Organizing

Computational Problem Solving Communities

via

Collusion-Resistant Semantic Game Tournaments

A dissertation presented

by

Ahmed Abdelmeged

to the Faculty of the Graduate School

of the College of Computer and Information Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Northeastern University

Boston, Massachusetts

May, 2014

ABSTRACT

Competitions have been successfully used to build and organize communities aiming to solve complex computational problems. Recently, the winner of an open online competition to solve a complex immunogenomics problem has produced an algorithm that has a higher accuracy and is 1,000 times faster than the corresponding algorithm developed by the US National Institutes of Health.

State of the art computational problem solving competitions follow the contest pattern where a *trusted* administrator is responsible for preparing an objective, anonymous, neutral, correct and thorough process to evaluate participants. Depending on the computational problem of interest, it can be a massive undertaking for the administrator to prepare such evaluation process. For example, consider the effort involved in collecting 2.86 Gigabytes worth of compressed benchmarks for the SAT 2013 competition.

The thesis of this dissertation is "semantic games of interpreted predicate logic sentences provide a useful foundation for organizing computational problem solving communities". Semantic games of interpreted predicate logic sentences specifying computational problems provide a *correct*, systematic approach for participants to *assist* the administrator in evaluating their opponents and therefore require a significantly lower administrator overhead.

In this dissertation, we tackle a key challenge of fending against collusion potential which *can* render the results of semantic game tournaments meaningless. We report on the surprising discovery that it is in fact possible to fend against collusion potential in semantic game tournaments by using certain ranking functions. We give a, first of its kind, formal characterization of collusion-resistant ranking functions for semantic game tournaments. We also give a *representation theorem* of ranking functions possessing the limited collusion effect property as well as other basic monotonicity properties. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting.

In this dissertation, we also sketch out a structured interaction space that we specifically designed to hold semantic-game-based computational problem solving competitions. The purpose is to give our readers a concrete sense of what is it like to organize or to participate in a semantic-game-based computational problem solving competition.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Competitions have been successfully used to build and organize communities aiming to solve complex computational problems. A recent success of this paradigm is demonstrated by the following quote from a recent Nature publication [20]: “Historically, prize-based contests have had striking success in attracting unconventional individuals who can overcome difficult challenges. To determine whether this approach could solve a real big-data biologic algorithm problem, we used a complex immunogenomics problem as the basis for a two-week online contest broadcast to participants outside academia and biomedical disciplines. Participants in our contest produced over 600 submissions containing 89 novel computational approaches to the problem. Thirty submissions exceeded the benchmark performance of the US National Institutes of Health’s MegaBLAST. The best achieved both greater accuracy and speed (1,000 times greater).”

There are numerous other examples of competitions organized to encourage the research and development on computational problem solving. Examples include the SAT competition [4] organized to encourage the development of high performance SAT solvers. Examples also include software development competitions held on platforms such as TopCoder [30]. There are also numerous other examples of software development competitions organized for educational purposes such as competitions held on platforms such as Project Euler [2] and Jutge [23].

A computational problem solving competition produces either a rating or a ranking of the problem solving skills of its participants. A community aiming to solve a computational problem can benefit from the result of a competition held between its members in a number of ways. First, to motivate community members to develop skills related to solving the underlying computational problem. The competition result is often used to objectively distribute a prize. Even with no prize to distribute, the announcement of competition results can be rewarding to some participants. Second, to effectively diffuse problem solving knowledge. The knowledge of top ranked participants can be shared with other community members to learn from before the next competition starts. This is called “leveling-the-boats” and is a quite common practice. Finally, sponsors may be interested in the competition results in order to hire top ranked participants or to use the solutions they provide during competitions. Therefore, sponsors may offer a prize to attract more high quality members to the community.

For a competition to successfully attract participants and sponsors, it is crucial that the competition organizer or administrator provides a guarantee that the competition result reflects an *accurate* assessment of the skill level of participants. For this to happen, a competition must have the following five features: objectivity, anonymity, neutrality, monotonicity and thoroughness. Objectivity means that a precise definition of the skills that may be reflected in the competition result is included as a part of the competition definition. A typical definition of a computational problem solving skill is the ability to solve an instance of that computational problem. Anonymity means that the competition result is independent of participants’ identities and is solely based on skills that participants demonstrate their possession or lack during the competition. Neutrality means that the competition does not give an advantage to any of the participants. For example, a seeded tournament where the seed (or the initial ranking) can affect the final ranking is not considered neutral. Monotonicity means that a participant's competition result can only be positively affected by a demonstrated possession of skill, and can only be negatively affected by a demonstrated lack of skill. For example, in a competition where participants' skills are assessed using a benchmark of problems together with their reference solutions, monotonicity requires that the reference solutions are indeed correct. Thoroughness means that the competition result is based on a wide enough range of skills that participants demonstrate during the competition.

State of the art computational problem solving competitions follow the contest pattern where a *trusted* administrator is responsible for preparing an objective, anonymous, neutral, correct and thorough process to evaluate the skills of individual participants. Depending on the computational problem of interest, it can be a massive undertaking for the administrator to prepare such evaluation process. For example, consider the effort involved in collecting 2.86 Gigabytes worth of compressed benchmarks for the SAT 2013 competition [4]. In order to reduce their overhead and the amount of trust they request from the community, it is not uncommon for administrators to publish the entire evaluation process they develop to be scrutinized by community members. In the SAT competition, the administrator takes even a further step by making a call for benchmarks before the competition. However, the administrator is still responsible for coordinating the development efforts for the competition evaluation process. Furthermore, making the entire evaluation process publicly available can encourage participants to shift their goal from solving the underlying computational problem to solving the instances that they may encounter in the competition.

Compared to computational problem solving competitions, sports competitions put a smaller overhead on the administrator. In sports, the administrator's role, which is typically called a referee, is to ensure that participants follow a set of *easily checkable* rules. Participants essentially *assist* in evaluating their opponents. This sports-like approach was used in a historic computational problem solving competition held, in 1535, between Tartaglia and Fior to figure out who knows how to solve cubic equations more efficiently. The rules were that each provides the other with 30 cubic equations to solve and the faster wins. One imagines it was a relatively easy task to ensure that equations supplied by both Tartaglia and Fior were indeed cubic equations and solutions produced by both Tartaglia and Fior were indeed correct.

Unfortunately, direct adoption of this approach to more complex problems may defeat the purpose of reducing the overhead on the administrator. For example, consider the following two-party hypothetical competition to develop a correct SAT solver. Each party provides 30 CNF formulas to be solved by the SAT solver developed by their opponent. In this competition, the administrator would be required to check the correctness of solutions produced by both SAT solvers. When one of the solvers claims a particular CNF formula to be unsatisfiable, the administrator has to check that the CNF formula is indeed unsatisfiable. It is however a much harder task than it is to check the correctness of some claimed solution to some cubic equation.

In this dissertation, we aim to develop an approach to organize computational problem solving communities using competitions where participants assist in evaluating their peers yet remain meaningful. In the rest of this chapter, we present a thesis summarizing our approach together with the rationale of our thesis. Then, describe the key challenge facing the development of our approach as well as our contributions. Finally, we describe the organization of this dissertation.

1.1 Thesis Statement and Rationale

Our thesis is: "Semantic games of interpreted logic sentences provide a useful foundation to organize computational problem solving communities.". In the following few paragraphs, we briefly introduce semantic games and discuss the rationale behind our thesis.

A Semantic Game (SG) is a constructive debate of the truth of some interpreted logic sentence, or claim for short, between two distinguished parties: the verifier which asserts that the claim holds, and the falsifier which asserts that the claim does not hold. The rules of an SG are systematically derived from the syntax of the underlying claim. The rules of an SG only allow the verifier to strengthen the current claim and only allow the falsifier to weaken the current claim. Each legal move reduces the number of logical connectives in the current claim. The verifier wins if a $true$ primitive claim is eventually reached. Otherwise, a $false$ primitive claim is eventually reached. An SG gives a meaning to its underlying claim in the sense that the underlying claim is $true$ (respectively $false$) if and only if there is a winning strategy for the verifier (respectively falsifier).

We now illustrate SGs by an example. Consider the following logical sentence specifying the MAXimum SATisfiability (MAX-SAT) problem: $\forall \phi \in CNFs\ \exists v \in assignments(\phi) \forall f \in assignments(\phi).\ fsat(f,\phi) \leq fsat(v,\phi)$. This logical sentence is interpreted in a structure that defines all non-logical symbols; namely, $CNFs$, $assignments$, $fsat$ and $\leq$. The details of the structure specification are well known and we omit them here but they are an essential part of the claim specification just like the claim's logical formula. Before an SG can be played, the two participants pick their sides in the debate. For now, we assume they picked opposite sides, although the underlying logical sentence is trivially true and therefore, by definition of SGs, there is a winning strategy for the verifier. An SG played on this claim proceeds as follows:

1. The falsifier provides a CNF formula ϕ_0. If the administrator determines, according to the interpreting structure, that ϕ_0 is not well-formed (i.e. $\phi_0 \notin CNFs$), the falsifier loses at once. By providing ϕ_0, the falsifier effectively weakens the underlying claim to: $\exists v \in assignments(\phi_0) \forall f \in assignments(\phi_0).\ fsat(f, \phi_0) \leq fsat(v, \phi_0)$.

2. Given ϕ_0, the verifier provides an assignment v_0 for the variables in ϕ_0. If the administrator determines, according to the interpreting structure, that v_0 is not well-formed (i.e. $v_0 \notin assignments(\phi_0)$), the verifier loses at once. By providing v_0, the verifier effectively strengthens the underlying claim to: $\forall f \in assignments(\phi_0).\ fsat(f, \phi_0) \leq fsat(v_0, \phi_0)$.

3. Given ϕ_0 and v_0, the falsifier provides an assignment f_0 for the variables in ϕ_0. If the administrator determines, according to the interpreting structure, that f_0 is not well-formed (i.e. $f_0 \notin assignments(\phi_0)$), the falsifier loses at once. By providing f_0, the falsifier effectively weakens the underlying claim to: $fsat(f_0, \phi_0) \leq fsat(v_0, \phi_0)$.

4. The administrator evaluates, according to the interpreting structure, the primitive claim $fsat(f_0, \phi_0) \leq fsat(v_0, \phi_0)$. The verifier wins if this primitive claim evaluates to *true* otherwise the falsifier wins. (i.e. the verifier wins if the assignment v_0 it provided satisfies at least as many clauses as those satisfied by the assignment f_0 provided by the falsifier to the formula ϕ_0 provided by the falsifier).

The rationale behind our thesis is that an SG can be used as a two-party competition that has several desirable features that we shall illustrate using our MAX-SAT example:

- An SG is an *objective* competition. In an SG, participants are required to provide test inputs as well as to solve instances of computational problems that are *formally* specified by the claim The result of an SG solely depends on how well participants solve these computational problems. A simple analysis of our MAX-SAT example shows that it is in the best interest of the verifier to provide an assignment v_0 that satisfies the maximum satisfiable fraction of clauses. A similar analysis shows that it is in the best interest of the falsifier to provide an assignment f_0 that satisfies more than v_0 does, when possible. It is also in the best interest of the falsifier to provide a CNF formula ϕ_0 where it is hard for the verifier to find the assignment satisfying the maximum fraction of satisfiable clauses. In Section 3.3, we give further details on how administrators can formulate claims such that participants are required to solve specific computational problems.

- SGs can be carried out with a minimal overhead on the administrator. The administrator is essentially responsible for implementing the structure in which the underlying logical sentence is interpreted. It is always possible to rewrite the underlying logical formula scraping some of the quantifiers either in or out of the structure and consequently adding or reducing overhead on the administrator. The reduced responsibility of the administrator makes it easier for participants and sponsors to *trust* the competition correctness. Furthermore, the structure is only a part of the evaluation process that can be published without encouraging participants to shift their goal to solving the instances they may be required to solve during the competition. It is the opponent's responsibility to provide the other part of the evaluation process consisting of test inputs. In our MAX-SAT example, the administrator only has to implement functions to check whether a given formula is indeed a well-formed CNF formula, check whether an assignment is indeed a correct assignment for the variables in a given formula, compute the fraction of clauses of some CNF formula satisfied by some assignment, and compare two such fractions. All of these are relatively easy tasks for the administrator to implement. We may rewrite the underlying formula scraping the right most quantifier into the structure as: $\forall\phi \in CNFs\ \exists v \in assignments(\phi)$. *max-sat*$(\phi, v)$. By doing so, the administrator becomes responsible for the much harder task of implementing a checker of whether an assignment is indeed satisfying the maximum satisfiable fraction in a given CNF formula.

- An SG is a *correct* competition. In an SG, the winner is the participant that successfully demonstrates their opponent's lack of skills. If the verifier wins, then the underlying claim is either a $true$ claim indeed or that the falsifier has weakened the claimed too much during game play. Likewise, if the falsifier wins, then the underlying claim is either a $false$ claim indeed or that the verifier has strengthened the claim too much during game play. In our MAX-SAT example, the underlying claim is indeed $true$. The falsifier weakens the claim by providing CNF formula ϕ_0. However, any weaker claim remains $true$. The verifier then strengthens the current claim by providing an assignment v_0 for the variables of ϕ_0. Any assignment provided by the verifier satisfying less than the maximum fraction of satisfiable clauses in ϕ_0 would strengthen the current claim too much. Should this happen, the falsifier has a chance to demonstrate the verifier's lack of skill by providing an assignment that satisfies a larger fraction of clauses in ϕ.

- SG traces contain concrete targeted feedback that losers can learn from. In our MAX-SAT example, suppose that the falsifier has provided a CNF formula ϕ and an assignment f that satisfies a larger fraction than the assignment v provided by the verifier. The verifier can analyze the assignment f and update its future course of action accordingly.

- In SGs, participants interact through well defined interfaces. This enables participants to codify their strategies [1] for participating in an SG-based computational problem solving competitions. These codified strategies enable a more efficient and effective evaluation. Furthermore, the algorithms in these codified strategies can also be useful byproducts of SG-based computational problem solving competitions. In our MAX-SAT example, a codified strategy for participating in an SG-based computational problem solving competition consists of a function to pick a side in SGs of the aforementioned MAX-SAT claim, a function to provide a CNF formula, a MAX-SAT solver (i.e. a function to provide an assignment satisfying the maximum fraction of clauses in a given CNF formula), and a function that is given a CNF formula and an assignment and produces another assignment that satisfies, in the given formula, at least as many clauses as the given assignment.

In support of our thesis, we developed an approach to scale the aforementioned simple two-party, SG-based competition to an n-party, SG-based competition that is objective, anonymous, neutral, monotonic, thorough and puts a minimal overhead on the administrator. Developing an n-party, SG-based competition that is guaranteed to be anonymous is the most challenging issue that we faced. In the next section, we discuss this issue as well as the contributions of this dissertation.

[1] Provided that all functions in these strategies are indeed computable.

1.2 Key Challenge and Contributions

In sports, tournaments are commonly used to scale two-party games to n-party games. Likewise, a tournament can be used to scale two-party SGs to n-party games. An SG tournament is objective by virtue of being comprised of a number of objective SGs. Similarly, organizing a tournament is a relatively simple additional task for the administrator.

The contributions of this dissertation can be summarized as developing a thorough, arguably neutral and provably anonymous and monotonic SG tournament. More specifically we claim the following contributions:

1. Computational Problem Solving Labs (CPSLs): CPSLs are structured interaction spaces that we specifically designed to organize SG-based computational problem solving competitions. A CPSL is centered around a claim specifying the lab's computational problem. Competition participants submit their codified strategies for playing a more thorough, yet simpler, version of semantic games. The submitted strategies compete in a thorough, arguably neutral and provably anonymous and monotonic tournament.

2. Simplified Semantic Games (SSGs): A sports match is generally considered as a thorough two person competition because in a sports match, participants get to exercise a wide range of skills, enough to ensure that the match result is accurate. In an SG, however, participants get essentially a single chance to demonstrate a lack of skills of their opponents. A double round robin tournament of SGs is not necessarily thorough because it gives every participant only two chances to demonstrate a lack of skills of any other participant. To address this issue, we developed a simplified version of semantic games that allows participants to provide several values for a quantified variable

and therefore can be considered thorough. SSGs also use auxiliary games to replace some of the moves that participants would normally be required to take and thus simplify the process of codifying strategies. Further details are given in Section 3.2.

3. Collusion-resistant ranking functions: Ensuring anonymity of the ranking resulting from an SG tournament is quite challenging due to collusion potential. Collusion is possible because participants can arrange to identify their colluding participants through the values they produce in the course of playing an SG. In our MAX-SAT example, participants may, for example, use specific variable names to disclose their identities to other colluding participants. Colluding participants can arrange to lose on purpose for the benefit of a specific participant and thus affecting the accuracy of the competition result. In Section 4.1 we give a detailed example showing how collusion can affect the result of a competition. The potential of collusion can be further aggravated by the potential of participating in a computational problem solving competition using Sybil identities.

   A surprising discovery to us is that in SG tournaments, it is in fact possible to fend against collusion potential by using certain ranking functions. In this dissertation, we give a, first of its kind, formal characterization of collusion-resistant ranking functions for SG tournaments. We also give a *representation theorem* of ranking functions possessing the limited collusion effect property as well as other basic monotonicity properties. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting. We also present an SG-based tournament and argue for its objectivity, anonymity, neutrality, monotonicity and thoroughness.

4. Neutral SSG tournament design: Individual SGs of interpreted predicate logic sentences are *not* neutral because the underlying claim is either true or false and consequently either the verifier or the falsifier has a winning strategy. Double round robin tournaments have been traditionally used in sports to neutralize the advantages that an individual match gives to one of its participants such as having the match played at the home field of one of the participants. Likewise, a double round robin can be used to neutralize the advantage that an SG gives to either the verifier or the falsifier by having every pair of participants play two games where they exchange taking the verifier and the falsifier sides. Unfortunately, a monotonic, collusion-resistant ranking function has to completely ignore those games where both participants are forced to take sides opposing to the sides they choose. Since the number of participants choosing to take the verifier side is not necessarily the same as the number of participants choosing to take the falsifier side, verifiers and falsifiers participate in a different number of non-ignored games and neutrality is jeopardized. We argue that a round robin tournament remains neutral under a specific monotonic, collusion-resistant ranking function.

1.3 Organization

The rest of this dissertation is organized as follows: In Chapter 2 we give a background of SGs. In Chapter 3 we sketch out a structured interaction space that we specifically designed to organize computational problem solving communities using an SG-based competitions. The purpose is to give our readers a concrete sense of what is it like to organize or to participate in an SG-based competition. In Chapter 4, we present a, first of its kind, formal characterization of collusion-resistant ranking functions for SG tournaments and give a *representation theorem* of ranking functions possessing the limited collusion effect property as well as other basic monotonicity properties. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting. We also present an SG tournament and argue for its objectivity, anonymity, neutrality, monotonicity and thoroughness. Chapter 5 concludes this dissertation.

Chapter 2

Background: Semantic Games

In this chapter we describe Hintikka's standard SGs for classical predicate logic as background information on SGs.

We use the term "claim" to abbreviate the phrase "interpreted logical sentence". An SG is a formal two-party debate of the *truth* of an underlying claim. The two sides of the debate are called the *verifier* side and the *falsifier* side [1]; participants taking the verifier side assert that the claim is true. Participants taking the falsifier side assert that the claim is false.

In the theory of Game Theoretic Semantics (GTS) of Hintikka, SGs give meaning to claims [24], [19] in the following sense: a winning strategy for the *verifier* exists if and only if that the underlying claim is indeed *true* and a winning strategy for the *falsifier* exists if and only if the underlying claim is indeed *false*. Participants can use this to take a side in SGs by deciding whether or not the underlying claim holds. Participants deciding the underlying claim holds become verifiers. Participants deciding the underlying claim does not hold become falsifiers.

[1] other names have been also used in the literature such as *I* and *Nature*, *Proponent* and *Opponent*, *Alice* (female) and *Bob* (male), and $\exists$loise and $\forall$belard.

Φ	Move	Subgame
$\forall x : \Psi(x)$	fal provides x_0	$SG(\langle \Psi[x_0/x],\, A\rangle,\, ver,\, fal)$
$\Psi \wedge \chi$	fal chooses $\theta \in \{\Psi, \chi\}$	$SG(\langle \theta,\, A\rangle,\, ver,\, fal)$
$\exists x : \Psi(x)$	ver provides x_0	$SG(\langle \Psi[x_0/x],\, A\rangle,\, ver,\, fal)$
$\Psi \vee \chi$	ver chooses $\theta \in \{\Psi, \chi\}$	$SG(\langle \theta,\, A\rangle,\, ver,\, fal)$
$\neg\Psi$	N/A	$SG(\langle \Psi,\, A\rangle,\, fal,\, ver)$
$p(x_0)$	N/A	N/A

Table 2.1: Rules for $SG(\langle \Phi,\, A\rangle,\, ver,\, fal)$

The rules of SGs are prompted by the logical connectives encountered in claims. Table 2.1 shows the rules for first order logic proposed by Hintikka [19]. We use $SG(\langle \Phi,\, A\rangle,\, ver,\, fal)$ to denote an SG where the underlying claim is comprised of the formula Φ interpreted in the structure A and ver, respectively fal, denotes the participant currently taking the verifier, respectively falsifier, side. For universally quantified formulas $\forall x : \Psi(x)$, the current falsifier provides a value x_0 for the quantified variable x and the game proceeds on the logically weaker subclaim $\Psi[x_0/x]$ as $SG(\langle \Psi[x_0/x],\, A\rangle,\, ver,\, fal)$. For existentially quantified formulas $\exists x : \Psi(x)$, the current verifier provides a value for the quantified variable and the game proceeds on the logically stronger subclaim $\Psi[x_0/x]$ as $SG(\langle \Psi[x_0/x],\, A\rangle,\, ver,\, fal)$. For and-compounded formulas, the current falsifier chooses one of the subformulas for the game to proceed on. For or-compounded formulas, the current verifier selects a subformula. For negated formulas $\neg\Psi$, no moves are required and the game proceeds as $SG(\langle \Psi,\, A\rangle,\, fal,\, ver)$. Primitive formulas are evaluated in the underlying structure and the current verifier wins if they hold, otherwise the current falsifier wins.

We illustrate SG rules using the logical sentence $\forall x \in [0,1] \exists y \in [0,1].\ x \cdot y + (1-x)\cdot(1-y^2) \geq 0.5$ interpreted in the structure of real arithmetic. According to the rules, the falsifier is required to provide a value for the universally quantified variable x. Suppose that the falsifier provided 0 for x. The game then proceeds on the logically weaker claim $\exists y \in [0,1].\ (1-y^2) \geq 0.5$. According to the rules, the verifier is required to provide a value for existentially quantified variables. Suppose that the verifier provided 0, then the game proceeds on the logically stronger claim $1 \geq 0.5$. This is a true primitive claim in the structure of real arithmetic, therefore the verifier wins according to the rules.

Chapter 3

Organizing Computational Problem Solving Communities

In this chapter we sketch out a structured interaction space that we specifically designed to organize computational problem solving communities with minimal effort required by a central administrator. The purpose is to give our readers a concrete sense of what is it like to organize or to participate in an SG-based computational problem solving competition. We call our newly designed space, a Computational Problem Solving Laboratory (CPSL). A CPSL is centered around a claim, i.e. an interpreted predicate logic sentence, that formally specifies an underlying computational problem.

Members of a computational problem solving community contribute to a CPSL by submitting their strategies for playing an adapted version of semantic games of the lab's claim. We call our adapted version of semantic games, Simplified Semantic Games (SSGs). We specifically designed SSGs to simplify strategy development as well as to enable SSG participants to ensure that their opponents are *thoroughly* evaluated. We describe SSGs in Section 3.2.

Once a new strategy is submitted in a CPSL, it is entered into a competition against the strategies submitted by other members in a *provably* collusion-resistant tournament of SSGs. The outcome of the competition is a ranking that is announced to all members through their CPSL interface as we shall describe in Section 3.1. However, the competition design is the main subject of Chapter 4.

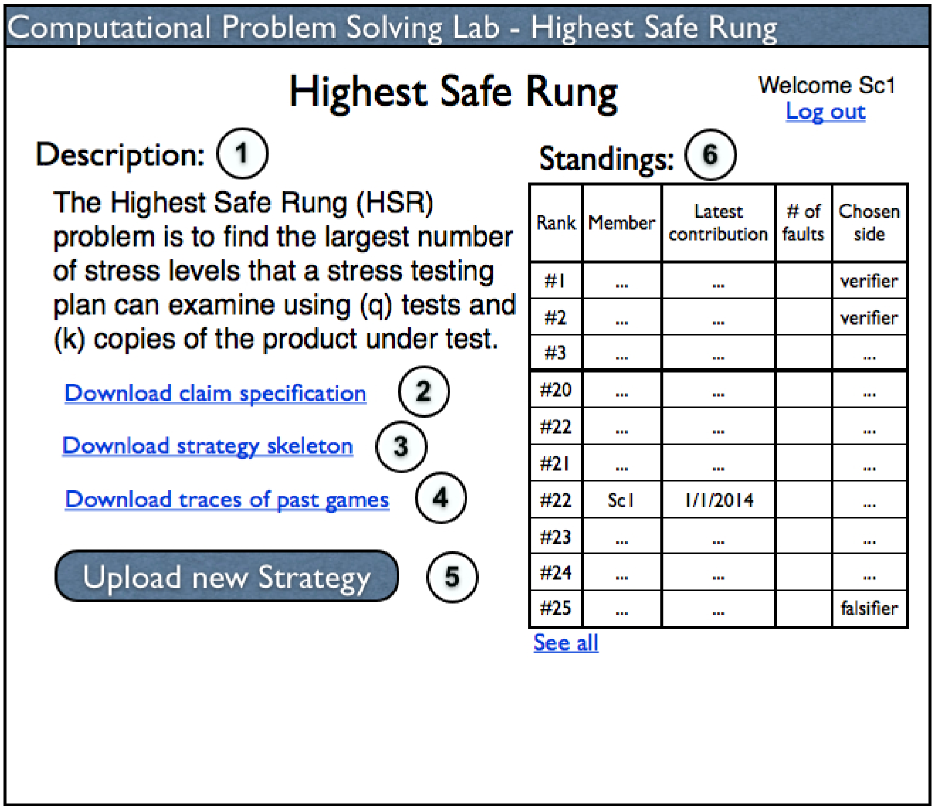
Following the CPSL introduction in Section 3.1 and the description of SSGs in Section 3.2, we describe how a CPSL is defined and contributed to in Sections 3.3 and 3.4. We then present few sample CPSLs in Section 3.5. Finally, we discuss work related to organizing computational problem solving communities in Section 3.6.

3.1 Computational Problem Solving Labs

In a nutshell, a CPSL is a structured interaction space centered around a claim, i.e. an interpreted predicate logic sentence, that formally specifies an underlying computational problem. Members of a computational problem solving community contribute to a CPSL by submitting their strategies for playing a simplified version of semantic games of the lab's claim. Once a new strategy is submitted in a CPSL, it competes against the strategies submitted by other members in a *provably* collusion-resistant tournament of simplified semantic games.

Figure 3.1 shows a mockup of a CPSL member's interface. The interface facilitates the following actions:

1. View an informal description of the CPSL's underlying problem.
2. Download the claim that formally specifies the CPSL's underlying problem. In Section 3.4, we further describe how claims are expressed.
3. Download a code skeleton of the strategy for playing a simplified version of semantic games of the lab's claim. In Section 3.4, we also describe strategies


Computational Problem Solving Lab - Highest Safe Rung
Welcome Sc1
Log out
Description: 1
Download claim specification 2
Download strategy skeleton 3
Download traces of past games 4
Upload new Strategy 5
Standings: 6
See all

Highest Safe Rung

The Highest Safe Rung (HSR) problem is to find the largest number of stress levels that a stress testing plan can examine using (q) tests and (k) copies of the product under test.

Rank	Member	Latest contribution	# of faults	Chosen side
#1	...	...		verifier
#2	...	...		verifier
#3	...	...		...
#20	...	...		...
#22	...	...		...
#21	...	...		...
#22	Sc1	1/1/2014		...
#23	...	...		...
#24	...	...		...
#25	...	...		falsifier

Figure 3.1: Mockup of a CPSL Member's Interface

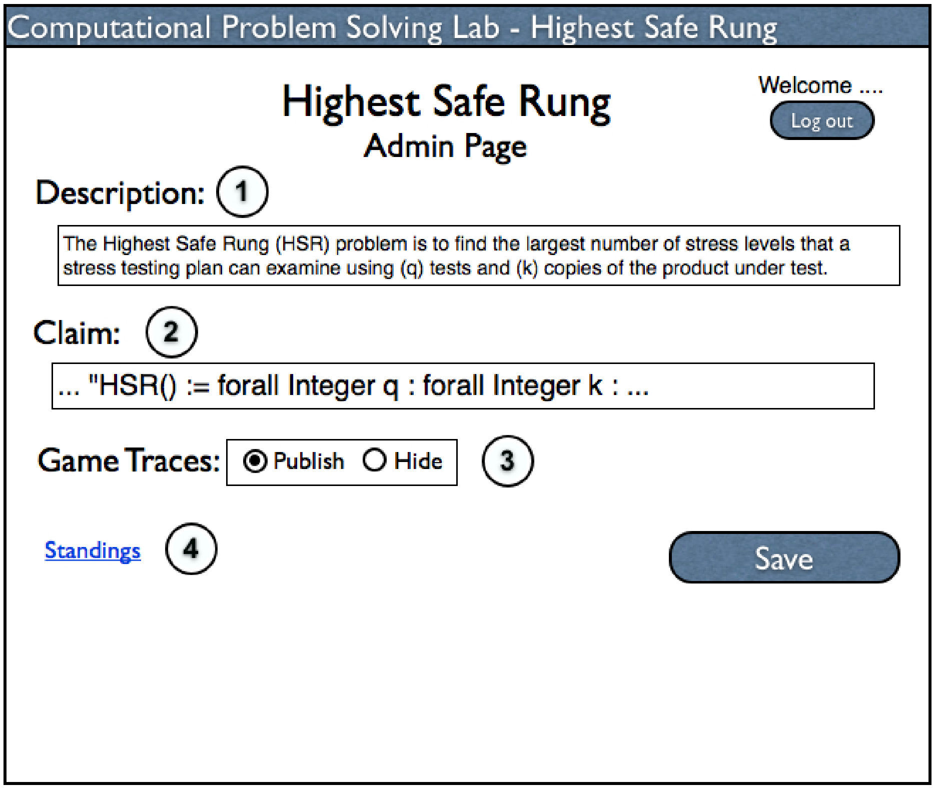
for playing a SSGs.

4. Download traces of the past games held in the lab, provided that the administrator has chosen to publish past game traces. These traces can help members improve their strategies.

5. Submit their strategies for playing a simplified version of semantic games of the CPSL's claim.

6. View the rankings of the most recently submitted strategy of each lab member. Besides the rank of each participant, two other pieces of information that can also be useful in assessing the quality of strategies. These two pieces are the time most recent update to the participant's strategy, and the number of

faults incurred in the most recent competition. A non zero number of faults means that the strategy is not perfect yet. Also, the further back in time the most recent update to the participant's strategy the more thoroughly tested the strategy is. Another useful piece of information is the side chosen by each participant. For CPSLs for claims about search and optimization problems, the correct side is often trivially known. Having this piece of information gives an additional assertion about the proceedings of the competition. In CPSLs for claims specifying a single decision problem instances, the side chosen by the top performing participants may indicate the solution of the underlying decision problem instance.



The Highest Safe Rung (HSR) problem is to find the largest number of stress levels that a stress testing plan can examine using (q) tests and (k) copies of the product under test.

... "HSR() := forall Integer q : forall Integer k : ...

Figure 3.2: Mockup of a CPSL Admin's Interface

Figure 3.2 shows a mockup of CPSL admin's interface. The interface facilitates the following actions:

1. Edit an informal description of the CPSL's underlying problem.

2. Edit the claim that formally specifies the CPSL's underlying problem. In Section 3.3, we further discuss claim formulation.

3. Control whether or not to publish past game traces. Participants can learn by examining past game traces. However, there are cases where traces may give away too much information about the solution of the lab's underlying computational problem such as in the Gale-Shapely worst case input lab described in Section 3.5. In those cases, the administrator may choose to not publish past game traces.

4. View the rankings of the most recently submitted strategy of each lab member.

We now describe an adopted version of Semantic Games that we base our competitions on.

3.2 Simplified Semantic Games

We designed SSGs to simplify strategy development as well as to enable SSG participants to ensure that their opponents are *thoroughly* evaluated. Strategies for playing SSGs are simpler to develop because they only need to provide values for universally and existentially quantified variables in the underlying claim. On the other hand, strategies for playing SGs need to take a side on the underlying claim, provide values for universally and existentially quantified variables in the underlying claim, and select a subformula in and-compounded and or-compounded formulas. Basically, SSGs make up for the missing moves through auxiliary SSGs. To decide the underlying claim, a participant plays an SSG against itself. To select a subformula, a participant decides the left subformula then selects a subformula accordingly. SSGs enable participants to ensure that their opponents are thoroughly evaluated because it is legal to provide *several* alternate values for universally and existentially quantified variables in the underlying claim. Thus making an SSG correspond to a collection of SGs, one SG for each possible choice of values provided for quantified variables in the underlying claim. The result of an SSG is a number $[0,1]$ denoting the fraction of those SGs that the verifier wins in the course of that SSG.

Table 3.1 specifies the rules for playing an SSG between two participants: a verifier *ver* and a falsifier fal. The rules show an SSG of a claim consisting of a logical formula Φ interpreted in a structure A. As we mentioned above, an SSG corresponds to a collection of SGs. The output of an SSG is the fraction of SGs won by the verifier.

The first rule applies to universally quantified formulas, i.e. formulas of the form $\forall x : \Psi(x)$. The falsifier is responsible for providing a set $\bar{x}$ of alternative values for x. For each provided value $x_i \in \bar{x}$, an SSG is played on the subformula Ψ with x_i substituted for the free occurrences of x in Ψ.

The second rule applies for and-compounded formulas, i.e. formulas of the form $\Psi \wedge \chi$. The falsifier is responsible for selecting one subformula, i.e. either Ψ or χ. To select a subformula, the falsifier decides the left subformula Ψ. If the falsifier decides the left subformula Ψ to be $false$, the selected subformula is Ψ. Otherwise, the selected subformula is χ. The rationale is that for the game to reach this point, the falsifier must have at some point taken a position implying that $\Psi \wedge \chi$ is $false$. To decide the left subformula Ψ, the falsifier plays an auxiliary SSG against itself with Ψ as the underlying claim. As mentioned above, this auxiliary

SSG corresponds to a collection of SGs. If the falsifier wins all of these SGs in the falsifier role, then Ψ is decided to be $false$. On the other hand, if the falsifier wins all of these SGs in the verifier role, then Ψ is decided to be $true$. However, if the falsifier wins only some of these games in either role, then there is no ground for deciding Ψ to be either $true$ or $false$. The falsifier is considered to have failed to decide Ψ and loses the SSG at once.

The third and forth rules apply to existentially quantified formulas and or-compounded formulas respectively and are quite similar to the first and second rules. Therefore we give no further explanation for these two rules.

The fifth rule applies for negated formulas, i.e. formulas of the form $\neg\Psi$. There are no moves required by the verifier nor by the falsifier. The SSG proceeds on the subformula Ψ with the verifier and falsifier exchanging their roles.

The sixth applies to primitive formulas, i.e. formulas of the form $p(\bar{x})$ where p is a primitive defined in the structure A and $\bar{x}$ is a vector of constants defined in the structure A. The verifier wins if $p(\bar{x})$ holds in A. Otherwise, the falsifier wins. The final rule applies to formula references, i.e. formulas of the form $p(\bar{x})$ where $p(\bar{x})$ refers to another claim Ψ. In this case, the SSG proceeds on Ψ. This rule enables modular specification of claims.

We now describe the process of formulating claims for CPSLs.

3.3 Formulating Claims

A CPSL is centered around a claim that is specified by the administrator. Claim specification is the task that requires the most overhead on the administrator side. Furthermore, claim specification also determines the overhead on CPSL members. Starting with a specific computational problem, there is usually a space of potential claim specifications for the administrator to choose from usually under the follow-

Φ	Move	Subgame
$\forall x : \Psi(x)$	fal provides $\bar{x}$	$1/\lvert\bar{x}\rvert \cdot \sum_{x_i \in \bar{x}} SSG(\langle \Psi[x_i/x],\, A\rangle,\, ver,\, fal)$
$\Psi \wedge \chi$	N/A	$\begin{cases} SSG(\langle \Psi,\, A\rangle,\, ver,\, fal) & ,SSG(\langle \Psi,\, A\rangle,\, fal,\, fal) = 0 \\ SSG(\langle \chi,\, A\rangle,\, ver,\, fal) & ,SSG(\langle \Psi,\, A\rangle,\, fal,\, fal) = 1 \\ 1 & ,otherwise \end{cases}$
$\exists x : \Psi(x)$	ver provides $\bar{x}$	$1/\lvert\bar{x}\rvert \cdot \sum_{x_i \in \bar{x}} SSG(\langle \Psi[x_i/x],\, A\rangle,\, ver,\, fal)$
$\Psi \vee \chi$	N/A	$\begin{cases} SSG(\langle \Psi,\, A\rangle,\, ver,\, fal) & ,SSG(\langle \Psi,\, A\rangle,\, ver,\, ver) = 1 \\ SSG(\langle \chi,\, A\rangle,\, ver,\, fal) & ,SSG(\langle \Psi,\, A\rangle,\, ver,\, ver) = 0 \\ 0 & ,otherwise \end{cases}$
$\neg\Psi$	N/A	$1 - SSG(\langle \Psi,\, A\rangle,\, fal,\, ver)$
$p(\bar{x}) \neq \Psi$	N/A	$\begin{cases} 1 & ,A \models p(\bar{x}) \\ 0 & ,A \not\models p(\bar{x}) \end{cases}$
$p(\bar{x}) = \Psi$	N/A	$SSG(\langle \Psi,\, A\rangle,\, ver,\, fal)$

Table 3.1: Rules for $SSG(\langle \Phi,\, A\rangle,\, ver,\, fal)$

ing two constraints.

The first constraint is to ensure that a solution to the administrator's computational problem is a part of strategies submitted by CPSL members. Typically, this constraint is satisfied by straight forward formulations of search and optimization problems. However, straight forward formulations of decision problems often violate this constraint. Formulating claims to satisfy this constraint appears in our upcoming discussion of claim formulation for decision, search and optimization problems.

The second constraint is to find an *appropriate* trade-off between the complexity of the claim's logical formula and the claim's interpreting structure. In certain situations, the administrator may be able to reduce the overhead of claim specification by simplifying the claim's structure and complicating the claim's formula. This situation appears in our upcoming discussion of promise problems. In other situations, the administrator may be able to reduce the overhead of claim specification by simplifying the claim's formula and reusing existing complex computer simulation models or interpreters. This situation appears in our upcoming discussion of complex computational problems and complexity requirements. In other situations, dropping a quantified variable, with a relatively easy algorithm to find, from the claim's formula at the expense of a slight increase in the complexity of the structure might also be the appropriate trade-off to focus members on solving interesting problems.

The first kind of computational problems we address is decision problems. A decision problem can be specified by an arbitrary claim. For example, consider Bertrand's postulate that there is always at least one prime between n and $2 \cdot n$ which can be specified as $\forall n \in \{3,4,5,\ldots\}.\ \exists k \in \{n, n+1, \ldots, 2 \cdot n\}.\ prime(k)$. A strategy for playing an SSG of Bertrand's postulate consist of two functions: one to provide values for the universally quantified variable n and the other to provide values for the existentially quantified variable k given a value for n.

Given a strategy s, a solution to Bertrand's postulate can be obtained through playing an SSG with s playing both the roles of the verifier and the falsifier. In case s always wins as the verifier, s's solution is that Bertrand's postulate holds. In case s always wins as the falsifier, s's solution is that Bertrand's postulate does not hold.

It is also possible to use an alternative formulation where a solution to Bertrand's postulate is directly obtainable from a strategy for playing SSGs of Bertrand's postulate. By reformulating Bertrand's postulate as: $\exists d \in \{true, false\}.\ d \Leftrightarrow \forall n \in \{3,4,5,\ldots\}.\ \exists k \in \{n, n+1, \ldots, 2 \cdot n\}.\ prime(k)$, strategies for playing an SSG on the reformulated Bertrand's postulate will contain a function to provide a value for the existentially quantified boolean variable d. This function can be considered a solution to Bertrand's postulate because d is logically equivalent to Bertrand's postulate.

The second kind of computational problems we address is search or function problems. A search or function problem can be specified by a claim of the form

$\forall i\ :\exists o : \Phi(i,o)$ where Φ is a logical formula that holds when o is the correct output for the input i. A strategy for playing SSGs of such claims would contain a function to provide values for o given a value for i. This function can be considered as a solution to the underlying search or function problem. For example, consider the search problem of finding a topological ordering of a DAG. This problem can be specified as $\forall g \in DAG.\ \exists s \in sequences(nodes(g)).\ correct(g,s)$. A strategy for playing an SSG on this claim must contain a function that provides values for s given a value for g. This function can be considered as a solution to the problem of finding a topological ordering of a DAG.

The third kind of computational problems we address is optimization problems. An optimization problem can be specified by a claim of the form $\forall i.\ \exists o_1.\ \forall o_2.\ \Psi(i,o_1,o_2)$ where $\Psi(i,o_1,o_2)$ is a logical formula that holds when o_1 is the correct output for i and o_1 is better than o_2 or when o_2 is not a correct output for i. The function for providing o_1 in a strategy for playing SSGs of such claims can be considered as a solution to the underlying optimization problem. The MAX-SAT problem is a sample optimization problem that can be specified as $\forall f \in CNF.\ \exists j_1 \in assignments(vars(f)).\ \forall j_2 \in assignments(vars(f)).\ fsat(j_1,f) \geq fsat(j_2,f)$.

Alternatively, when only approximate solutions are sought, an optimization problem can be specified by a claim of the form $\forall i.\ \forall \delta.\ \exists o_1.\ \forall o_2.\ \Psi(i,\delta,o_1,o_2)$ where $\Psi(i,\delta,o_1,o_2)$ is a logical formula that holds when o_1 is the correct output for i and o_2 is at most δ better than o_1 or when o_2 is not a correct output for i. Again, the function for providing o_1 in a strategy for playing SSGs of such claims can be considered as a solution to the underlying optimization problem.

The forth kind of computational problems we address is promise problems. A promise problem is formed by adding an extra constraint, or promise, to the input domain of an existing computational problem. Promises tend to be semantic constraints that are computationally intensive to verify. Examples of promise problems include finding a satisfying assignment for satisfiable CNF formulas which can be formulated as: $\forall f \in SatisfiableCNFs.\ \exists g \in assignments(f).\ satisfies(g, f)$ interpreted in a structure that specifies the set $SatisfiableCNFs$, the function $assignments$ and the relation $satisfies$. The promise here being the extra semantic check that f is indeed a satisfiable formula which has to be performed by the administrator and can be a burden. Alternatively, this claim can be reformulated such that the burden of showing the satisfiability of the CNF formula f provided by the falsifier falls on the falsifier itself rather than on the administrator. The reformulated claim is: $\forall f \in CNFs.\ \neg(\exists h \in assignments(f).\ satisfies(h, f)) \vee (\exists g \in assignments(f).\ satisfies(g, f))$. In this reformulation, the verifier would be required to choose either the subclaim on the left or on the right hand side of the $\vee$. The verifier can choose the right hand side of the $\vee$ if it decides the CNF formula f to be have a satisfying assignment. In this case, the verifier will be required to demonstrate the satisfiability of f by providing a satisfying assignment g. The verifier can also choose the left hand side of the $\vee$ if it decides the CNF formula f to be unsatisfiable. In this case, the falsifier would be required to demonstrate the satisfiability of f by providing a satisfying assignment h. In general, promises can be added to universally quantified variables using logical implications but added to existentially quantified variables using logical conjunctions.

For complex computational problems, the overhead of claim specification is typically reduced by putting most of the complexity into the claim's structure rather than the formula. The rationale is that the structure can be specified in a general purpose programming language and may use existing software. We now give two examples of claims specifying complex computational problems.

The first example is a claim about the complexity of a finding a topological ordering of a graph. This claim is formulated as $\exists algo \in TopOrdAlgos.\ \exists v_0 \in \mathbb{N}.\ \exists e_0 \in \mathbb{N}.\ \exists c \in \mathbb{R}^+.\ \forall g \in Graphs.\ v_0 > vertices(g) \vee e_0 > edges(g) \vee correct(runFor$

$(algo, c \cdot (v0 + e0)), g)$ interpreted in a structure that defines the sets $TopOrdAlgos$, $\mathbb{N}$, $\mathbb{R}^+$ and $Graphs$ and the functions $runFor$, $vertices$, $edges$, $+$ and $\cdot$ and the relations $>$ and $correct$. The implementation of the function $runFor$ can use an existing interpreter to run the submitted algorithms for a specific number of steps.

The second example is a claim about the folding[1] of a the $\boldsymbol{HSP60}$[2] protein, according to a specific computer simulation model of the corresponding natural phenomena. The claim is formulated as $\exists f \in \boldsymbol{HSP60}Foldings.\ \forall f_2 \in \boldsymbol{HSP60}Foldings.\ energy(\boldsymbol{HSP60}, f) \leq energy(\boldsymbol{HSP60}, f)$ interpreted in a structure that defines the constant $\boldsymbol{HSP60}$, the set $\boldsymbol{HSP60}Foldings$, the function $energy$ and the predicate $\leq$. The implementation of the function $energy$ can use an existing computer simulation model to compute the energy of a particular protein folding.

Finally, the administrator may choose to put more complexity into the structure in order to focus the participants on specific problems. For example, the aforementioned formulation of Bertrand's postulate has primality testing included the structure. Alternatively, Bertrand's postulate may be formulated as $\forall n \in \{3, 4, 5, \ldots\}.\ \exists k \in \{n, n+1, \ldots, 2 \cdot n\}.\ \forall j \in 2, \ldots, k-1.\ remainder(k, j) > 0$. In this formulation, members are required to provide a factorization algorithm as a part of their strategies.

[1] The folding of a protein is a 3-D structure of the protein. Proteins comprise long chains of amino acids. Certain amino acids attract, others repulse. Certain amino acids are hydrophilic and would rather be on the outside closer to water molecules, others are hydrophobic and would rather be inside away from water molecules. These forces determine the native state of the protein which is the most stable folding of the protein.

[2] $\boldsymbol{HSP60}$ is one of the Heat Shock Proteins that are responsible for maintaining the integrity of cellular proteins in response to high levels of heat

3.4 Expressing Claims and Strategies

As mentioned above, a CPSL lab is centered around a claim that is specified by the administrator. It is our goal to make the tasks of claim specification and strategy development as convenient and as accessible as possible. In this section we describe a number of conventions for expressing claims and strategies in an Object Oriented Programming Language (OOPL) as conveniently as possible. In Section 3.5, we demonstrate these conventions in action through a number of examples.

A claim is specified by a class in an OOPL. The claim's formula can be modularly specified in a class-level or static array of strings named `FORMULAS`. Each entry in the `FORMULAS` array declares a formula using the formula declaration language formally specified in Figure 3.3. Formula names are unique within every `FORMULAS` array. The first formula declaration in a `FORMULAS` array is a no parameter formula declaration that declares the claim's logical formula and may refer to other formulas. Modular specification of the claim's logical formula is intended only as a syntactic convenience; recursion is disallowed. Non-logical symbols in the formula declarations (i.e. constants, functions, sets and relations) can only be valid references according to the host language scoping rules. Constants are references to an existing class-level or static field. Functions are references to existing class-level or static methods. Relations are references to existing boolean class-level or static methods. Sets are either references to existing classes, abstract classes or interfaces. Set membership tests are handled by the host language runtime at object construction time.

Figure 3.3 shows the grammar, in EBNF [1] notation, for our formula declaration language. A formula declaration consists of a name followed by a parameter list on the left hand side and a formula on the right hand side. The symbol $:=$ is used to separate the sides. A formula is inductively constructed from a primitive predicate or claim reference via conjunction, disjunction, negation, quantification and parenthesization operations. Arguments to predicate or claim references may be variable names, constants or function references. All operations associate to the right. Quantification has the least precedence followed by disjunction then conjunction then negation and parenthesization. Quantified variables and parameters have their types declared. Quantified variable names cannot shadow the names of other variable names in the same formula. This is a departure from standard predicate logic syntax that enables quantified variables to be identified by their name. This simplifies strategy development. References are interpreted according to the scoping rules of the host language. As an example of a formula declaration, consider HSR():= forall Integer q : forall Integer k : exists Integer n : HSRnqk(n, k, q) and ! exists Integer m : greater (m, n) and HSRnqk(m, q, k).

In a CPSL centered around a claim specified by the class `qualifier.SomeClaim`, a strategy for playing SSGs of `qualifier.SomeClaim` is specified by a class named `qualifier.SomeClaimStrategy`. `qualifier.SomeClassStrategy` must contain a class-level or static method that provides a collection of values for each quantified variable that is reachable through references from the logical formula of `qualifier.SomeClaim`. Methods are matched to the variables they provide values for by name. The method name matches the corresponding variable name qualified with its enclosing formula's name and the formula's enclosing class fully qualified name. Depending on the host language, separators may need to be replaced. For example, a method for providing values for a variable named `x` defined in a formula named `Formula1` declared in `qualifier.SomeClaim` may be called `qualifier_SomeClaim_Formula1_x`. The parameters of the method providing values for some variable `var` are all variables that are in scope at the declaration site of `var`.

Now, we demonstrate these conventions in action through a number of exam-

```
Declaration = Identifier , ParameterList , ’:=’ , Formula ;

Formula = AndCompound , [ ’or’ , Formula ] ;
AndCompound = Simple , [ ’and’ , AndCompound ] ;

Simple = Quantified | Negated | Reference | Parenthesized ;

Quantified = Quantifier , VarDecl , ’:’ , Formula ;
Negated = ’!’ , Simple ;
Parenthesized = ’(’ , Formula , ’)’ ;
Reference = PredicateOrClaimReference , ArgumentList ;

Quantifier = ’exists’ | ’forall’ ;
VarDecl = VarType , VarName ;
VarName = Identifier ;
VarType = QualifiedIdentifier ;
PredicateOrClaimReference = QualifiedIdentifier ;
QualifiedIdentifier = Identifier , {’.’ , Identifier} ;

ArgumentList = ’(’ , [ Term , { ’,’ , Term } ] , ’)’  ;
Term = VarName | ConstantName | Reference ;
ConstantName = QualifiedIdentifier ;

ParameterList = ’(’ , [ VarDecl , { ’,’ , VarDecl } ] , ’)’  ;

Identifier = (Letter | Symbol) , { Letter | Digit | Symbol } ;

Letter = ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’ | ’G’
       | ’H’ | ’I’ | ’J’ | ’K’ | ’L’ | ’M’ | ’N’
       | ’O’ | ’P’ | ’Q’ | ’R’ | ’S’ | ’T’ | ’U’
       | ’V’ | ’W’ | ’X’ | ’Y’ | ’Z’
       | ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’ | ’g’
       | ’h’ | ’i’ | ’j’ | ’k’ | ’l’ | ’m’ | ’n’
       | ’o’ | ’p’ | ’q’ | ’r’ | ’s’ | ’t’ | ’u’
       | ’v’ | ’w’ | ’x’ | ’y’ | ’z’ ;
Digit = ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’
    ;
Symbol = ’$’ | ’_’;
```

Figure 3.3: EBNF [1] Grammar of Formula Declaration Language

ples using Java [13] as a host language.

3.5 Sample Computational Problem Solving Labs

3.5.1 The Highest Safe Rung Lab

The Highest Safe Rung (HSR) problem is to find the largest number of stress levels that a stress testing plan can examine using `q` tests and `k` copies of the product under test. To further illustrate the problem, we take jars as a representative product and the rungs of a ladder as a representative of stress levels. A stress testing plan is a decision tree for determining the highest level of stress a product can endure. This corresponds to the highest rung from which a jar can be thrown without breaking. If all we have is a single jar, then we cannot risk breaking it without figuring out the highest safe rung. Therefore, we have to linearly search the rungs from the lowest to the highest until the jar breaks. If we have $k > 1$ jars and $q = k$ tests, we can afford to break all the jars in a binary search that is capable of examining a maximum of 2^k stress levels. But, if we have more tests (i.e. $q > k$) what would the maximum number of stress levels that a stress testing plan can examine?

Figure 3.4 shows a claim specification for the HSR problem. The claim's logical formula is named `HSR` and uses the subformula `HSRnqk` twice. The claim's logical formula contains two sets: `Integer` which refers to a standard Java [13] library class and `SearchPlan` which refers to an interface defined in the scope of `HSRClaim`. The claim's logical formula also contains two relations: `greater` and `correct`. Both relations refer to a static method defined in the scope of `HSRClaim`.

Figure 3.5 shows the skeleton of a strategy for playing SSGs of the claim `HSR`. The skeleton defines a method for all the quantified variables reachable through reference from the claim `HSR`.

```
class HSRClaim {
  public static final String[] FORMULAS = new String[]{
    "HSR() := forall Integer q : forall Integer k : exists
        Integer n : HSRnqk(n, k, q) and ! exists Integer m :
        greater(m, n) and HSRnqk(m, q, k)",
    "HSRnqk(Integer n, Integer q, Integer k) := exists SearchPlan
         sp : correct(sp, n, q, k)"
  };

  public static boolean greater(Integer n, Integer m){
    return n > m;
  }

  public static interface SearchPlan{}
  public static class ConclusionNode implements SearchPlan{
    Integer hsr;
  }
  public static class TestNode implements SearchPlan{
    Integer testRung;
    SearchPlan yes; // What to do when the jar breaks.
    SearchPlan no; // What to do when the jar does not break.
  }

  public static boolean correct(SearchPlan sp, Integer n, Integer
       q, Integer k){
    // sp satisfies the binary search tree property, has n leaves
        , of depth at most q, all root-to-leaf paths have at most
        k "yes" branches.
    ...
  }
}
```

Figure 3.4: Claim Specification for The Highest Safe Rung Lab

3.5.2 Gale-Shapley's Worst Case Input Lab

The focal problem of the Gale-Shapley's worst case input lab is finding the input making the outermost loop of Gale-Shapley's stable matching algorithm [10] iterate the most. Figure 3.6 shows the claim specification of the Gale-Shapley's worst case input lab.

```
class HSRStrategy {
  public static Iterable<Integer> HSR_q(){
    // provide an integer q such that the formula ”forall Integer
         k : exists Integer n : HSRnqk(n, k, q) and ! exists
        Integer m : greater(m, n) and HSRnqk(m, q, k)” does not
        hold
    ...
  }
  public static Iterable<Integer> HSR_k(Integer q){
    // provide an integer k such that the formula ”exists Integer
         n : HSRnqk(n, k, q) and ! exists Integer m : greater(m, n
        ) and HSRnqk(m, q, k)” does not hold.
    ...
  }
  public static Iterable<Integer> HSR_n(Integer q, Integer k){
    // provide an integer n such that the formula ”HSRnqk(n, k, q
        ) and ! exists Integer m : greater(m, n) and HSRnqk(m, q,
        k)” holds.
    ...
  }
  public static Iterable<Integer> HSR_m(Integer q, Integer k,
      Integer n){
    // provide an integer m such that the formula ”exists Integer
         m : greater(m, n) and HSRnqk(m, q, k)” holds.
    ...
  }
  public static Iterable<SearchPlan> HSRnqk_sp(Integer n, Integer
       q, Integer k){
    //provide a SearchPlan sp such that the formula ”correct(sp,
        n, q, k)” holds
    ...
  }
}
```

Figure 3.5: HSR Strategy Skeleton

3.5.3 Minimum Graph Basis Lab

The focal problem of the minimum graph basis lab is finding the smallest basis for a given directed graph. The basis of a directed graph is a subset of the graph nodes such that every node in the graph is reachable from some node in the basis. Figure 3.7 shows the claim specification of the minimum graph basis lab.

```
class WorstCaseGaleShapleyClaimFamily {
  public static final String[] FORMULAS = new String[]{
    ”WorstCaseInput() := forall Integer n : exists Preferences
        pref1 : hasSize(pref1, n) and ! exists Preferences pref2 :
         hasSize(pref2, n) and moreExpensive(pref2, pref1)”
  };

  public static boolean moreExpensive(Preferences pref1,
      Preferences pref2){
    // Does pref1 make the outermost loop of Gale−Shapley's
        stable matching algorithm iterate more than pref2 does?
    ...
  }
  public static boolean hasSize(Preferences pref, Integer n){
    // Does pref have the preferences of exactly n parties?
    ...
  }
  public static class Preferences{... }
}
```

Figure 3.6: Claim Family Specification of The Gale-Shapley's Worst Case Input Lab

```
class MinGraphBasisClaimFamily {
  public static final String[] FORMULAS = new String[]{
    ”MinBasis() := forall Graph g : exists NodeSet ns1 :
        allReachable(ns1, g) and ! exists NodeSet ns2 :
        allReachable(ns2, g) and smaller(ns2, ns1)”
  };

  public static boolean allReachable(NodeSet ns, Graph g){
    // Is every node in g reachable from some node in ns?
    ...
  }
  public static boolean smaller(NodeSet ns2, NodeSet ns1){
    // Does ns2 have fewer nodes than ns1?
    ...
  }
  public static class NodeSet{ ... }
  public static class Graph{ ... }
}
```

Figure 3.7: Claim Family Specification of The Minimum Graph Basis Lab

3.6 Related Work

3.6.1 Crowdsourcing and Human Computation

There are several existing systems and approaches that can be used to organize human communities to perform complex tasks including the task of solving computational problems. Algorithm development competitions held on platforms such as TopCoder [30], Project Euler [2] and Jutge [23] are similar our CPSLs in that they are based on organizing algorithm development competitions. In all three platforms, algorithms submitted by community members are evaluated using a benchmark prepared by the competition or the system administrator. Moreover, formal specification is not mandatory in any of these systems.

FoldIt [7] and EteRNA [5] are two crowdsourcing systems aiming to leverage human intelligence to solve two particular optimization problems at the instance level. FoldIt leverages human intelligence to find the native structure of specific proteins according to a particular computer simulation model [3] of the natural phenomena of protein folding. EteRNA, on the other hand, leverage human intelligence to design RNA molecules with a certain folding as their native state. The two systems are quite similar to each other and we limit our discussion here to FoldIt. FoldIt is similar to CPSLs in that it organizes a community to solve a computational problem. FoldIt is also similar to CPSLs in that the overhead on the administrator is minimal and that FoldIt has a peer-evaluation nature in the sense that the system only accepts foldings that are better than any other folding that has been previously submitted by other human participants. FoldIt has a competitive nature that comes from different leader-boards maintained by the system. However, it is controversial whether competition is the main motivation for human participants to contribute solutions to open protein folding problems. The main difference between FoldIt and CPSLs is that FoldIt organizes a community to solve a *fixed* computational problem at the instance level only. CPSLs can be used to organize a community to solve computational problems in general.

Wikipedia has also been quite successful in organizing communities to perform complex tasks. Today, there are numerous Wikipedia pages that informally describe computational problems as well as algorithms to solve them. Wikipedia significantly differs from CPSLs in that community members *collaboratively* edit Wikipedia pages. When conflicts arise, they are *subjectively* resolved through negotiation and arbitration. In CPSLs, each member has their own version of the solution. These versions are *always* in a conflict that is solved through an *objective* competition. Moreover, in Wikipedia, formal specification of problems is not mandatory. Furthermore, algorithms are typically specified at a high-level which reduces the chances for conflicts yet making algorithms on the Wikipedia less directly reusable and testable.

Crowdsourcing has become an important problem solving approach that enables us to tackle large scale problems that require human intelligence to solve. There are two main reasons that human intelligence is required to solve a problem. First, the problem is *underspecified* such as image labeling [31], the construction of web page classifiers [16], and the creation of Wikipedia pages. Humans are needed to partially specify *what* the problem is. Second, The problem is formally specified but complex enough that we have either no known solution procedure or a rather inefficient one. Examples include, programming and discovering protein folding [7], [5]. Humans are needed either to *solve* the problem or to decide *how* to solve the problem. CPSLs has the potential of being used as crowdsourcing systems for formally specified computational problems because they provide attractive solutions to most of the key challenges of crowdsourcing systems and have other features desirable in crowdsourcing systems.

CPSLs provide attractive solutions to most of the following four key challenges that crowdsourcing systems need to address [8]:

1. What contributions can users make? In a CPSL, community members are required to provide a strategy for playing an SSG of the claim formally specifying the underlying computational problem of the CPSL.

2. How to evaluate users and their contributions? CPSLs evaluate strategies submitted by community members using a collusion-resistant tournament of SSGs where members are guaranteed to always have a chance to rank at the top and to always have a chance to expose problems with the solutions of their opponents.

3. How to combine user contributions to solve the target problem? As described earlier, CPSLs do not combine user contributions. Instead, community members can receive targeted feedback from the SSGs that their strategies lost. Members can incorporate this feedback into their future strategies.

4. How to recruit and retain users? Competition outcome in a CPSL can serve as a basis for a host of user recruitment schemes such as offering a monetary prize to the competition winners.

As we mentioned earlier, CPSLs have features that are desirable for crowdsourcing systems. In [18], Kittur et al. argue that an ideal crowd work system would offer peer-to-peer and expert feedback and encourage self-assessment. Such a system would help workers to learn, and produce better results. In CPSLs, community members can get, through competition, targeted feedback about the weaknesses of their strategies.

3.6.2 Origin

We started this line of work with the Specker Challenge Game (SCG) [22]. The goal was to create an educational game in which students can learn from each other with a minimal interaction with the teaching staff. The rules where informally described by ad-hoc rules that were called refutation protocols [3].

[3] In reference to the seminal work [25] of the famous philosopher of science, Karl Popper.

Chapter 4

Collusion-Resistant Semantic Game Tournaments

In this chapter we develop an SG tournament that is arguably objective, anonymous, neutral, monotonic and thorough. As we mentioned earlier, collusion potential is a key challenge to ensuring the anonymity and success of a tournament. A set of colluding participants may arrange to lose, on purpose, against a specific participant in order to inflate that participant's rank and effectively worsen the rank of the opponents of that participant.

In Section 4.1 we give an example demonstrating how a set of colluding participants can worsen the rank of other participants behind their back and how certain ranking functions can fend against collusion. Then, in Section 4.2, we formalize the notions of beating functions representing SG tournament results as well as ranking functions. Then, in Section 4.3, we provide a formal, in depth study of collusion-resistant ranking functions that begins with a formal characterization of the limited collusion effect property of ranking functions. Informally, the limited collusion effect property means that no participant can have their rank worsened behind their back due to collusion. We then give a *representation theorem* of ranking functions possessing the limited collusion effect property as well as other basic monotonicity properties. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting. Then, in Section 4.4 we present an SG tournament and argue for its objectivity, anonymity, neutrality, monotonicity and thoroughness. Finally, in Section 4.5 we discuss the related work.

4.1 Collusion-Resistant Ranking Functions at a Glance

Suppose that we have an underlying *true* claim C specifying a computational problem. Therefore, there must be a winning strategy for a verifier in any SG of C. Suppose that p_1 and p_2 are two perfectly acting participants. Therefore, both will choose to take the verifier side in an SG of C. Also, both p_1 and p_2 will apply the winning strategy when taking the verifier side in an SG of C. Table 4.1 shows the outcome of a double round robin semantic game tournament between p_1 and p_2. p_1 and p_2 play two SGs on C. In the first game, shown in the top right cell, p_2 takes the verifier side and p_1 is forced to take the falsifier side. In the second game, shown in the bottom left cell, p_1 takes the verifier side and p_2 is forced to take the falsifier side. By virtue of being perfectly acting participants, p_2 wins the first game and p_1 wins the second game.

We demonstrate our notion of collusion-resistant ranking functions using the following four score-based ranking functions:

- Number of wins (#W): For each participant, we count the number of wins for every participant throughout the tournament. The higher the participant

Ver Fal	p_1	p_2
p_1	-	p_2
p_2	p_1	-

Table 4.1: Outcome of a Semantic Game Tournament with Two Perfectly Acting Participants

scores, the better the participant's rank is.

- Number of losses (#L): For each participant, we count the number of losses for every participant throughout the tournament. The lower the participant scores, the better the participant's rank is.

- Number of wins against a non forced participant (#WNF): we only count the number of games a participant has won against a non forced participant. A participant is said to be forced in an SG if it takes the opposite side to the side the participant chooses to take. The higher the participant score, the better the participant's rank is.

- Number of faults (#NFL): we only count the number of games a participant loses while taking its chosen side. The lower the participant scores, the better the participant's rank is.

Both #W and #L ignore the participants' side choice altogether. Also, Both #WNF and #NFL ignore games in which the loser is forced as a form of compensation for players at a disadvantage. The rationale is that in such games it could be that the loser had no chance of winning whatsoever.

Table 4.2 demonstrates the scores of p_1 and p_2 according to the four ranking functions. Each participant wins a single game. Therefore, both score a single point using the #W function. Each participant loses a single game. Therefore, both score a single point using the #L function. Each participant wins only against a

Participant	#W	#L	#WNF	#NFL
p_1	1	1	0	0
p_2	1	1	0	0

Table 4.2: Evaluating a Semantic Game Tournament with Two Perfectly Acting Participants

Ver Fal	p_1	p_2	p_3
p_1	-	p_2	p_3
p_2	p_1	-	p_2
p_3	p_1	p_2	-

Table 4.3: Outcome of a Semantic Game Tournament with Two Colluding Participants

forced player. Therefore, both score zero points using the #WNF function. Each participant loses only while forced. Therefore, both score zero points using the #NFL function. Using either of the four ranking functions, both p_1 and p_2 are top ranked.

Now, suppose that a third player p_3 has joined the tournament not for the purpose of competing with p_1 and p_2 for the top rank, but to cut p_1 short from being top ranked. We assume p_3 has access to the winning strategy of p_2 and will use it except against p_2. This situation is illustrated in Table 4.3. The highlighted cell marks the semantic game that p_3 loses on purpose for the benefit of p_2.

Now, we examine the four ranking functions to determine which ones are resistant to the collusion between p_3 and p_2. Table 4.4 shows the scores of p_1, p_2 and p_3 using the four ranking functions. For each ranking function, the cells corresponding to the best scores are highlighted. Among the four ranking functions we examined, we note that fault counting is the only collusion-resistant function.

In the following two sections, we provides a formal, in depth study of collusion-resistant ranking functions.

Participant	#W	#L	#WNF	#NFL
p_1	2	2	0	0
p_2	3	1	1	0
p_3	1	3	0	1

Table 4.4: Evaluating a Semantic Game Tournament with Two Colluding Participants

4.2 Formalizing Beating and Ranking Functions

In this section we formalize the notions of beating and ranking functions. We use a beating function to represent a tournament result and use a ranking function to produce an ordering of tournament participants based on a tournament result. We also describe the algebraic structure of beating functions. In subsequent sections, we rely on the operations in this structure to formulate properties of beating and ranking functions and use the properties of the structure of beating functions in our proofs.

4.2.1 Notation

We modeled our notation for variables after the Hungarian notation used to name variables in computer programs. The goal is to avoid as many quantifiers as possible when expressing logical formulas. For example, instead of writing $\forall p \in P.\ \Phi(p)$, we directly write $\Phi(p)$. Also, instead of writing $\forall a,b \in P.\ \Phi(a,b)$ we directly write $\Phi(p_a, p_b)$.

Explicitly, we use a single capital Latin letter to denote a set and use the same small Latin letter to denote an element of the set. Subscripts are used to distinguish multiple elements of the same set, when necessary. Constants are denoted using boldface font. Functions are denoted using small Latin letters and superscripts are used to denote their type parameters. Free variables are assumed to be universally quantified. Two distinctly named free variables are assumed to only take distinct values. We use $\preceq$ to denote an arbitrary ranking function. We also use subscripts to distinguish between ranking functions.

In our proofs, we put labels to the right of formulas. We use the notation $LABEL[term_1/var_1, \dots term_n/var_n]$ to denote a particular instantiation of the formula labeled $LABEL$ in which the freely occurring variables $var_1, \dots var_n$ are replaced with the terms $term_1, \dots term_n$ respectively.

4.2.2 Beating Functions

Let $\mathbf{s_v}$ and $\mathbf{s_f}$ be two constants denoting the verifier and falsifier sides respectively. Let $\mathbf{S} = \{\mathbf{s_v}, \mathbf{s_f}\}$. We use a **beating** function $b^p : P \times P \times \mathbf{S} \times \mathbf{S} \times \mathbf{S} \rightarrow \mathbb{Q}^+$ to represent the results of all semantic games comprising a tournament among a finite set of players P. $b^p(p_w, p_l, s_{wc}, s_{lc}, s_w)$ denotes the number, or fraction, of semantic games won by p_w against p_l where p_w chooses to take the side s_{wc} and p_l chooses to take the side s_{lc} and s_w is the actual side taken by the p_w. We use B^P to denote the set of all possible beating functions for a given finite set P of players.

4.2.3 Ranking Functions

We define a ranking to be a total preorder (i.e. a reflexive, transitive and total binary relation). We use R^P to denote the set of all possible rankings of a given set P of players. A **ranking function** $\preceq : B^P \rightarrow R^P$ associates some ranking to every beating function. We say that p_x is weakly better than p_y (i.e. p_x at least as good as p_y) according to the ordering assigned by the ranking function $\preceq$ to the beating relation b^p if $p_x \preceq(b^p)\, p_y$. We say that p_x is strictly better than p_y if $p_y \npreceq(b^p)\, p_x$. Formally, a ranking function satisfies the following axioms:

$$p \preceq(b^p)\, p \tag{REFL}$$

$$p_x \preceq(b^p)\, p_y \wedge p_y \preceq(b^p)\, p_z \Rightarrow p_x \preceq(b^p)\, p_z \tag{TRAN}$$

$$p_x \preceq(b^p)\, p_y \vee p_y \preceq(b^p)\, p_x \tag{TOTAL}$$

4.2.4 The Algebraic Structure of Beating Functions

The set B^P and pointwise rational addition operation $(b^p_x + b^p_y)(p_w, p_l, s_{wc}, s_{lc}, s_w) = b^p_x(p_w, p_l, s_{wc}, s_{lc}, s_w) + b^p_y(p_w, p_l, s_{wc}, s_{lc}, s_w)$ form an algebraic structure. The pointwise rational addition operation is associative, commutative and $\mathbf{b^p_0}$ is its identity element. $\mathbf{b^p_0}$ is the beating function representing the results of the empty set of semantic games. Therefore, $\mathbf{b^p_0}(p_w, p_l, s_{wc}, s_{lc}, s_w) = 0$.

We add the following four restriction operations to the structure of beating functions:

- **Win restriction**: we use $b^p|^w_{p_x}$ to denote a restricted version of b^p that only contains those games that p_x wins. Formally,

$$b^p|^w_{p_x}(p_w, p_l, s_{wc}, s_{lc}, s_w) = \begin{cases} b^p(p_w, p_l, s_{wc}, s_{lc}, s_w) & , p_w = p_x \\ 0 & , otherwise \end{cases} \tag{DEF.1}$$

- **Loss restriction**: we use $b^p|^l_{p_x}$ to denote a restricted version of b^p that only contains those games that p_x loses. Formally,

$$b^p|^l_{p_x}(p_w, p_l, s_{wc}, s_{lc}, s_w) = \begin{cases} b^p(p_w, p_l, s_{wc}, s_{lc}, s_w) & , p_l = p_x \\ 0 & , otherwise \end{cases} \tag{DEF.2}$$

- **Fault restriction**: we use $b^p|^{fl}_{p_x}$ to denote a restricted version of b^p that only contains the games in which p_x makes a fault. These are the games that p_x

loses while not forced.Formally,

$$b^p|_{p_x}^{fl}(p_w,p_l,s_{wc},s_{lc},s_w) = \begin{cases} b^p(p_w,p_l,s_{wc},s_{lc},s_w) & ,p_l = p_x \wedge s_{lc} \neq s_w \\ 0 & ,otherwise \end{cases} \tag{DEF.3}$$

- **Control restriction**: we use $b^p|_{p_x}^{c}$ to denote a restricted version of b^p that only contains those games that p_x controls. These are the games that p_x either wins or had a chance to win. Formally:

$$b^p|_{p_x}^{c} = b^p|_{p_x}^{w} + b^p|_{p_x}^{fl} \tag{DEF.4}$$

We also add a complement restriction operation for each of the aforementioned restriction operations. We use $b^p|_{p_x}^{!w}$ to denote a restricted version of b^p that only contains those games that p_x does not win. Formally:

$$b^p|_{p_x}^{w} + b^p|_{p_x}^{!w} = b^p \tag{DEF.5}$$

We use $b^p|_{p_x}^{!l}$ to denote a restricted version of b^p that only contains those games that p_x does not lose. Formally:

$$b^p|_{p_x}^{l} + b^p|_{p_x}^{!l} = b^p \tag{DEF.6}$$

We use $b^p|_{p_x}^{!fl}$ to denote a restricted version of b^p that only contains the games in which p_x does not make a fault. Formally:

$$b^p|_{p_x}^{fl} + b^p|_{p_x}^{!fl} = b^p \tag{DEF.7}$$

We use $b^p|_{p_x}^{!c}$ to denote a restricted version of b^p that only contains those games that p_x does not control. Formally:

$$b^p|_{p_x}^{c} + b^p|_{p_x}^{!c} = b^p \tag{DEF.8}$$

We now list some formal properties of the structure of beating functions that we use later in our proofs:

$$b^p|_{p_y}^{fl}|_{p_x}^{fl} = \mathbf{b_0^p} \tag{PROP.1}$$

$$b^p|_{p_y}^{l}|_{p_x}^{fl} = \mathbf{b_0^p} \tag{PROP.2}$$

$$b^p|_{p_y}^{w}|_{p_x}^{w} = \mathbf{b_0^p} \tag{PROP.3}$$

$$b^p|_{p_x}^{w}|_{p_x}^{fl} = \mathbf{b_0^p} \tag{PROP.4}$$

$$b^p|_{p_x}^{!fl} = b^p|_{p_x}^{w} + b^p|_{p_x}^{!c} \tag{PROP.5}$$

$$b^p|_{p_x}^{!c}|_{p_x}^{c} = \mathbf{b_0^p} \tag{PROP.6}$$

$$b^p|_{p_x}^{fl}|_{p_x}^{l} = b^p|_{p_x}^{fl} \tag{PROP.7}$$

4.3 Collusion-Resistant Ranking Functions

In this section, we formalize our notion of collusion-resistant ranking functions as ranking functions satisfying the limited collusion effect property. Then we formalize the two basic monotonicity properties of never penalizing wins nor rewarding losses. Finally, we provide a more practical alternative characterization of collusion-resistant ranking functions that never discourage winning nor encourage losing. Finally, we provide a *representation theorem* of ranking functions possessing the limited collusion effect property as well as the other two basic monotonicity properties. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting.

4.3.1 Limited Collusion Effect

A ranking function $\preceq$ is said to have the **Limited Collusion Effect (LCE)** property if for any two arbitrary players p_x and p_y the rank of p_y with respect to p_x cannot be improved by manipulating games that p_x can not control their outcome. These are the games that p_x is not involved in or the games p_x loses while forced. Formally, a ranking function satisfies the LCE property if it satisfies the following axioms:

$$b_2^p|_{p_x}^c = \mathbf{b_0^p} \wedge p_x \preceq(b_1^p)\, p_y \Rightarrow p_x \preceq(b_1^p + b_2^p)\, p_y \tag{LCE.I}$$

$$b_2^p|_{p_x}^c = \mathbf{b_0^p} \wedge p_y \not\preceq(b_1^p)\, p_x \Rightarrow p_y \not\preceq(b_1^p + b_2^p)\, p_x \tag{LCE.II}$$

The first axiom asserts that if p_x is ranked weakly better than p_y under the beating function b_1^p, then p_x remains weakly better than p_y when more games that p_x cannot control are added to b_1^p. The second axiom asserts that if p_x is ranked strictly better p_y under the beating function b_1^p, then p_x remains strictly better than p_y when more games that p_x cannot control are added to b_1^p.

4.3.2 Monotonicity

As we mentioned earlier, a monotonic ranking function must not reward losing nor penalize winning. In other words, a monotonic ranking function must have a **Non-Negative Regard for Winning (NNRW)** and a **Non-Positive Regard for Losing (NPRL)**. That is, a player's rank cannot be worsened by an extra winning nor can it be improved by an extra loss. Formally, a ranking function must satisfy the following axioms:

$$p_x \preceq(b_1^p)\, p_y \Rightarrow p_x \preceq(b_1^p + b_2^p|_{p_x}^w)\, p_y \tag{NNRW.I}$$

$$p_x \preceq(b_1^p + b_2^p|_{p_y}^w)\, p_y \Rightarrow p_x \preceq(b_1^p)\, p_y \tag{NNRW.II}$$

$$p_x \preceq(b_1^p)\, p_y \Rightarrow p_x \preceq(b_1^p + b_2^p|_{p_y}^l)\, p_y \tag{NPRL.I}$$

$$p_x \preceq(b_1^p + b_2^p|_{p_x}^l)\, p_y \Rightarrow p_x \preceq(b_1^p)\, p_y \tag{NPRL.II}$$

4.3.3 A Representation Theorem for Monotonic, Collusion-Resistant Ranking Functions

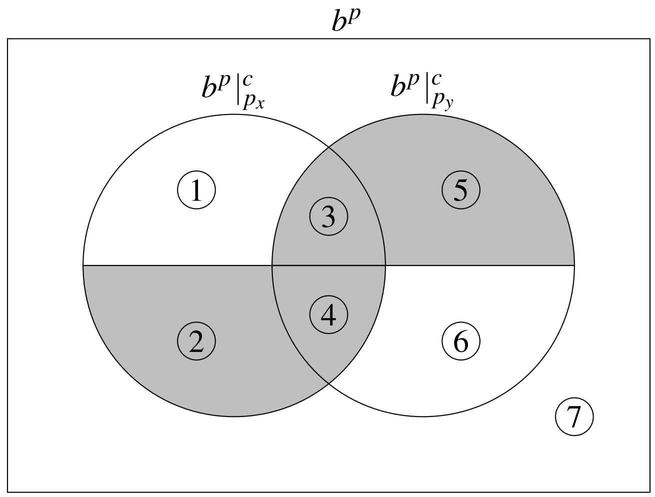
We now give a representation theorem for monotonic, collusion-resistant ranking functions. In essence, we show that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting. Being fault-based is a quite unusual design principle for ranking functions that ranking function designers are likely to violate. In sports tournaments, it is often the case for ranking functions to be based on wins or at least based on both wins and losses but not solely on losses [21]. Being fault-based is also a more practical characterization of monotonicity and possessing the limited collusion effect property as it is easier to design fault-based ranking functions than it is to design ranking functions possessing the limited collusion effect.

A ranking function $\preceq$ is said to be **Local Fault Based (LFB)** if for any two arbitrary players p_x and p_y the relative rank $\preceq$ assigns to p_x with respect to p_y solely depends on the games where p_x or p_y make a fault. Formally,

$$p_x \preceq\!\left(b^p|_{p_x}^{fl} + b^p|_{p_y}^{fl}\right) p_y \Leftrightarrow p_x \preceq\!\left(b^p\right) p_y \tag{LFB}$$

Theorem 4.3.1. *For any ranking function having NNRW and NPRL, LCE is equivalent to LFB. Formally, NNRW $\wedge$ NPRL $\Rightarrow$ (LCE $\Leftrightarrow$ LFB).*

Figure 4.1 presents a partitioning of the games represented by an arbitrary beating function b^p. This partitioning illustrates the intuition behind this theorem and its proof. The intuition is that a ranking function satisfying the LFB property $\preceq$ must completely decide the relative rank of any two arbitrary players p_x and p_y based on the games in the shaded partitions only. Games in the unshaded partitions cannot influence the relative rank of p_x and p_y assigned by $\preceq$.



① $b^p|^{!fl}_{p_y}|^{w}_{p_x} = b^p|^{!c}_{p_y}|^{w}_{p_x}$

② $b^p|^{!w}_{p_y}|^{fl}_{p_x} = b^p|^{!c}_{p_y}|^{fl}_{p_x}$

③ $b^p|^{fl}_{p_y}|^{w}_{p_x} = b^p|^{c}_{p_y}|^{w}_{p_x}$

④ $b^p|^{w}_{p_y}|^{fl}_{p_x} = b^p|^{c}_{p_y}|^{fl}_{p_x}$

⑤ $b^p|^{!w}_{p_x}|^{fl}_{p_y} = b^p|^{!c}_{p_x}|^{fl}_{p_y}$

⑥ $b^p|^{!fl}_{p_x}|^{w}_{p_y} = b^p|^{!c}_{p_x}|^{w}_{p_y}$

⑦ $b^p|^{!c}_{p_x}|^{!c}_{p_y}$

Figure 4.1: Beating Functions Representing Partitions of the Semantic Games Represented by b^p

We now give an informal proof of this theorem using the partitioning shown in Figure 4.1. We break our theorem into the following two lemmas. The first lemma is that NNRW and LCE imply LFB. The second lemma is that NPRL and LFB imply LCE. Our theorem follows directly from both lemmas.

To prove the first lemma, let $\preceq$ be a ranking function that violates the LFB property. By definition of the LFB property, there must be two players p_x and p_y such that the games in the unshaded region influence the relative rank assigned by $\preceq$ to p_x and p_y. The influence can either be positive (case I) or negative (case II) for p_x. Suppose that games in the unshaded region positively influence the rank assigned by $\preceq$ to p_x with respect to p_y. But, assuming that $\preceq$ satisfies the LCE property, games in partitions ①, ⑦ cannot improve p_x's rank with respect to p_y because it only contains games not under p_y's control. Also, assuming that $\preceq$ satisfies the NNRW property, games in partition ⑥ cannot improve p_x's rank with respect to p_y because it only contains games that p_y has won. Therefore, our assumption that $\preceq$ satisfies both LCE and NNRW cannot be true. We have shown the contrapositive of the first lemma for case I. We now consider case II. Suppose that games in the unshaded region negatively influence the rank assigned by $\preceq$ to p_x with respect to p_y. But assuming that $\preceq$ satisfies the LCE property, games in partitions ⑥, ⑦ cannot worsen p_x's rank with respect to p_y because it only contains games not under p_x's control. Also, assuming that $\preceq$ satisfies the NNRW property, games in partition ① cannot worsen p_x's rank with respect to p_y because it only contains games that p_x has won. Therefore, our assumption that $\preceq$ satisfies both LCE and NNRW cannot be true. We have shown the contrapositive of the first lemma for case II and the first part of the proof is now complete.

To prove the second lemma, let $\preceq$ be a ranking function satisfying both NPRL and LFB. By definition of the LFB property, only games in the shaded region influence the relative rank assigned by $\preceq$ to p_x and p_y. Games in the regions ②,③ and ④ are under the control of p_x. Only games in region ⑤ can influence the relative rank assigned by $\preceq$ to p_x and p_y, yet games in region ⑤ are not under the control of p_x. However, games in region ⑤ are all faults made by p_y and by NPRL they cannot improve the rank of p_y with respect to p_x. Therefore, only games under the control of p_x may worsen p_x's rank with respect to p_y. An identical argument applies to the rank of p_y. This completes the prove of the second lemma and hence the theorem.

Now we present our formal proof. We start with few lemmas. The first lemma formalizes the partitioning shown in Figure 4.1. Essentially, our first lemma asserts that if we add the games in the shaded region, represented by the beating function $b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y}$, to the games in partitions ①, ⑥ and ⑦ represented by the beating functions $b^p|^{!fl}_{p_y}|^{w}_{p_x}$, $b^p|^{!fl}_{p_x}|^{w}_{p_y}$ and $b^p|^{!c}_{p_x}|^{!c}_{p_y}$ respectively, we get b^p.

Lemma 4.3.2. $b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_x}|^{!c}_{p_y} = b^p$

Proof.

$$b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x}$$

By DEF.8$[b^p|^{fl}_{p_y}/b^p]$:

$=b^p|^{fl}_{p_x}+b^p|^{fl}_{p_y}|^{c}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_y}|^{w}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By DEF.4$[b^p|^{fl}_{p_y}/b^p]$:

$=b^p|^{fl}_{p_x}+b^p|^{fl}_{p_y}|^{w}_{p_x}+b^p|^{fl}_{p_y}|^{fl}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_y}|^{w}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By PROP.1 and identity of + :

$=b^p|^{fl}_{p_x}+b^p|^{fl}_{p_y}|^{w}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_y}|^{w}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By commutativity of + and distributivity of + on restrictions :

$=b^p|^{fl}_{p_x}+(b^p|^{fl}_{p_y}+b^p|^{!fl}_{p_y})|^{w}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By DEF.7 :

$=b^p|^{fl}_{p_x}+b^p|^{w}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By commutativity of + and DEF.4 :

$=b^p|^{c}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!fl}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By PROP.5 :

$=b^p|^{c}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+(b^p|^{w}_{p_x}+b^p|^{!c}_{p_x})|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By distributivity of + on restriction operations:

$=b^p|^{c}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{w}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By PROP.3 and identity of + :

$=b^p|^{c}_{p_x}+b^p|^{fl}_{p_y}|^{!c}_{p_x}+b^p|^{!c}_{p_x}|^{w}_{p_y}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By commutativity of + and restrictions and distributivity of + on restrictions :

$=b^p|^{c}_{p_x}+(b^p|^{w}_{p_y}+b^p|^{fl}_{p_y})|^{!c}_{p_x}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By DEF.4$[p_y/p_x]$:

$=b^p|^{c}_{p_x}+b^p|^{c}_{p_y}|^{!c}_{p_x}+b^p|^{!c}_{p_y}|^{!c}_{p_x}$

By distributivity of + on restrictions :

$=b^p|^{c}_{p_x} + (b^p|^{c}_{p_y} + b^p|^{!c}_{p_y})|^{!c}_{p_x}$

By DEF.8$[p_y/p_x]$ and commutativity of + :

$=b^p|^{c}_{p_x} + b^p|^{!c}_{p_x}$

By DEF.8 and commutativity of + :

$=b^p$

□

Our second lemma asserts that partitions $\textcircled{1}$ and $\textcircled{7}$, represented by the beating functions $b^p|^{!fl}_{p_y}|^{w}_{p_x}$ and $b^p|^{!c}_{p_x}|^{!c}_{p_y}$ respectively, are not under p_y's control.

Lemma 4.3.3.

$$(b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})|^{c}_{p_y} = \mathbf{b_0^p}$$

Proof.

$(b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})|^{c}_{p_y}$

By PROP.5$[p_y/p_x]$:

$=(b^p|^{w}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})|^{c}_{p_y}$

By PROP.3 and identity of + :

$=(b^p|^{!c}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})|^{c}_{p_y}$

By commutativity of + and restrictions and distributivity of + on restrictions :

$=((b^p|^{w}_{p_x} + b^p|^{!c}_{p_x})|^{!c}_{p_y})|^{c}_{p_y}$

By PROP.6$[p_y/p_x, (b^p|^{w}_{p_x} + b^p|^{!c}_{p_x})/b^p]$:

$=\mathbf{b_0^p}$

□

Our third lemma asserts that NNRW and LCE imply LFB.

Lemma 4.3.4. *NNRW $\wedge$ LCE $\Rightarrow$ LFB.*

Proof. We show the contrapositive of the lemma; Let $\preceq$ be a ranking function violating LFB. Formally, $p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \not\Leftrightarrow p_x \preceq(b^p)\, p_y$. We show that $(p_x \preceq (b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \not\Leftrightarrow p_x \preceq(b^p)\, p_y) \Rightarrow$ **false** under the assumption that $\preceq$ satisfies both LCE and NNRW properties.

$p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \not\Leftrightarrow p_x \preceq(b^p)\, p_y$

$\Rightarrow p_x \not\preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \preceq(b^p)\, p_y \vee p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \not\preceq(b^p)\, p_y$ (I)

Consider the left disjunct only :

$p_x \not\preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \preceq(b^p)\, p_y$

Using Lemma 4.3.3 :

$\Rightarrow ((b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})|^{c}_{p_y} = \mathbf{b^p_0}) \wedge p_x \not\preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \preceq(b^p)\, p_y$

By LCE.II$[p_y/p_x, p_x/p_y, (b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})/b^p_1, (b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})/b^p_2]$:

$\Rightarrow p_x \not\preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})\, p_y \wedge p_x \preceq(b^p)\, p_y$

By the contrapositive of NNRW.II$[(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y})/b^p_1, b^p|^{!fl}_{p_x}/b^p_2]$:

$\Rightarrow p_x \not\preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_y}|^{w}_{p_x} + b^p|^{!c}_{p_x}|^{!c}_{p_y} + b^p|^{!fl}_{p_x}|^{w}_{p_y})\, p_y \wedge p_x \preceq(b^p)\, p_y$

By Lemma 4.3.2 and commutativity of + and restrictions :

$\Rightarrow p_x \not\preceq(b^p)\, p_y \wedge p_x \preceq(b^p)\, p_y$

$\Rightarrow$**false** (II)

Consider the right disjunct only :

$p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \not\preceq(b^p)\, p_y$

Using Lemma 4.3.3$[p_y/p_x, p_x/p_y]$:

$\Rightarrow ((b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x})|^{c}_{p_x} = \mathbf{b^p_0}) \wedge p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \wedge p_x \not\preceq(b^p)\, p_y$

By LCE.I$[(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})/b^p_1, (b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x})/b^p_2]$:

$\Rightarrow p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x})\, p_y \wedge p_x \not\preceq(b^p)\, p_y$

By NNRW.I$[(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x})/b^p_1, b^p|^{!fl}_{p_y}/b^p_2]$:

$\Rightarrow p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y} + b^p|^{!fl}_{p_x}|^{w}_{p_y} + b^p|^{!c}_{p_y}|^{!c}_{p_x} + b^p|^{!fl}_{p_y}|^{w}_{p_x})\, p_y \wedge p_x \not\preceq(b^p)\, p_y$

By Lemma 4.3.2 and commutativity of + and restrictions :

$\Rightarrow p_x \preceq(b^p)\, p_y \wedge p_x \not\preceq(b^p)\, p_y$

$\Rightarrow$**false** (III)

From I , II and II :

$p_x \preceq(b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y \not\Leftrightarrow p_x \preceq(b^p)\, p_y) \Rightarrow$ **false**

□

Our forth lemma asserts that NPRL and LFB imply LCE.

Lemma 4.3.5. *NPRL* $\wedge$ *LFB* $\Rightarrow$ *LCE.*

Proof. We separately derive the R.H.S. of each of the LCE axioms from its corresponding L.H.S. under the assumptions of NPRL and LFB.

Consider the L.H.S. of LCE.I:

$b^p_2|^{c}_{p_x} = \mathbf{b^p_0} \wedge p_x \preceq(b^p_1)\, p_y$

Using LFB:

$\Rightarrow b^p_2|^{c}_{p_x} = \mathbf{b^p_0} \wedge p_x \preceq(b^p_1|^{fl}_{p_x} + b^p_1|^{fl}_{p_y})\, p_y$

By DEF.4:

$\Rightarrow b^p_2|^{fl}_{p_x} + b^p_2|^{w}_{p_x} = \mathbf{b^p_0} \wedge p_x \preceq(b^p_1|^{fl}_{p_x} + b^p_1|^{fl}_{p_y})\, p_y$

By definition of a beating function and properties of rational addition :

$\Rightarrow b^p_2|^{fl}_{p_x} = \mathbf{b^p_0} \wedge p_x \preceq(b^p_1|^{fl}_{p_x} + b^p_1|^{fl}_{p_y})\, p_y$

Since $\mathbf{b^p_0}$ is the identity element for + :

$\Rightarrow p_x \preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl})\, p_y$

By NPRL.I$[b_2^p|_{p_y}^{fl}/b_2^p]$:

$\Rightarrow p_x \preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl} + b_2^p|_{p_y}^{fl}|_{p_y}^{l})\, p_y$

By PROP.7$[p_y/p_x, b_2^p/b^p]$:

$\Rightarrow p_x \preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl} + b_2^p|_{p_y}^{fl})\, p_y$

By distributivity:

$\Rightarrow p_x \preceq (b_1^p + b_2^p|_{p_x}^{fl} + b_1^p + b_2^p|_{p_y}^{fl})\, p_y$

By LFB:

$\Rightarrow p_x \preceq (b_1^p + b_2^p)\, p_y = R.H.S.$ (I)

Consider the L.H.S. of LCE.II:

$b_2^p|_{p_x}^{c} = \mathbf{b_0^p} \wedge p_y \not\preceq (b_1^p)\, p_x$

Using LFB:

$\Rightarrow b_2^p|_{p_x}^{c} = \mathbf{b_0^p} \wedge p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl})\, p_x$

By DEF.4:

$\Rightarrow b_2^p|_{p_x}^{fl} + b_2^p|_{p_x}^{w} = \mathbf{b_0^p} \wedge p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl})\, p_x$

By definition of a beating function and properties of rational addition :

$\Rightarrow b_2^p|_{p_x}^{fl} = \mathbf{b_0^p} \wedge p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl})\, p_x$

Since $\mathbf{b_0^p}$ is the identity element for + :

$\Rightarrow p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl})\, p_x$

By the contrapositive of NPRL.II$[p_y/p_x, b_2^p|_{p_y}^{fl}/b_2^p]$:

$\Rightarrow p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl} + b_2^p|_{p_y}^{fl}|_{p_y}^{l})\, p_x$

By PROP.7$[p_y/p_x, b_2^p/b^p]$:

$\Rightarrow p_y \not\preceq (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{fl} + b_1^p|_{p_y}^{fl} + b_2^p|_{p_y}^{fl})\, p_x$

By distributivity:

$\Rightarrow p_y \npreceq (b_1^p + b_2^p|_{p_x}^{fl} + b_1^p + b_2^p|_{p_y}^{fl})\, p_x$

By LFB:

$\Rightarrow p_y \npreceq (b_1^p + b_2^p)\, p_x = R.H.S.$ (II)

From I, II:

NPRL $\wedge$ LFB $\Rightarrow$ LCE

□

We now proceed to prove Theorem 4.3.1.

Proof.

By 4.3.4:

NNRW $\wedge$ LCE $\Rightarrow$ LFB

Therefore:

NNRW $\wedge$ NPRL $\wedge$ LCE $\Rightarrow$ LFB (I)

By 4.3.5

NPRL $\wedge$ LFB $\Rightarrow$ LCE

Therefore:

NNRW $\wedge$ NPRL $\wedge$ LFB $\Rightarrow$ LCE (II)

Combining I $\wedge$ II :

$(\text{NNRW} \wedge \text{NPRL} \wedge \text{LCE} \Rightarrow \text{LFB}) \wedge (\text{NNRW} \wedge \text{NPRL} \wedge \text{LFB} \Rightarrow \text{LCE})$

Simplifying:

NNRW $\wedge$ NPRL $\Rightarrow$ (LCE $\Leftrightarrow$ LFB)

□

We now present our arguably objective, anonymous, neutral, monotonic and thorough SG tournament.

4.4 Tournament Design

A tournament is comprised of a match scheduler and a ranking function. The match scheduler determines the matches comprising the tournament. The ranking function determines the standings of participants. We present our scheduler first then our ranking function then we present our arguments for objectivity, anonymity, neutrality, monotonicity and thoroughness.

Figure 4.2 shows our scheduler which takes a set Q of participants, a claim consisting of a logical formula Ψ and an interpreting structure A. The scheduler first determines the side choices of participants. Participants that fail to take a side are dropped out of the competition. Each pair of distinct participants choosing to take different sides, plays a single SSG where they both take their chosen sides. Each pair of distinct participants choosing to take the same side, plays two SSGs where they switch the sides. The scheduler then returns the results represented as a beating function.

Figure 4.3 shows our ranking function, the fault counting ranking function, in which players are ranked according to the number of faults they incur; the fewer the number of faults the better the rank.

We now argue for the objectivity, anonymity, collusion resistance , neutrality, monotonicity and thoroughness of our tournament design. Our tournament is objective because the resulting ranking solely depends on the results of individual SGs which are objective as we argued earlier. Our tournament is thorough because each pair of participants play at least one SSG which are thorough as we argued earlier.

```
function scheduler($Q$, $\Psi$, $A$)
  let $V = \{q \in Q \mid SSG(\langle\Psi,A\rangle, q, q) = 1\}$
  let $F = \{q \in Q \mid SSG(\langle\Psi,A\rangle, q, q) = 0\}$
  let $P = V \cup F$
  let $result(p_x, p_y) = \begin{cases} SSG(\langle\Psi,A\rangle, p_x, p_y) & , p_x \in V \vee p_y \in F \\ \textbf{undefined} & , otherwise \end{cases}$
  let $side(p) = \begin{cases} \mathbf{s_v} & , p \in V \\ \mathbf{s_f} & , p \in F \end{cases}$
  let $b^p(p_w, p_l, s_{wc}, s_{lc}, s_w) = \begin{cases} result(p_w, p_l) & , s_{wc} = side(p_w) \wedge s_{lc} = side(p_l) \wedge s_w = \mathbf{s_v} \\ & \wedge\ result(p_w, p_l)\ not\ \textbf{undefined} \\ 1 - result(p_l, p_w) & , s_{wc} = side(p_w) \wedge s_{lc} = side(p_l) \wedge s_w = \mathbf{s_f} \\ & \wedge\ result(p_l, p_w)\ not\ \textbf{undefined} \\ 0 & , otherwise \end{cases}$
  return $b^p$
```

Figure 4.2: Tournament Scheduler

$$p_x \preceq_f(b^p)\, p_y = faults^{b^p}(p_x) \leq faults^{b^p}(p_y) \qquad \text{(DEF.9)}$$

$$faults^{b^p}(p) = \sum_{p_w, p_l \in P \wedge s_{wc}, s_{lc}, s_w \in \mathbf{S}} b^p|_p^{fl}(p_w, p_l, s_{wc}, s_{lc}, s_w) \qquad \text{(DEF.10)}$$

Figure 4.3: Fault Counting Ranking Function $\preceq_f$

The arguments for anonymity, monotonicity and neutrality are more involved and we present them in the following subsections.

4.4.1 Monotonicity

According to Theorem 4.4.1, the fault counting ranking function has a non-negative regard for winning and a non-positive regard for losing. We now give an informal proof of Theorem 4.4.1. A win for participant p_x cannot be a fault and therefore cannot increase the number of faults p_x incurs and hence cannot worsen p_x's rank. A loss for participant p_x may be a fault if p_x was not forced. In this case, the fault count of p_x increases and consequently the rank of p_x may only worsen. A loss for participant p_x while p_x is forced would not be counted and the rank of p_x will not change.

Theorem 4.4.1. *$\preceq_f$ satisfies the NNRW and NPRL properties.*

We start our formal proof by proving few auxiliary lemmas.

Lemma 4.4.2. $faults^{b_1^p + b_2^p|_{p_x}^{w}}(p_x) = faults^{b_1^p}(p_x)$.

Proof.

$$\text{L.H.S.} = faults^{b_1^p + b_2^p|_{p_x}^{w}}(p_x)$$

By definition DEF.10$[b_1^p + b_2^p|_{p_x}^{w}/b^p,\ p_x/p]$:

$$= \sum_{p_w, p_l \in P \wedge s_{wc}, s_{lc}, s_w \in \mathbf{S}} (b_1^p + b_2^p|_{p_x}^{w}|_{p_x}^{fl})(p_w, p_l, s_{wc}, s_{lc}, s_w)$$

By distributivity of fault restriction on + :

$$= \sum_{p_w, p_l \in P \wedge s_{wc}, s_{lc}, s_w \in \mathbf{S}} (b_1^p|_{p_x}^{fl} + b_2^p|_{p_x}^{w}|_{p_x}^{fl})(p_w, p_l, s_{wc}, s_{lc}, s_w)$$

By PROP.4 :

$$= \sum_{p_w, p_l \in P \wedge s_{wc}, s_{lc}, s_w \in \mathbf{S}} (b_1^p|_{p_x}^{fl} + \mathbf{b_0^p})(p_w, p_l, s_{wc}, s_{lc}, s_w)$$

By identity of + :

$$= \sum_{p_w, p_l \in P \wedge s_{wc}, s_{lc}, s_w \in \mathbf{S}} b_1^p|_{p_x}^{fl}(p_w, p_l, s_{wc}, s_{lc}, s_w)$$

By DEF.10$[b_1^p/b^p,\ p_x/p]$:

$$= faults^{b_1^p}(p_x) = R.H.S.$$

□

Lemma 4.4.3. $faults^{b_1^p + b_2^p|_{p_y}^{w}}(p_x) \geq faults^{b_1^p}(p_x)$.

Proof.

$$\text{L.H.S.} = faults^{b_1^p + b_2^p|^w_{p_y}}(p_x)$$

By definition DEF.10$[b_1^p + b_2^p|^w_{p_y}/b^p,\ p_x/p]$:

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p + b_2^p|^w_{p_y}|^{fl}_{p_x})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By distributivity of fault restriction on + :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p|^{fl}_{p_x} + b_2^p|^w_{p_y}|^{fl}_{p_x})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By definition of beating functions :

$$\geq \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} b_1^p|^{fl}_{p_x}(p_w,p_l,s_{wc},s_{lc},s_w)$$

By identity of + :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} b_1^p|^{fl}_{p_x}(p_w,p_l,s_{wc},s_{lc},s_w)$$

By DEF.10$[b_1^p/b^p,\ p_x/p]$:

$$= faults^{b_1^p}(p_x) = R.H.S.$$

□

Lemma 4.4.4. $faults^{b_1^p + b_2^p|^l_{p_x}}(p_x) \geq faults^{b_1^p}(p_x)$.

Proof.

$$\text{L.H.S.} = faults^{b_1^p + b_2^p|^l_{p_x}}(p_x)$$

By definition DEF.10$[b_1^p + b_2^p|^l_{p_x}/b^p,\ p_x/p]$:

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p + b_2^p|^l_{p_x}|^{fl}_{p_x})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By distributivity of fault restriction on + :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p|^{fl}_{p_x} + b_2^p|^l_{p_x}|^{fl}_{p_x})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By definition of beating functions :

$$\geq \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} b_1^p|_{p_x}^{fl}(p_w,p_l,s_{wc},s_{lc},s_w)$$

By DEF.10[b_1^p/b^p, p_x/p]:

$$= faults^{b_1^p}(p_x) = R.H.S.$$

□

Lemma 4.4.5. $faults^{b_1^p+b_2^p|_{p_y}^l}(p_x) = faults^{b_1^p}(p_x)$.

Proof.

L.H.S. $= faults^{b_1^p+b_2^p|_{p_y}^l}(p_x)$

By definition DEF.10[$b_1^p + b_2^p|_{p_y}^l/b^p$, p_x/p]:

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p + b_2^p|_{p_y}^l|_{p_x}^{fl})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By distributivity of fault restriction on + :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p|_{p_x}^{fl} + b_2^p|_{p_y}^l|_{p_x}^{fl})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By PROP.2 :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} (b_1^p|_{p_x}^{fl} + \mathbf{b_0^p})(p_w,p_l,s_{wc},s_{lc},s_w)$$

By identity of + :

$$= \sum_{p_w,p_l \in P \wedge s_{wc},s_{lc},s_w \in \mathbf{S}} b_1^p|_{p_x}^{fl}(p_w,p_l,s_{wc},s_{lc},s_w)$$

By DEF.10[b_1^p/b^p, p_x/p]:

$$= faults^{b_1^p}(p_x) = R.H.S.$$

□

We now proceed to prove Theorem 4.4.1.

Proof.

Suppose that the L.H.S. of NNRW.I holds for $\preceq_f$:

L.H.S. $= p_x \preceq_f(b_1^p)\, p_y$

By definition DEF.9[b_1^p/b^p]:

$\Rightarrow faults^{b_1^p}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.2:

$\Rightarrow faults^{b_1^p+b_2^p|_{px}^w}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.3[p_x/p_y,p_y/px]:

$\Rightarrow faults^{b_1^p+b_2^p|_{px}^w}(p_x) \leq faults^{b_1^p+b_2^p|_{px}^w}(p_y)$

By definition DEF.9[$b_1^p + b_2^p|_{p_x}^w/b^p$]:

$$p_x \preceq_f(b_1^p + b_2^p|_{p_x}^w)\, p_y = \text{R.H.S.} \tag{I}$$

Suppose that the L.H.S. of NNRW.II holds for $\preceq_f$:

L.H.S. $= p_x \preceq_f(b_1^p + b_2^p|_{p_y}^w)\, p_y$

By definition DEF.9[$b_1^p + b_2^p|_{p_y}^w/b^p$]:

$\Rightarrow faults^{b_1^p+b_2^p|_{py}^w}(p_x) \leq faults^{b_1^p+b_2^p|_{py}^w}(p_y)$

By lemma 4.4.2:

$\Rightarrow faults^{b_1^p+b_2^p|_{py}^w}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.3:

$\Rightarrow faults^{b_1^p}(p_x) \leq faults^{b_1^p}(p_y)$

By definition DEF.9[b_1^p/b^p]:

$$p_x \preceq_f(b_1^p)\, p_y = \text{R.H.S.} \tag{II}$$

Suppose that the L.H.S. of NPRL.I holds for $\preceq_f$:

L.H.S. $= p_x \preceq_f(b_1^p)\, p_y$

By definition DEF.9$[b_1^p/b^p]$:

$\Rightarrow faults^{b_1^p}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.5:

$\Rightarrow faults^{b_1^p+b_2^p|^l_{p_y}}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.4$[p_x/p_y, p_y/px]$:

$\Rightarrow faults^{b_1^p+b_2^p|^l_{p_y}}(p_x) \leq faults^{b_1^p+b_2^p|^l_{p_y}}(p_y)$

By definition DEF.9$[b_1^p+b_2^p|^l_{p_y}/b^p]$:

$p_x \preceq_f(b_1^p+b_2^p|^l_{p_y})\, p_y =$ R.H.S. (III)

Suppose that the L.H.S. of NPRL.II holds for $\preceq_f$:

L.H.S. $= p_x \preceq_f(b_1^p+b_2^p|^l_{p_y})\, p_y$

By definition DEF.9$[b_1^p+b_2^p|^l_{p_y}/b^p]$:

$\Rightarrow faults^{b_1^p+b_2^p|^l_{p_y}}(p_x) \leq faults^{b_1^p+b_2^p|^l_{p_y}}(p_y)$

By lemma 4.4.4$[p_x/p_y, p_y/p_x]$:

$\Rightarrow faults^{b_1^p+b_2^p|^l_{p_y}}(p_x) \leq faults^{b_1^p}(p_y)$

By lemma 4.4.5:

$\Rightarrow faults^{b_1^p}(p_x) \leq faults^{b_1^p}(p_y)$

By definition DEF.9$[b_1^p/b^p]$:

$p_x \preceq_f(b_1^p)\, p_y =$ R.H.S. (IV)

□

4.4.2 Anonymity

The ranking resulting from our tournament are solely based on skills that participants demonstrate their possession or lack during the tournament because our scheduler and ranking functions ignore participants' identities. Furthermore, according to Theorem 4.4.6 our ranking function has the limited collusion effect property.

Theorem 4.4.6. $\preceq_f$ *satisfies the LCE property.*

The proof of Theorem 4.4.6 falls immediately from Theorem 4.3.1 and Lemma 4.4.7 which states that the fault counting ranking function has the LFB property.

Lemma 4.4.7. $\preceq_f$ *satisfies the LFB property*

Proof.

Consider the L.H.S of LFB applied to $\preceq_f$

L.H.S. $= p_x \preceq_f (b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y})\, p_y$

By definition DEF.9$[b^p|^{fl}_{p_x} + b^p|^{fl}_{p_y}/b^p]$:

$faults^{b^p|^{fl}_{p_x}+b^p|^{fl}_{p_y}}(p_x) \leq faults^{b^p|^{fl}_{p_x}+b^p|^{fl}_{p_y}}(p_y)$

By definition DEF.10, distributivity of fault restriction on +, PROP.1 and identity of + :

$faults^{b^p}(p_x) \leq faults^{b^p}(p_y)$

By definition DEF.9:

$p_x \preceq_f (b^p)\, p_y = R.H.S.$

□

4.4.3 Neutrality

Even though verifiers and falsifiers do not play the same number of games according to our scheduler, the maximum number of faults that both verifiers and falsifiers can make is the same. Furthermore, every participant, regardless of their chosen side, can make a single fault at most against every other participant. To illustrate this point, consider a tournament with n_v participants choosing to be verifiers and n_f participants choosing to be falsifiers. According to our scheduler, verifiers play two SSGs against every other verifier and one SSG against every other falsifier for a total of $2 \cdot (n_v - 1) + n_f$ SSGs. Falsifiers play two SSGs against every other falsifier and one SSG against every other verifier for a total of $2 \cdot (n_f - 1) + n_v$. Even though verifiers and falsifiers play a different number of games, only $n_v + n_f - 1$ games can contribute to the final score of every verifier and every falsifier. A verifier takes on the verifier role in $n_v + n_f - 1$ out of the $2 \cdot (n_v - 1) + n_f$ it plays. These are the games in which a verifier is not forced and can make a fault. In the remaining $n_v - 1$ games, a verifier takes on the falsifier role against other verifiers and although it may lose, this loss does not count as a fault and is therefore ignored by the ranking function.

4.5 Related Work

4.5.1 Tournament Ranking Functions

Rating methods can be used to rank tournament participants. There is a vast body of literature on the topic of *heuristic* [6] rating methods aiming to estimate the skill level of participants such as the Elo [9] rating method. [21] gives a recent comprehensive overview of rating methods used in sports tournaments. Our work differs from this vast body of literature in two important aspects. First, our axioms and ranking method are the first to be developed for an extended framework that we developed specifically to capture some of the peculiarities of SG tournaments such as side choice and forcing. Second, our work is the *first* to be concerned with collusion resilience.

In [27], Rubinstein provides an axiomatic treatment of tournament ranking functions that bears some resemblance to ours. Rubinstein's treatment was developed in a primitive framework where "beating functions" are restricted to complete, asymmetric relations. Rubinstein showed that the points system, in which only the winner is rewarded with a single point is *completely* characterized by the following three *natural* axioms:

- anonymity which means that the ranks are independent of the names of participants,
- positive responsiveness to the winning relation which means that changing the results of a participant p from a loss to a win, guarantees that p would have a better rank than all other participants that used to have the same rank as p, and
- Independence of Irrelevant Matches (IIM) which means that the relative ranking of two participants is independent of those matches in which neither is involved.

Our LCE axioms are, in some sense, at least as strong as Rubinstein's IIM because, according to LCE, the relative rank of some participant p_x w.r.t. another participant p_y cannot be worsened by games that p_x does not participate in nor can it be improved by games that p_y does not participate in.

In [12], the authors provide an axiomatic study of several ranking functions that are based on rating methods. Eight ranking methods, including the points system, and fourteen different axioms, including Rubinstein's IIM, are studied in the paper. Each of the ranking methods is analyzed to determine the axioms it possesses. Only the points system possesses the IIM axiom. The IIM axiom is, however, considered to be an undesirable axiom to have because it is thought to prevent the ranking function from making up for any advantage given to certain participants by a tournament schedule that contains only a subset of the games that would be played in a round robin tournament. Again, none of these ranking functions is specifically developed for SG tournaments. Also, we use a round-robin-like tournament and IIM-like axioms are not undesirable to have.

4.5.2 Tournament Scheduling

There is also a vast body of literature related to tournament scheduling. Different design goals give rise to families of tournament schedulers. We focus our discussion on scheduling two major families of tournaments: double round robin and elimination tournaments.

In a double round robin tournament, every pair of participants play two matches where they alternate their roles (e.g. as the host team or as the white player). This is generally considered a fair selection of each participant's opponents during the course of the tournament. It is also considered to neutralize any advantage that a match may give to one of the participants. As we discussed earlier, when matches cannot end with a tie, the points system where the winner is rewarded with a single point is a widely accepted ranking algorithm for round robin tournaments [27]. Unfortunately, a monotonic, collusion-resistant ranking function has to completely ignore the results of those games where both participants are forced to take sides opposite to the sides of their choice. Since the number of participants choosing to take the verifier side is not necessarily the same as the number of participants choosing to take the falsifier side, verifiers and falsifiers participate in a different number of non-ignored games and neutrality is jeopardized.

The matches of a double round robin tournament of n participants are commonly modeled as the edges of the complete graph K_n. A double round robin tournament schedule partitions the edge set with no two adjacent edges in the same partition [17]. Double round robin tournament schedules are considered static as matches are independent of the current standings of participants. Double round robin tournament schedules are chosen to optimize other psychological or logistical objectives. For example to minimize breaks and carry over effects in the home-away patterns. An annotated bibliography of double round robin tournament scheduling is given in [17]. Fortunately, logistic and psychological concerns that apply to classical sports do not carry over to computational problem solving competitions are often held online.

A downside of round robin tournaments is that they often involve matches that are uninteresting to spectate for various reasons such as matches between two "weak" participants, matches between a "weak" and a "strong" participant where the result is pretty much expected, and matches where the result matters for one of the participants but not as much for the other. This is also not an issue for online computational problem solving competitions where participants are usually interested in the match outcome rather than in spectating matches. Another downside of round robin tournaments is the potential of collusion where a set of colluding participants lose on purpose against a specific participant in order to inflate that participant's rank. While this is an issue with round robin sports tournaments, in online computational problem solving competitions, this issue can be aggravated by the potential of Sybil identities.

Elimination tournaments avoid the pitfalls of round robin tournaments by dynamically scheduling matches only between "strong" participants that have a chance of winning the tournament. Participants that lose a single game are considered "weak" and are eliminated from the tournament. Collusion is not effective in elimination tournaments because losing a match in an elimination tournament does not give the winner an advantage against other participants.

Single elimination tournament schedules are modeled as trees with internal nodes representing games. Participants start at the leaves and winners flow towards the root. The assignment of participants to leaves is called seeding. The probability of a participant winning the tournament depends on the number and the strengths of opponents it may meet on their path to the root. Unless the number of participants is a power of two [1] and it is safe to assume that all participants are of equal strength, a single elimination tournament cannot be considered neutral.

In fact, there is a large body of literature that is concerned of biasing elimination tournaments to favor the stronger participants according to a precalculated rating. The rationale there is that elimination tournaments are only used for competition finals and that the better winning chances in the finals is considered to be a reward for ranking high in the initial phase of the competition. In the literature, this is called increasing the *predictive power* [28] of the tournament. The predictive power of a tournament is the probability that the best participant wins the tournament. Some variants of the elimination tournament skew the tree to shorten the paths taken by the strongest participant [32]. Other variants, such as the McIntyre System, add more paths to the root for the top participant(s). Other variants add more paths to the root for all participants such as the double elimination tournaments. In those variants, the tournament is no longer a tree. There are also variants that dynamically seed the participants after each round [15], [29], [11], [32].

It is however not possible to have a correct elimination tournament of semantic

[1] Unless the number of participants is a power of two, some participants will have to play more games and are therefore at a disadvantage.

games with three or more participants. Simply because the tournament may get stuck when it is time for two verifiers (or falsifiers) to play. Forcing one of the participants to take the opposite side sacrifices neutrality.

4.5.3 Match-Level Neutrality

There are games that offer asymmetric roles to players where participants in certain roles have an advantage over participants in other roles. For example, in chess there are the white and black participant roles where the white role provides the participant with the first move advantage. In soccer, each team attacks a different goal. One team can have an advantage due to wind direction for example. In SGs, participants taking the verifier role have an advantage when the underlying claim is true and participants taking the falsifier role have an advantage when the underlying claim is false.

There are *generic* approaches to restore fairness either in a single game round or across multiple game rounds. One approach is to play two game rounds where participants alternate their roles. For example, in soccer, matches are split in halves where teams switch the goal they attack (and the team kicking off the half-match). In chess tournaments, it is often the responsibility of the administrator to ensure that each participant plays, as nearly as possible, the same number of games as White and Black. This approach is often combined with round robin tournaments to neutralize the home game advantage. The combined approach is called the double round robin tournament. As we mentioned earlier, a monotonic, collusion-resistant ranking function has to completely ignore the results of those games where both participants are forced to take sides opposite to the sides of their choice. Since the number of participants choosing to take the verifier side is not necessarily the same as the number of participants choosing to take the falsifier side, verifiers and falsifiers participate in a different number of non-ignored games and neutrality is jeopardized.

Another approach is adding compensation points for the participants at a disadvantage or subtracting compensation points form participants at an advantage. An example is the Komi points added to the black participant in the game Go. In a tournament where all participants play the same number of games, the fault counting ranking function is equivalent to subtracting compensation points from participants winning against participants at the disadvantage of being forced.

A third approach is the Pie rule in reference to a class of logical games called cut-and-choose games [14]. In the traditional cut-and-choose game one participant cuts a piece of cake into two smaller pieces; then the opponent chooses one of the pieces and eats it, leaving the other one for the cutter. This mechanism is supposed to put pressure on the cutter to cut the cake fairly.

The pie rule can be applied to games with a demonstrated first move advantage as follows: participants are first assigned roles at random. The first participant makes a move; then the second participant gets to choose which side whether or not to swap roles with the first participant. This puts pressure on the first participant not to take advantage of the first move. This rule has been used in board games such as Hex and Mancala. It has also been applied to an extended version of go where the first move is to select the amount of Komi points to compensate the black participant. The Pie rule is not applicable to selecting a side in SGs because there are only two possible moves one that is good and one that is bad.

There are, also, game specific approaches to restore fairness that involve tweaking the rules of the game so that each role has a different form of advantage. For example, in soccer, one team that gets to choose the goal to attack and the other gets the kick-off. In chess, different starting configurations where the white is missing more pieces than the black were proposed [26]. We do not see this approach applicable to semantic games as well.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this dissertation we developed a sports-like computational problem solving competition that can be organized with arbitrarily low overhead on the admin because participants assist in the evaluation of their opponents. At the same time, our competition maintains five desirable properties, namely: objectivity, anonymity, monotonicity, neutrality, and thoroughness.

We proposed the idea of organizing sports-like open online computational problem solving competitions as tournaments of semantic games. The use of semantic games of interpreted predicate logic sentences specifying computational problems ensures that participants are correctly evaluated by their peers. We also developed a simplified version of semantic games for which it is simpler to codify a strategy for playing them. Furthermore, our simplified version of semantic games enable participants to thoroughly evaluate their opponents.

We also sketched out a structured interaction space that we specifically designed to hold semantic-game-based computational problem solving competitions. Our purpose was to give our readers a concrete sense of what is it like to organize or to participate in a semantic-game-based computational problem solving competition.

We also reported on the surprising discovery that it is in fact possible to fend against collusion potential in semantic game tournaments by using certain ranking functions. We presented the first of its kind, formal characterization of collusion-resistant ranking functions for SG tournaments. We also presented a *representation theorem* of ranking functions possessing the limited collusion effect property as well as other basic monotonicity properties. In essence, we showed that under basic monotonicity properties of ranking functions, the limited collusion effect property is logically equivalent to using a ranking function that is based on a generalized form of fault counting.

Finally, we presented a specific SG-based tournament design and argued for its objectivity, anonymity, neutrality, monotonicity and thoroughness.

5.2 Future Work

There are three directions in which we would like to extend this dissertation. We shall describe them in the following subsections.

5.2.1 Utilizing Relations Between Computational Problem Solving Labs

It is possible to make meta-claims about the relationship between other claims. For example, the minimum graph basis claim from Section 3.5.3 is equivalent to a claim about the number of source nodes of the corresponding strongly connected components dag. Like regular claims, it is possible to define CPSLs around meta-claims. However, we see a potential to further utilize meta-claims to translate strategies across labs.

5.2.2 Social Computing

Social computing technologies such as blogs, instant messaging, wiki systems have been used to amend the structured interactions of FoldIt. Social computing technologies increase participants' engagement and enhance the diffusion of knowledge among community members. In future, we would like to incorporate social computing technologies into the design of CPSLs.

5.2.3 Evaluating Thoroughness

Devising a suitable measure for how thorough is one participant's evaluation of other participants is important to encourage participants to improve the thoroughness of their evaluation as well as to give a sense of the reliability of competition results.

REFERENCES

[1] ISO/IEC standard 14977 - Information technology - Syntactic metalanguage - Extended BNF.

[2] Project Euler. Website. http://projecteuler.net/.

[3] Rosetta The premier software suite for macromolecular modeling. Website. https://www.rosettacommons.org/.

[4] The international SAT Competitions. Website. http://www.satcompetition.org/.

[5] EteRNA. Website, 2011. http://eterna.cmu.edu/.

[6] J.D. Beasley. *The Mathematics of Games*. Dover books on mathematics. Dover Publications, Incorporated, 2006.

[7] Seth Cooper, Adrien Treuille, Janos Barbero, Andrew Leaver-Fay, Kathleen Tuite, Firas Khatib, Alex Cho Snyder, Michael Beenen, David Salesin, David Baker, and Zoran Popović. The challenge of designing scientific discovery games. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 40–47, New York, NY, USA, 2010. ACM.

[8] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, April 2011.

[9] A.E. Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[10] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):pp. 9–15, 1962.

[11] Mark E. Glickman. Bayesian locally optimal design of knockout tournaments. *Journal of Statistical Planning and Inference*, 138(7):2117 – 2127, 2008.

[12] Julio Gonzlez-Daz, Ruud Hendrickx, and Edwin Lohmann. Paired comparisons analysis: an axiomatic approach to ranking methods. *Social Choice and Welfare*, 42(1):139–169, 2014.

[13] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[14] Wilfrid Hodges. Logic and games. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2009 edition, 2009.

[15] F. K. Hwang. New concepts in seeding knockout tournaments. *The American Mathematical Monthly*, 89(4):pp. 235–239, 1982.

[16] P. Ipeirotis, F. Provost, V. Sheng, and J. Wang. Repeated labeling using multiple noisy labelers. *This work was supported by the National Science Foundation under GrantNo. IIS-0643846, by an NSERC P, Vol*, 2010.

[17] Graham Kendall, Sigrid Knust, Celso C. Ribeiro, and Sebastin Urrutia. Scheduling in sports: An annotated bibliography. *Computers Operations Research*, 37(1):1 – 19, 2010.

[18] Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. The future of crowd work. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW '13, pages 1301–1318, New York, NY, USA, 2013. ACM.

[19] J. Kulas and J. Hintikka. *The Game of Language: Studies in Game-Theoretical Semantics and Its Applications*. Synthese Language Library. Springer, 1983.

[20] Karim R Lakhani, Kevin J Boudreau, Po-Ru Loh, Lars Backstrom, Carliss Baldwin, Eric Lonstein, Mike Lydon, Alan MacCormack, Ramy A Arnaout, and Eva C Guinan. Prize-based contests can provide solutions to computational biology problems. *Nature Biotechnology*, 31(2):pp. 108–111, 2013.

[21] A.N. Langville and C.D. Meyer. *Who's #1?: The Science of Rating and Ranking*. Princeton University Press, 2012.

[22] Karl J. Lieberherr, Ahmed Abdelmeged, and Bryan Chadwick. The Specker Challenge Game for Education and Innovation in Constructive Domains. In *Keynote paper at Bionetics 2010, Cambridge, MA, and CCIS Technical Report NU-CCIS-2010-19*, December 2010. http://www.ccs.neu.edu/home/lieber/evergreen/specker/paper/bionetics-2010.pdf.

[23] Jordi Petit, Omer Giménez, and Salvador Roura. Jutge.org: an educational programming judge. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 445–450, New York, NY, USA, 2012. ACM.

[24] Ahti Pietarinen. Games as formal tools vs. games as explanations. Technical report, 2000.

[25] Karl Raimund Popper. *Conjectures and refutations: the growth of scientific knowledge, by Karl R. Popper*. Routledge, London, 1969.

[26] Thomas H. Quinn. Level the playing field: Nullifying first-move advantage in chess. http://cargocollective.com/tomquinn/Level-the-Playing-Field-Nullifying-First-Move-Advantage-in-Chess, may 2011.

[27] Ariel Rubinstein. Ranking the participants in a tournament. *SIAM Journal on Applied Mathematics*, 38(1):pp. 108–111, 1980.

[28] Dmitry Ryvkin and Andreas Ortmann. Three prominent tournament formats: Predictive power and costs. Cerge-ei working papers, The Center for Economic Research and Graduate Education - Economic Institute, Prague, 2006.

[29] Allen J. Schwenk. What is the correct way to seed a knockout tournament? *The American Mathematical Monthly*, 107(2):pp. 140–150, 2000.

[30] TopCoder. The TopCoder Community. Website, 2009. http://www.topcoder.com/.

[31] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 319–326, New York, NY, USA, 2004. ACM.

[32] Thuc Duy Vu. *Knockout Tournament Design: A Computational Approach*. PhD thesis, Stanford University, Department of Computer Science, 2010.