

[7] Emile Borel. La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes rendus de lAcadémie des Sciences*, 173(1304-1308):58, 1921.

[8] George Christodoulou and Elias Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, pages 67–73. ACM, 2005.

[9] Rachel Cummings, Michael Kearns, Aaron Roth, and Zhiwei Steven Wu. Privacy and truthful equilibrium selection for aggregative games. In *International Conference on Web and Internet Economics*, pages 286–299. Springer, 2015.

[10] Andrew V Goldberg, Jason D Hartline, Anna R Karlin, Michael Saks, and Andrew Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, 2006.

[11] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A small scale reflection extension for the Coq system. Technical report, INRIA, 2015.

[12] Daniel Huang and Greg Morrisett. An application of computable distributions to the semantics of probabilistic programming languages. In *European Symposium on Programming Languages and Systems*, pages 337–363. Springer, 2016.

[13] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413. Springer, 1999.

[14] Stéphane Le Roux. *Generalisation and formalisation in game theory*. PhD thesis, École normale supérieure (Lyon), 2008.

[15] Stéphane Le Roux. Acyclic preferences and existence of sequential nash equilibria: a formal and constructive equivalence. *Theorem Proving in Higher Order Logics*, pages 293–309, 2009.

[16] Pierre Lescanne. Dependent types for extensive games. *arXiv preprint arXiv:1611.06101*, 2016.

[17] Pierre Lescanne and Perrinel Matthieu. On the rationality of escalation. *arXiv preprint arXiv:1004.5257*, 2010.

[18] Pierre Lescanne and Matthieu Perrinel. "Backward" coinduction, Nash equilibrium and the rationality of escalation. *Acta Informatica*, 49(3):117–137, 2012.

[19] Brendan Lucier and Allan Borodin. Price of anarchy for greedy auctions. In *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, pages 537–553. Society for Industrial and Applied Mathematics, 2010.

[20] Assia Mahboubi and Enrico Tassi. Canonical structures for the working Coq user. In *International Conference on Interactive Theorem Proving*, pages 19–34. Springer, 2013.

[21] Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996.

[22] John Nash. Non-cooperative games. *Annals of Mathematics*, pages 286–295, 1951.

[23] John F Nash. Equilibrium in n-player games. *Proceedings of the National Academy of the Sciences (PNAS)*, 36(1):48–49, 1950.

[24] J. von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*, volume 60. Princeton University Press, 1944.

[25] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic Game Theory*, volume 1. Cambridge University Press, 2007.

[26] David Nowak. A framework for game-based security proofs. In *International Conference on Information and Communications Security*, pages 319–333. Springer, 2007.

[27] Tim Roughgarden. *Selfish routing and the price of anarchy*, volume 174. MIT press Cambridge, 2005.

[28] Tim Roughgarden. Intrinsic robustness of the price of anarchy. In *Proceedings of the forty-first annual ACM Symposium on Theory of Computing*, pages 513–522. ACM, 2009.

[29] Tim Roughgarden. The price of anarchy in games of incomplete information. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 862–879. ACM, 2012.

[30] Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, 2016.

[31] Tim Roughgarden and Florian Schoppmann. Local smoothness and the price of anarchy in splittable congestion games. *Journal of Economic Theory*, 156:317–342, 2015.

[32] Stéphane Le Roux, Érik Martin-Dorel, and Jan-Georg Smaus. An existence theorem of Nash equilibrium in Coq and Isabelle. In *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017.*, pages 46–60, 2017.

[33] Bas Spitters and Eelis Van der Weegen. Type classes for mathematics in type theory. *Mathematical Structures in Computer Science*, 21(04):795–825, 2011.

[34] René Vestergaard. A constructive approach to sequential Nash equilibria. *Information Processing Letters*, 97(2):46–51, 2006.

[35] Adrian Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 416–425. IEEE, 2002.

[36] Santiago Zanella-Béguelin, Benjamin Grégoire, Gilles Barthe, and Federico Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy, S&P 2009*, pages 237–250. IEEE Computer Society, 2009.

[37] Ernst Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proceedings of the fifth international congress of mathematicians*, volume 2, pages 501–504. II, Cambridge UP, Cambridge, 1913.