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, $\epsilon$-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 $\epsilon$-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:


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 $A$. 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 $(a_1, a_2, \ldots, a_N)$, where the action of player $i$ is drawn from $A_i$. The cost $C_i$ to player $i$ is calculated wrt. the tuple $(a_1, \ldots, a_i, \ldots, a_N)$ 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 $(a_1, \ldots, a_i, \ldots, a_N)$ is a *Pure Nash Equilibrium (PNE)* [23] if for every $i$ and potential deviant action $a'_i$,

$$C_i(a_1, \ldots, a_i, \ldots, a_N) \leq C_i(a_1, \ldots, a'_i, \ldots, a_N)$$

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 \( \sigma \) over states \( a \) such that
\[
E_{a \sim \sigma}[C_i(a_1, \ldots, a_i, \ldots, a_N)] \leq E_{a \sim \sigma}[C_i(a_1, \ldots, a'_i, \ldots, a_N)]
\]
for all \( i \) and \( a'_i \). The expected cost to player \( i \) in \( \sigma \) of \( a = (a_1, \ldots, a_i, \ldots, a_N) \) is less than or equal to the expected cost of \( (a_1, \ldots, a'_i, \ldots, a_N) \).

All the equilibrium notions above have approximate counterparts. For example,
\( \epsilon \)-approximate CCEs are those distributions \( \sigma \) such that
\[
E_{a \sim \sigma}[C_i(a_1, \ldots, a_i, \ldots, a_N)] \leq E_{a \sim \sigma}[C_i(a_1, \ldots, a'_i, \ldots, a_N)] + \epsilon
\]
Player \( i \) can gain (in expectation) at most \( \epsilon \) by deviating to \( a'_i \).

3.2 Algorithmic Game Theory

Algorithmic game theory (AGT) [25] applies traditional computer science tech-
niques 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 so-
cially 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 dynam-
ics 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 \( \epsilon \)-approximate CCEs. [30, Chapter 17]

3.4 Ssreflect/Coq

We use Ssreflect [11] libraries throughout our formalization. For readers more fa-
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 $A : \text{finType}$) as pairs of the type $A$ and an enumerator $\text{enum} : \text{list} A$. The enumerator satisfies the property:

$$\forall a : A. \text{count} a \text{ enum} = 1.$$ 

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

Finite Functions. Ssreflect models functions with finite domain: 

$$\{ \text{ffun } A \to B \}$$

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 $[0 \ldots N)$ 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 $[0 \ldots N)$ to values of type $A$ has type:

$$\{ \text{ffun } 'I N \to A \}$$

or in the notation which we use in this paper:

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

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:

\begin{verbatim}
Class CostClass (N : nat) (R : realFieldType) (A : finType) ≜
    cost_fun : [N] → \{ ffun [N] → A \} → R.
Notation ""cost"" ≜ (@cost_fun _ _ _). (at level 30).
Class CostAxiomClass N R A '(CostClass N R A) ≜
    cost_axiom (i : [N]) (f : \{ ffun [N] → A \}) : 0 ≤ cost i f.
Lemma cost_nonneg i f : 0 ≤ cost i f. Proof. apply: cost_axiom. Qed.
End costLemmas.
Class MovesClass (N : nat) (A : finType) ≜ moves_fun : [N] → rel A.
Notation ""moves"" ≜ (@moves_fun _ _ _). (at level 50).
\end{verbatim}

$^2$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].
Class game (A : finType) (N : nat) (R : realFieldType)
  (costAxiomClass : CostAxiomClass N R A)
  (movesClass : MovesClass N T) : Type ≜ {}.

The operational type class CostClass N R A – in a game with N players and action space A – asserts the existence of an R-valued cost function cost_fun (notation cost) mapping a player index of type [N] and state of type \{ffun [N] → A\} (assigning an action of type A to each player) to a real-valued cost of type 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 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 ≜ (fun [N] → A) type`.

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

**Definition** `PNE (t : state N A) : Prop ≜ ∀ (i : [N]), moves i t t' i = cost i t ≤ cost i (upd i t t')`.

In the definition of `PNE`, the relation `moves i t 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 $i ≠ j$.

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 ≜ [∀ i : [N], ...] ∧ [∀ t' : state N A, Move i t t' ≡⇒ (cost i t ≤ 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 `[∀ i : [N], ...]` of a boolean-valued predicate over a finite domain, as well as `Ssreflect`’s boolean implication (⇒) 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** `R : realFieldType`.

**Definition** `dist axiom (f : {fun A → R}) : bool ≜ [&& ∑ a (f a) == 1] ∧ [∀ a : A, f a ≥ 0]`.

**Record** `dist : Type ≜ mkDist { pmf := {fun A → R}; dist_ax : dist_axiom pmf }`.

(`* ... canonical projections ... *`)

**End** `Dist`.

We represent discrete distributions as finite probability mass functions `pmf` (type `{fun A → R}`) that map values in the event space $A$ to their weights in $R$. 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.\footnote{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.}

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

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

**Definition** `expectedCondValue` ($f : A \rightarrow \mathbb{R}$) (p : pred $A$) $\triangleq$  
\[
\left(\sum_{(t : A \mid p \_ t)} (d \_ t \ast f \_ t)\right) / \left(\sum_{(t : A \mid p \_ t)} d \_ t\right).
\]

**Definition** `expectedValue` ($f : A \rightarrow \mathbb{R}$) $\triangleq$ `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_lin` $f$ $g$ :  
`expectedValue` ($\lambda t \Rightarrow f \_ t + g \_ t$) = `expectedValue` $f$ + `expectedValue` $g$.

**Lemma** `expectedValue_mull` $f$ $c$ :  
`expectedValue` ($\lambda t \Rightarrow c \ast f \_ t$) = $c \ast$ `expectedValue` $f$.

**Lemma** `expectedValue_const` $c$ : `expectedValue` ($\lambda \_ \Rightarrow c$) = $c$.

**Lemma** `expectedValue_range` $f$ :  
($\forall t : A$, $0 \leq f \_ t \leq 1$) $\rightarrow$ $0 \leq$ `expectedValue` $f \leq 1$.

\[(\ast \ldots \ast)\]

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 $A : \text{finType}$.
Variable $t_0 : A$.

**Definition** `uniform_dist` : `ffun` $A \rightarrow \text{rat}$ $\triangleq$  
`ffun` ($\lambda \_ \Rightarrow 1 / \#|A|\%\mathbb{R}$).

**Lemma** `uniform_normalized` : `dist_axiom` `uniform_dist`.

**Definition** `uniformDist` : `dist` $A$ [numDomainType of `rat`] $\triangleq$
Lemma expectedValue_uniform (f : A → rat) :
expectedValue_uniformDist f = (∑_(t : A) (f t)) / 1 / |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 state N A]) (rty ≜ R) prod_pmf.
Definition prod_dist : dist [finType of state N A] R ≜ mkDist prod_pmf prod_pmf_dist.
End product.

We assume \( N \) distributions, given by the finite function \( f : (ffun [N] → dist A R) \) 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 \( \epsilon \)-approximate CCEs, and then specializing \( \epsilon = 0 \) to yield nonapproximate CCEs. CEs are a refinement of \( \epsilon \)-approximate CEs, which are themselves a subset of \( \epsilon \)-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, \( \epsilon \)-CCEs, as follows (\( N, A, \) and \( R \) are section parameters):

Definition eCCE (\( \epsilon : R \)) (d : dist [finType of state N A] R) : Prop ≜
\[ \forall (i : [N]) (t'_i : A), \]
\[ (\forall t : state N A, t \in support d \rightarrow moves i (t i) t'_i) \rightarrow \]
expectedCost i d ≤ expectedUnilateralCost i d t'_i + \( \epsilon \).

Player \( i \) can gain at most \( \epsilon \) 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 \textit{cost} is the cost function associated with the game over type \textit{A}.

The function \textit{expectedUnilateralCost} gives the expected value, to player \textit{i}, of a unilateral move by \textit{i} to action \textit{t’}_i:

\textbf{Definition} \textit{expectedUnilateralCost}

\((i : [N]) (d : \text{dist} \text{[finType of state N A]} \mathbb{R}) (t’_i : A) \triangleq \text{expectedValue}(d (\lambda t : \text{state N A} \Rightarrow \text{cost}(i \text{upd} i t t’_i))).\)

The function \textit{upd} is the same as that used to define \textit{PNE} above. Nonapproximate CCEs specialize \(\epsilon\)-CCEs to \(\epsilon = 0\):

\textbf{Definition} \textit{CCE} \(d : \text{dist} \text{[finType of state N A]} \mathbb{R}) : \text{Prop} \triangleq \epsilon\text{CCE 0} d.\)

and thus are trivially also \(\epsilon\text{CCEs.}\)

Correlated equilibria are distributions \(\sigma\) over states \(a\) such that

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

for all \(i, b_i, \text{and } b’_i.\) That is, player \textit{i}’s calculation is conditioned on the fact \(a_i = b_i\) (the realization of player \textit{i}’s action in state \(a\) drawn from \(\sigma\) 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 \(\sigma\) 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]).

\textit{Price of Anarchy.} In our formal development, we define POA as:

\textbf{Definition} \textit{POA} \(\mathbb{R} \triangleq \)

\[ \text{Cost}(\text{arg \_ max PNEb Cost } t_0) / \text{Cost}(\text{arg \_ min predT Cost } t_0) \).

This definition’s main ingredients are:

—The objective function

\textbf{Definition} \textit{Cost} \((t : \text{state N A}) : \mathbb{R} \triangleq \sum_i (\text{cost}(i t)).\)

which sums the per-player costs \textit{cost} \(i t\) of state \(t\) in the context of game \textit{A};
— An optimal state of game $A$, defined as:

\[
\arg\min_{t_0} \text{predT} \text{ Cost } t_0
\]

satisfies optimality as given by the following predicate over states:

**Definition** optimal : pred (state $N A$) \(\triangleq\) \(\lambda t \to [\forall t', \text{Cost } t \leq \text{Cost } t']\).

The function \(\arg\min (P : \text{pred } I) (F : I \to \mathbb{R}) (i_0 : I)\) — 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 $t_0 : \text{state } N A$ to $\arg\min$ to ensure that $\text{state } N A$ is inhabited, and $\arg\min \text{predT}$ is therefore total (because all $t_0$ satisfy the top predicate $\text{predT}$).

— The maximum-cost Pure Nash Equilibrium state

\[
\arg\max \text{PNEb Cost } t_0,
\]

a state of type $\text{state } N A$ that maximizes the cost function $\text{Cost}$ restricted to $\text{PNEb}$.

To define POA as a computable (boolean- rather than Prop-valued) function, it is important that the auxiliary predicates used above — optimal and $\text{PNEb}$ — are themselves computable. To use $\text{Ssreflect}$’s boolean quantification (e.g., $[\forall t', \text{Cost } t \leq \text{Cost } t']$, returning a boolean), we must also know that states $\text{state } N A$ are finite. To prove, e.g., that the state $\arg\max \text{PNEb Cost } t_0$ 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**\(\text{POS} : \mathbb{R} \triangleq\) \(\text{Cost } (\arg\min \text{PNEb Cost } t_0) / \text{Cost } (\arg\min \text{predT Cost } t_0)\).

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**\(\text{POS} \leq \text{POA}\)

(has\_PNE : \text{PNEb } t_0) : \text{POS} \leq \text{POA}.

In order for POS and POA to be defined, we must assume that game $A$ has at least one PNE (has\_PNE : \text{PNEb } t_0). The cost function for the game must also be nonnegative, a constraint satisfied by the game’s \text{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 $\Phi$ – mapping game states to $\mathbb{R}$ – such that

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

For any unilateral deviation by some player $i$ from action $t_i$ to $t_i'$, the potential function $\Phi$ 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 $\Phi$) 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:

\begin{verbatim}
Class PhiClass (N : nat) (R : realFieldType) (A : finType) (
  (costAxiomClass : CostAxiomClass N R A)
  (movesClass : MovesClass N A) : Type \triangleq
Phi : state N A \rightarrow R.
\end{verbatim}

and then a type class for the potential axiom:

\begin{verbatim}
Class PhiAxiomClass (N : nat) (R : realFieldType) (A : finType) (
  (costAxiomClass : CostAxiomClass N R A)
  (movesClass : MovesClass N A)
  (phiClass : PhiClass costAxiomClass movesClass) : Type \triangleq
PhiAxiom :
  \forall (i : [N]) (t : state [N] A) (t_i' : A),
  moves i (t i) t_i' \rightarrow
  let t' \triangleq \text{upd } i t t_i' \text{ in } \Phi t' - \Phi t = \text{cost } i t' - \text{cost } i t.
\end{verbatim}

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: $\text{moves } i (t i) t_i'$. Stated another way, the $\Phi$ 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 $\lambda_\_ \_ \rightarrow \text{true}$ (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:

\begin{verbatim}
Theorem exists_PNE (t_0 : state N A) : \exists t : state N A, PNE t.
\end{verbatim}

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 \texttt{minimal} those states that minimize the potential function $\Phi$:

**Definition** \texttt{minimal} : \texttt{pred (state N A)} $\triangleq$

\[
[pred t : \text{state N A} | \forall t' : \text{state N A}, \Phi t \leq \Phi t'].
\]

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

**Lemma** \texttt{minimal_PNE} ($t : \text{state N A}$) : minimal $t \rightarrow$ PNE $t$.

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

**Lemma** \texttt{Phi_cost_le} ($t : \text{state N A}$) i ($t'_i : A$):

\[
\text{moves } i \text{ (} t \text{) } t'_i \rightarrow
\]

\[
\text{let } t' \triangleq \text{upd } i \text{ (} t \text{) } t'_i \text{ in}
\]

\[
\Phi t \leq \Phi t' \rightarrow \text{cost } i \text{ (} t \text{) } \leq \text{cost } i \text{ (} t' \text{)}.
\]

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

From **Lemma** \texttt{minimal_PNE}, it is straightforward to prove that at least one PNE exists, by exhibiting a state $t$ that minimizes $\Phi$:

**Definition** \texttt{Phi_minimizer} ($t_0 : \text{state N A}$) : $\text{state N A} \triangleq \text{arg min } \text{pred} T \Phi t_0.$

In order to build \texttt{Phi_minimizer}, we must first ensure that at least one state exists ($t_0 : \text{state N A}$).

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

**Hypothesis** (HAgt0 : 0 < $\alpha$)

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

**Hypothesis** AB_bound_Phi :

\[
\forall t : \text{state N A}, \text{Cost } t / \alpha \leq \Phi t \leq \beta \ast \text{Cost } t.
\]

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

**Lemma** POS_bounded ($t : \text{state N A}$) (PNE t : PNE $t$) : POS $t \leq \alpha \ast \beta$.

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

\[
\text{Cost (arg_min PNEb Cost } t_0) \leq \text{Cost } t^\Phi \leq \alpha \ast \Phi t^\Phi \leq \alpha \ast \Phi t^* \leq \alpha \beta \ast \text{Cost } t^*.
\]
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 \( \lambda \) and \( \mu \): as Roughgarden shows in [28, Section 2.1], a game that is \( (\lambda,\mu) \)-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 \( (\lambda,\mu) \)-smooth if for every two states \( t \) and \( t^* \), the following condition holds:

\[
\sum_{i=1}^{N} \text{cost } i \left( t_1, \ldots, t^*_i, \ldots, t_N \right) \leq \lambda \cdot \text{Cost } t^* + \mu \cdot \text{Cost } t
\]

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 \( \lambda \) times the cost of the new state \( t^* \) plus \( \mu \) 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 \( \lambda \) parameter, which is typically greater than or equal to 1, relates the mixed state on the left to the “optimal” deviation \( t^* \). The \( \mu \) 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 \( (\lambda,\mu) \)-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:

\[
\text{Cost } t = \sum_{i=1}^{N} \text{cost } i \left( t_1, \ldots, t^*_i, \ldots, t_N \right) \leq \lambda \cdot \text{Cost } t^* + \mu \cdot \text{Cost } t
\]

\[
\leq \lambda \cdot \text{Cost } t^* + \mu \cdot \text{Cost } t \]

\[
\leq \frac{\lambda}{1-\mu} \cdot \text{Cost } t^*
\]

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:
A Library for Algorithmic Game Theory in Ssreflect/Coq

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.

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.

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' →
\sum_(i : [N]) cost i (upd i t (t' i)) ≤
lambda of A * Cost t' + mu of A * Cost t.
Notation """"smooth AX"""" ≜
(@SmoothnessAxiom _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Lemma smooth_CCE \((d : \text{dist}[\text{finType of state } N A][R])\) \((t' : \text{state } N A)\):

\[
\text{CCE } d \to \text{dist_valid_Move } d t' \to \text{ExpectedCost } d \leq (\lambda A / (1 - \mu A)) \ast \text{Cost } t'.
\]

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 \(\sum(i : [N]) \text{expectedCost } i d\), 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, \(\text{step}\), and a predicate, \(\text{halted}\), that specifies when an execution has safely ended.

**Section stepDefs.**

- **Context** \(\{A : \text{Type}\}\).
- **Variable** \(\text{step} : A \to A \to \text{Prop}\).
- **Variable** \(\text{halted} : A \to \text{Prop}\).
- **Hypothesis** \(\text{haltedP} : \forall t t' : A, \text{halted } t \to \text{step } t t' \to \text{False}\).

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

**Inductive** better_response_step \(N A : \{\text{ffun } [N] \to A\} \to \{\text{ffun } [N] \to A\} \to \text{Prop} ~\triangleq ~
\begin{cases}
\text{better_response_step_progress } t (i : [N]) t'_i : \\
\text{mov es } i (t i) t'_i \to \\
\text{let } t' \triangleq \text{upd } i t t'_i \text{ in } \\
\text{cost } i t' < \text{cost } i t \to \\
\text{better_response_step } t t'.
\end{cases}
\]

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 \(\text{moves}\) relation).

We define the reflexive transitive closure of the \(\text{step}\) relation as the following fixed point on the number of steps \(n\):

**Fixpoint** \(\text{stepN} (n : \text{nat}) : A \to A \to \text{Prop} ~\triangleq ~
\begin{cases}
\lambda t t' \Rightarrow \\
\text{if } n \text{ is } S n' \text{ then } \\
\exists t'', \lambda t t'' \& \text{stepN } n' t'' t'
\end{cases}
\]

\begin{cases}
\text{else } t = t'.
\end{cases}

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

\begin{verbatim}
Inductive step_star : A -> A -> Prop \equiv
  | step_refl t : step_star t t
  | step_trans t'' t t' :
    step t t'' \rightarrow
    step_star t'' t' \rightarrow
    step_star t t'.

Lemma stepN_step_star t t' : (\exists n, stepN n t t') \leftrightarrow step_star t t'.
\end{verbatim}

We say a state \( t : A \) is \texttt{safe}, as is standard, if

\begin{verbatim}
Definition safe t \equiv
  \forall t'', \text{step_star} t t'' \rightarrow [\forall' \exists t', \text{step} t'' t' \mid \text{halted} t''].
\end{verbatim}

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 \texttt{step} relations.

A state \( t : A \) \texttt{everywhere halts} if every state it could possibly reach has at least one path to a \texttt{halted} state:

\begin{verbatim}
Definition everywhere_halts (t : A) \equiv
  \forall t'', \text{step_star} t t'' \rightarrow
  \exists t', [\land \text{step_star} t'' t' \& \text{halted} t'].
\end{verbatim}

By contrast, we say a state \( t \) \texttt{somewhere halts} if there exists a halting path from state \( t \):

\begin{verbatim}
Definition somewhere_halts (t : A) \equiv
  \exists t', [\land \text{step_star} t t' \& \text{halted} t'].
\end{verbatim}

It naturally follows that if a state satisfies \texttt{everywhere_halts} then it also satisfies \texttt{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 \texttt{step} relation that satisfies certain properties, by mapping \texttt{step} to a new operational semantics \texttt{hstep}, for “\texttt{step} with history”, that tracks the history of states visited at each point in an execution.

We build \texttt{hstep} within a section parameterized by a game over actions of type \( A \):

\begin{verbatim}
Section history.
  Context \{ A \} \{\text{gameClass : game A}\}.
  Notation state \equiv (\text{ffun} [N] \rightarrow A).
  Variable step : state \rightarrow state \rightarrow Prop.
  (*\ldots\*)
\end{verbatim}

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

\begin{verbatim}
Let hstate \equiv (\text{simp\_pred state} \ast \text{simp\_pred state} \ast \text{state})\%type
\end{verbatim}
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 \texttt{hstep} relation is defined as:

\[
\text{Inductive} \ hstep : \text{hstate} \to \text{hstate} \to \text{Prop} \triangleq \\
\text{let} : s' \triangleq \text{predU} t' \quad \text{in} \\
\text{let} : u' \triangleq \text{predD} u t' \quad \text{in} \\
\quad u t' \to \\
\quad \text{step} t t' \to \\
\quad \text{hstep} (s,u,t) (s',u',t').
\]

where \( \text{predU} t' \) is the predicate corresponding to the set \( \{ t' \} \cup s \) while \( \text{predD} u t' \) corresponds to the set \( u - \{ t' \} \). That is, we can take an \texttt{hstep} from \( (s,u,t) \) to \( (s',u',t') \) as long as \( t' \) is unvisited \( u t' \), \texttt{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 \texttt{hstate}:

\[
\text{Definition inv} (s,u,t) : \text{Prop} \triangleq \\
\text{let} : (s,u,t) \triangleq \text{hstate} \text{in} \\
\quad \bigwedge s u t \quad \forall \lambda x \Rightarrow x \in (\text{enum state} \quad \text{(*Condition 1*)}) \\
\quad \text{predI} s u \quad \lambda \Rightarrow \text{false} \quad \text{(*Condition 2*)} \\
\quad \text{& s t} \quad \text{(*Condition 3*)}.
\]

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

\[
\text{Definition init} (t : \text{state}) \triangleq (\text{pred1} t, \text{predD1} \text{predT} t, t).
\]

parameterized by some underlying initial \text{state} \( t \) satisfies this invariant, for example. The predicate \( \text{pred1} t \) is the singleton set \( \{ t \} \). An \texttt{hstate} \( s,u,t \) is halted:

\[
\text{Definition hstep_halted} (s,u,t) \triangleq \\
\quad \bigvee \text{halted} \quad \text{let} : (s,u,t) \quad \text{in} \\
\quad \#|u| = 0
\]

when either the underlying state \( t \) is halted \( \text{halted} \) or the size of the unvisited set is 0 (there are no more states to visit).

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

\[
\text{Variable} P : \text{hstate} \to \text{Prop}. \\
\text{Hypothesis init_P} : \forall t, P \text{ (init } t). \\
\text{Hypothesis step_P} : \\
\quad \forall s u t t', \\
\quad \text{inv} (s,u,t) \to P \quad (s,u,t) \to \text{step} t t' \to \\
\quad \bigwedge u t' \quad \text{& P (predU} t' \quad s, \text{predD} u t', t'\big).
\]
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’:**

\[ \text{inv sut} \rightarrow \text{P sut} \rightarrow \text{step star hstep sut sut’} \rightarrow \text{[\& inv sut’ & P sut’].} \]

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’:**

\[ \text{inv (su,t)} \rightarrow \text{P (su,t)} \rightarrow \text{step t t’} \rightarrow \exists su’, [\& \text{hstep (su,t) (su’,t’) & P (su’,t’).}] \]

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 ≜**

\[ \text{let: (s,u,t) ≜ sut in} \]
\[ \forall t_0, s t_0 \rightarrow \Phi t \leq \Phi t_0. \]

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 (Φ t ≤ Φ 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 :**

\[ \text{safe hstep hstep_halted sut} \rightarrow \text{everywhere_halts hstep_halted sut.} \]

—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 :**

\[ \text{safe step_halted t} \rightarrow \text{inv (s,u,t)} \rightarrow \text{P (s,u,t)} \rightarrow \text{everywhere_halts hstep_halted (s,u,t)} \rightarrow \text{everywhere_halts step_halted t.} \]
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 everytime halts or stuck sut :
  safe hstep hstep halted sut →
  everywhere halts 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) →
  P (s,u,t) →
  safe step halted t →
  safe hstep halted (s,u,t).

The formal proof that step terminates (assuming stepP):

Theorem step everytime halts or stuck t :
  safe step halted t →
  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 ≜ (pred1 t, predD1 predT t, t). State init t also satisfies inv (because it is initial) and P (by initP). By the previously proved lemma everywhere halts hstep step, it suffices to prove that

  everywhere halts hstep halted (init t),

which itself follows by lemma hstep everytime 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 \( \text{num\,players} \), the graph \( g \) as an adjacency matrix, the codomain of the cost function \( R \), the cost functions \( \text{ecosts} \) associated with each edge in the graph, and a proof \( \text{ecosts\_pos} \) that the cost functions are positive.

**Section AtomicRoutingGame.**

**Variable** \( T : \text{finType} \).

**Variable** \( \text{num\,players} : \text{nat} \).

**Variable** \( g : \text{`M[bool]}(\#|T|, \#|T|)\).

**Variable** \( R : \text{realFieldType} \).

**Variable** \( \text{ecosts} : \forall x \, y : \#|T|, \text{nat} \rightarrow R \).

**Hypothesis** \( \text{ecosts\_pos} : \) \( \forall x \, y \, n, (0 \leq \text{ecosts} \, x \, y \, n)\%R. \)

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 \([0, \#|T|]\). We represent the graph \( g : \text{`M[bool]}(\#|T|, \#|T|) \) as an adjacency matrix mapping each pair of vertices to a boolean value indicating whether or not there is an edge between them. The \( \text{ecosts} \) function takes (the indices of) two vertices as arguments, along with the traffic on that edge (type \( \text{nat} \)), and returns the cost (type \( R \)).

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

**Record** \( \text{player} : \text{Type} \triangleq \) \( \text{mkPlayer} \{ \text{source} : \#|T|; \text{sink} : \#|T| \} \).

and the set of all players as a function from player indices \([\text{num\,players}]\) to player records:

**Variable** \( \text{players} : [\text{num\,players}] \rightarrow \text{player} \).

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 \( \text{sspath\_pred} \, x \, y \), standing for “source–sink path from vertex \( x \) to \( y \)”:

**Definition** \( \text{path} \triangleq ([\#|T|]^{\#|T|})\%\text{type} \).

**Fixpoint** \( \text{sspath\_rec} \, n \, (x \, y : \#|T|) : \text{pred} \, \text{path} \triangleq \)

\[
[pred \, p \, : \, \text{path} \\
| \text{[]} \] \[| \[
| \text{[]} \text{&} \text{&} \text{p \, x} \, = \, = \, y \, \text{&} \text{g \, x} \, (p \, x) \\
| \text{[]} \text{&} \text{g \, x} \, (p \, x) \, \text{&} \, \text{if} \, n \, \text{is} \, n' \, \, + \, \, 1 \, \text{then} \, \text{sspath\_rec} \, n' \, (p \, x) \, y \, p \, \text{else} \, \text{false}]\).

**Definition** \( \text{sspath\_pred} \, (x \, y : \#|T|) \triangleq \text{sspath\_rec} \, \#|T| \, x \, y \).

A path satisfies $\text{sspath}_\text{rec} \ n \ x \ y$ (and therefore $\text{sspath}_\text{pred} \ x \ y$, 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 ($[\land \land p \ x \ == \ y \ \& \ g \ x \ (p \ x)]$), or (2) $n$ is greater than 0 and $p$ maps vertex $x$ to an available vertex $p \ x$ such that $\text{sspath}_\text{rec} \ (n - 1) \ (p \ x) \ y \ (p \ x, y$ is recursively a valid path). The type $\text{strategy} \ i$:

- Notation $\text{src} \ i \triangleq (\text{source} \ (\text{players} \ i))$.
- Notation $\text{snk} \ i \triangleq (\text{sink} \ (\text{players} \ i))$.

**Definition** $\text{strategy} \ i : \ [\text{num}\_\text{players}] \triangleq \text{sig} \ (\lambda \text{the}\_\text{path} \Rightarrow \text{sspath}_\text{pred} \ (\text{src} \ i) \ (\text{snk} \ i) \ \text{the}\_\text{path})$.

packages together in a sigma type player $i$'s path with a proof that the path satisfies $\text{sspath}_\text{pred} \ (\text{src} \ i) \ (\text{snk} \ i)$.

The $\text{sspath}_\text{pred}$ 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 $\text{sspath}_\text{rec}$.

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** $\text{strategy}_\text{pkg} \triangleq \{ \{i : [\text{num}\_\text{players}] \& \text{strategy} \ i\} \}$.
- Notation $\text{st} \triangleq ((\text{strategy}_\text{pkg} \% \text{num}\_\text{players}) \% \text{type})$.

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** $\text{traffic}_\text{edge} \ (s : \text{st}) \ (x \ y : [\#|T|]) : \text{nat} \triangleq \#\text{edgePlayers} \ s \ x \ y$.

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

- **Definition** $\text{edgeOfPlayer} \ i \ (s : \text{st}) \ (x \ y : [\#|T|]) \triangleq \text{path}\_\text{of} \ s \ i \ x \ == \ y$.
- **Definition** $\text{edgePlayers} \ (s : \text{st}) \ (x \ y : [\#|T|]) : \text{pred} \ [\text{num}\_\text{players}] \triangleq [\text{pred} \ i \ | \ \text{edgeOfPlayer} \ i \ s \ x \ y]$.

The predicate $\text{edgeOfPlayer} \ i \ s \ x \ y$ 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 $\text{ecosts}$, as a function of the traffic over that edge:

- **Definition** $\text{cost}_\text{edge} \ (s : \text{st}) \ (x \ y : [\#|T|]) \triangleq \text{ecosts} \ x \ y \ (\text{traffic}_\text{edge} \ s \ x \ y)$.

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

- **Definition** $\text{costFun} \ (i : [\text{num}\_\text{players}]) \ (s : \text{st}) : \mathbb{R} \triangleq \sum_{x : [\#|T|]} (\text{cost}_\text{edge} \ s \ x \ y) \ \text{if} \ \text{edgeOfPlayer} \ i \ s \ x \ y \ \text{then} \ \text{cost}_\text{edge} \ s \ x \ y \ \text{else} \ 0$. 
Instance costInstance : CostClass num_players \( \mathbb{R} \) [finType of strategy_pkg]
\( \triangleq \) costFun.

**Program**
Instance costAxiomInstance : CostAxiomClass costInstance. (*proof elided*)

The costs given by \( \text{costFun} \) 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 \((i : \text{num_players})\) : rel strategy_pkg \( \triangleq \)
\[ \lambda p p' : \text{strategy_pkg} \Rightarrow \text{projT1} p = \text{projT1} p' \].

Instance movesInstance : MovesClass num_players [finType of strategy_pkg] \( \triangleq \) movesFun.

Instance gameInstance : game costAxiomInstance movesInstance.

This definition of movesInstance ensures that players only ever update their strategies, never the values \( i : \text{num_players} \) 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 \((s : \text{st})\) : \( \mathbb{R} \) \( \triangleq \)
\[
\sum_{x : \text{#T}} \sum_{y : \text{#T}} \sum_{1 \leq z < \text{traffic\_edge s x y}.+1} \text{ecosts x y z}.
\]

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:

\[ \text{phiFun } s' - \text{phiFun } s = \text{cost } i s' - \text{cost } i s. \quad (*\text{Potential Equation}*) \]

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 \((i : \text{num_players})\) \((x y : \text{[#T]})\) \( s s' : \text{Move } i s s' \rightarrow \text{edgeOfPlayer } i s' x y = \text{false} \rightarrow \text{edgeOfPlayer } i s x y = \text{false} \rightarrow \text{traffic\_edge s' x y = traffic\_edge s x y}.

**Lemma** traffic01 \((i : \text{num_players})\) \((x y : \text{[#T]})\) \( s s' : \text{Move } i s s' \rightarrow \text{edgeOfPlayer } i s' x y = \text{false} \rightarrow \text{edgeOfPlayer } i s x y \rightarrow (\text{traffic\_edge s' x y}).+1 = \text{traffic\_edge s x y}.

**Lemma** traffic10 \((i : \text{num_players})\) \((x y : \text{[#T]})\) \( s s' : \text{Move } i s s' \rightarrow \text{edgeOfPlayer } i s' x y \rightarrow \text{edgeOfPlayer } i s x y \rightarrow (\text{traffic\_edge s' x y}).+1 = \text{traffic\_edge s x y}.

**Lemma** traffic11 \((i : \text{num_players})\) \((x y : \text{[#T]})\) \( s s' : \text{Move } i s s' \rightarrow \text{edgeOfPlayer } i s' x y \rightarrow \text{edgeOfPlayer } i s x y \rightarrow \text{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 \(\text{movesFun}\) 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 traffic\(_{10}\) 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:

\[
\sum_{1 \leq z < (\text{traffic}_\text{edge} s' x y).+1} \text{ecosts} x y z - \sum_{1 \leq z < (\text{traffic}_\text{edge} s x y).+1} \text{ecosts} x y z = \text{ecosts} x y (\text{traffic}_\text{edge} s' x y) - 0
\]

which by traffic\(_{10}\) can be rewritten in terms of \(s\) as:

\[
\sum_{1 \leq z < (\text{traffic}_\text{edge} s x y).+2} \text{ecosts} x y z - \sum_{1 \leq z < (\text{traffic}_\text{edge} s x y).+1} \text{ecosts} x y z = \text{ecosts} x y (\text{traffic}_\text{edge} s x y).+1 - 0
\]

The left-hand side equals:

\[
\text{ecosts} x y (\text{traffic}_\text{edge} s x y).+1
\]

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

\[
\sum_{1 \leq i < m+.+1} f i - \sum_{1 \leq i < m} f i = f m
\]

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:

\[
\text{Instance PhiAxiomInstance : PhiAxiomClass phiInstance } \triangleq (\ast \ldots \ast).
\text{Instance AtomicPotentialInstance : Potential PhiAxiomInstance.}
\]

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 \(\text{PNE } t\)):

**Lemma** AtomicRouting\_exists\_PNE \((t_0 : \text{st}) : \exists t : \text{st}, \text{PNE } t\).

**Proof.** by apply: (\(\text{exists\_PNE } t_0\)). Qed.

**Lemma** AtomicRouting\_everywhere\_halts \((t : \text{st}) : \text{everywhere\_halts step halted } t\).

**Proof.** by apply: better\_response\_everywhere\_halts. Qed.

### 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 \(T : \text{finType}\) and the number of players \(\text{num\_players} : \text{nat}\).
A Library for Algorithmic Game Theory in Ssreflect/Coq

\[ c(x) = 1.5x \]

\[ 0.5x = 2.5 \]

\[ 6x = 2 \]

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 \( T : \text{finType} \). (** 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 \( \triangleq \{ \text{ffun } T \to \text{bool} \} \).

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' \( \triangleq \{ f : \text{ffun } T \to \text{bool} \mid \exists t : T. \ f \ t = \text{true} \} \)

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 \( \triangleq \)

\{ aCoeff : \text{rat};
   bCoeff : \text{rat};
   aCoeff_positive : 0 \leq aCoeff;
   bCoeff_positive : 0 \leq bCoeff \}\).

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

Variable costs : \{ \text{ffun } T \to \text{affineCostFunction} \}.

Definition evalCost \( t : T \) \( x : \text{nat} \) : \text{rat} \( \triangleq \)

\[ a\text{Coeff} \ (\text{costs } t) \ast x + b\text{Coeff} \ (\text{costs } t) \].

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 $st \doteq \{(\text{ffun \ [num\_players] \rightarrow \text{strategy}})\}$

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** $\text{load} \ (s \ : \ st \ (t \ : \ T) : \ nat \ \doteq \ |\{ \text{set \ } i \ | \ s \ i \ t \}|$.

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** $\text{costFun} \ (i : \text{[num\_players]}) \ (s : \text{st}) : \text{rat} \ \doteq \ \sum_t \text{if} \ s \ i \ t \ \text{then} \ \text{evalCost} \ (\text{load} \ s \ t) \ \text{else} \ 0$.

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

**Definition** $\text{movesFun} \ (i : \text{[num\_players]}) : \text{rel \ strategy} \ \doteq \ [\lambda \ : \ \text{strategy} \Rightarrow \text{true}]$.

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]:

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

which we formalize as the lemma:

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

In our statement of $\text{christodoulou}$, the ubiquitous $\%:Q$ simply coerces natural numbers to $Q$. 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_AG2`.

Smoothness of the game follows from a consequence of $\text{christodoulou}$ 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 $\epsilon$-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


