Crowdsourcing Formal Science using Games and Logic FSCP with Reductions: An Improved Platform for Crowdsourcing Formal Science

Ahmed Abdelmeged
Northeastern University
mohsen@ccs.neu.com

Karl Lieberherr
Northeastern University
lieber@ccs.neu.edu

ABSTRACT

We present the Formal Science Crowdsourcing Platform (FSCP). FSCP represents claims as interpreted predicate logic formulas. FSCP-based crowdsourcing systems focus a crowd of scholars on examining a family of claims to separate the true claims from false ones. Scholars examine a claim through participating in a substantiation game. Substantiation games are built on top of Hintikka's Game Theoretical Semantics (GTS). Furthermore, FSCP collects the defense and attack strategies on claims.

We also present an approach to evaluate scholars that we believe is new. We also present two approaches to estimate the truth likelihood of claims. We report on our experience with using an earlier version of FSCP in class.

Keywords

Crowdsourcing, Human computation, STEM innovation and education, epistemology, dialogic games, Karl Popper, mechanism design, social welfare, logic, defense strategies, games and quantifiers, virtual communities.

1. SELF

SSS is an international forum for researchers and practitioners in the design and development of distributed systems with self-* properties: (classical) self-stabilizing, self-configuring, self-organizing, self-managing, self-repairing, self-healing, self-optimizing, self-adaptive, and self-protecting.

SCG is self-protecting. A malicious scholar cannot disturb the system?

SCG is self-managing. Each game progresses the system: see the progress claim.

SCG is self-repairing. False claims wll eventually be eliminated?

SCG serves all three tenants of the academic mission, namely, research, education, and outreach.

2. COMPUTATIONAL PROBLEMS

2.1 Kinds of Crowdsourcing

To be submitted for review to the HCOMP 2013 Conference on Human Computation and Crowdsourcing November 7-9, 2013 - Palm Springs, California USA. Due May 1, 2013 5pm, **version March 25**

$http://www.academia.edu/963662/The\_Promise\_of\_Idea\_Crowdsourci$

Do we do knowledge discovery and management and broadcast search (crowd wisdom)? Distributed human intelligence tasking.

- Tools to help with predicate logic

  https://files.ifi.uzh.ch/rerg/arvo/ftp/papers/LOPSTR98.pdf

  University of Zurich

  Lot's of expertise in formal methods is around.

  Users of SCG only need to understand syntax and semantics of logical formulas. The semantics involves the concept of a structure/model and whether a formula holds in a structure. The concept of Skolem function is needed. They need to understand how a logical formula is translated into a semantical game between a verifier and falsifier.

With logics that have semantic games we can express lots of different computational problems for SCG-style crowdsourcing. Sometimes one logic formula is enough to define a useful computational problem. Sometimes we have a decision problem for an infinite set of logical formulas.

The logical formulas that define a computational problem are often trivially true. But there are exceptions.

Computational claims have the following form:

$Claim(inputParameters, outputParameters) =$
$\forall i \in Input : satisfyI(i, inputParameters)$
$\exists s \in Solution : satisfyO(s, outputParameters)$
$predicate(i, s)$

The set of inputParameters or outputParameters maybe empty.

2.2 Decision/Search Problems

2.2.1 *Truth of IPLFs*

Decision Problem; undecidable ? although interpreted.

Instance: Given an interpreted second order predicate logic formula. Solution: true/false

2.2.2 *Truth of first-order IPLFs: FIN-MC*

Decision Problem; PSPACE-complete

http://www.lsv.ens-cachan.fr/ goubault/Complexite/fagin.pdf

$\forall F \in FO \forall I \in Model(F) I \models F$

INPUT: a finite structure I, in its standard representation; a first-order sentence F. QUESTION: $I \models F$?

2.2.3 *Truth over structures: FIN-MC(F)*

Decision Problem

Given a fixed first-order formula F, the problem FIN-MC(F) is the following variant: INPUT: a finite structure I, in its standard representation. QUESTION: I |= F?

Can one solve FIN-MC(F) in polynomial time ? If so, for which class of first-order formulae F?

2.2.4 *Truth of Presburger arithmetic claims*

Decision Problem: decidable

Given a set of first-order axioms for Presburger arithmetic.

INPUT: ϕ a proposition in Presburger arithmetic. QUESTION: Is ϕ true or false in Presburger arithmetic?

2.2.5 *Truth of arithmetic claims*

Decision Problem: undecidable

Given a set of first-order axioms for basic arithmetic defining a theory T.

INPUT: ϕ a proposition in basic arithmetic. QUESTION: Is ϕ true or false in T?

2.2.6 *Fagin's Theorem*

An existential second order claim (ESO) defines a problem in NP.

A problem $\pi \in NP \iff$ There is a second order sentence of the form $\phi = \exists R_1 \exists R_2 ... \exists R_m \psi$, where ψ is a first order formula such that $||\phi|| = \pi$.

See

https://wiki.engr.illinois.edu/download/attachments/56000570/notes26.pdf?version=1&modificationDate=1335465330000

CS498MV: Logical Foundations of Computer Science Mahesh Viswanathan vmahesh@cs.uiuc.edu

2.2.7 *IMPORTANT: Logic to Computational Problem*

Each sentence ϕ in logic L defines a computational problem $||\phi|| = \{A | A \models \phi\}$. In other words, given A (a structure), the problem asks if $A \models \phi$.

Example: structure = CNF, ϕ = exists a satisfying assignment.

2.2.8 *One IPLF: Topological Order*

Decision/Search Problem: decidable: decision: always yes; function problem: O(n+m)

$\forall g \in DAG$ $\exists$ sequence s of nodes in g: topologicalOrder(s)

Instance: DAG g. Solution: true or topological order.

2.2.9 *One IPLF: SAT*

Decision/Search Problem

decidable: decision yes/no: NP-complete; function problem: satisfying assignment or none: NP-hard.

$SATLabClaim(S \in \text{CNF}) =$
$\exists J \in$ assignments(S): satisfied(S,J)

$SATLabClaim(S)$ is true iff S is satisfiable.

2.2.10 *One IPLF: Partial SAT*

Decision/Search Problem

$PartialSAT2LabClaim(g1 \in \mathbb{R}) =$
$\forall S \in 2-\text{satisfiable CNF}$
$\exists J \in$ assignments(S):
$fsatisfied(S, J) \geq g1$

Let $g = (\sqrt(5) - 1)/2$. $PartialSAT2LabClaim(0.6)$ is true, $PartialSAT2LabClaim(g)$ is true, $PartialSAT2LabClaim(0.7)$ is false.

2.2.11 *One IPLF: Partial SAT negated*

Decision/Search Problem

$PartialSAT2NLabClaim(g1 \in \mathbb{R}) =$
$\forall \epsilon \in \mathbb{R} s.t. \epsilon > 0$
$\exists S \in 2-\text{satisfiable CNF}$
$\forall J \in assignments(S):$
$fsatisfied(S, J) < g1 + \epsilon$

$g1$ is an output parameter. $PartialSAT2NLabClaim(0.6)$ is false and $PartialSAT2NLabClaim(0.7)$ is true and $PartialSAT2NLabClaim(g)$ is true.

2.2.12 *Bertrand's Postulate*

$Bertrand(n \in \mathbb{N}, n > 1) =$
$\exists k \in [n, 2 \cdot n] : prime(k)$

An arithmetic fact: Between n and $2 \cdot n$ there is always a prime.

$prime(n \in \mathbb{N}) =$
$\forall k s.t. 1 < k < n : \neg divides(k, n)$

2.3 Optimization Problems

Optimization problems are now also expressed using logic but they require a minimum special treatment. We need a predicate $stronger(c_1, c_2)$ to compare claims.

We denote a claim by $c(\vec{x} = \vec{v}) = f(v)$. $\vec{x} = \vec{v}$ is an assignment to the free variables $\vec{x}$ in f. f is the predicate logic formula defining the claim.

$stronger(c(\vec{x} = \vec{v}), c(\vec{x} = \vec{v}'))$ is a predicate that is claim-specific.

$c(\vec{x} = \vec{v})$ is optimum if
$\neg(\exists \vec{v}' : stronger(c(\vec{x} = \vec{v}'), c(\vec{x} = \vec{v}'))$ and $c(\vec{x} = \vec{v})$

Sometimes we don't want to express that a claim is maximum but that a solution is maximum. The pattern for maximization is (s has maximum quality):

$(\exists s \in Solution$
$\neg(\exists s_{better} \in Solution :$
$quality(s_{better} > quality(s)$
) THE FOLLOWING OPTIMIZATION formalization is outdated. There is a simpler way of expressing it without quantifying over scholars and claims and referring to refutation games. See below the HSR example.

We want to express that a claim c(x=v) is optimum: there exists a scholar s1 who has a defense strategy for $c(x = v)$ and a refutation strategy for any stronger claim.

optimum (c(x=v)=f(v) = $\exists s1 \in Scholar$ so that the following two conditions hold: 1.$\forall s2 \in Scholar$ $s1 = c.RG(c(x = v), s1, s2)$ 2.$\forall s2 \in Scholar \forall c(x = v')$ that are stronger than $c(x = v) : s1 = RG(c(x = v'), s2, s1)$

2.3.1 *One IPLF: Max Sat*

Optimization problem; computable: NP-hard.

$MaxSatLabClaim(S \in CNF, q \in \mathbb{N}) =$
$\exists J \in$ assignments(S):
$satisfied(S, J) = q \wedge$
$\forall J_1 \in assignments(S):$
$satisfied(S, J_1) \leq q$

2.3.2 *Algorithm Worst-Case Lab (AWCLab)*

We are given an algorithm A that takes inputs i in INP. Algorithm A returns its resource consumption on an input i as ResourceUse(A,i) as an integer. For example, A might consists of one outermost loop. ResourceUse will count the number of loop iterations. We consider all inputs of size n and want to determine an input i for which ResourceUse(A,i) is maximum.

Predicate logic expression:

$AWCLabClaim(n \in \mathbb{N}, q \in \mathbb{N}, A) =$
$\exists i \in INP :$

$size(i) = n \land ResourceUse(A, i) = q \land$
$\forall i_1 \in INPs.t.size(i_1) = n :$
$ResourceUse(A, i_1) \leq q.$

Free variables are n, q and the algorithm A. The maximum property is expressed with the universal quantifier.

In our implementation we don't permit this level of parameterization. We would have to hardwire the algorithm.

Example:

- $AWCLab(n = 10, q = 20, A = GaleShapley)$

  The GaleShapley algorithm consists of one while loop (if written in the style of a typical algorithm text book []) and ResourceUse(GaleShapley,i) counts the number of iterations on input i. The claim says that 20 iterations is maximum for inputs with 10 men and 10 women.

2.3.3 *Highest Safe Rung (HSR)*

One predicate logic expression defines a claim family = knowledge base to be maintained. Each member of the claim family is a test case for the computational problem to be solved by the lab. If the computational problem is not optimally solved the knowledge base is not clean and will contain false (non-optimal) claims.

As an example, consider exercise 8 in chapter 2 of Kleinberg and Tardos []. For k=2, the problem asks for HSR(n,2,f(n)) so that f(n) grows slower than linearly. The problem asks for an algorithm that correctly finds the highest safe rung but not necessarily with the minimal number of questions but with an asymptotically good solution. There are lots of suboptimal functions f(n). But there is an optimal function that can be computed efficiently.

Ahmed's solution:

- A decision tree has minimum depth:

  $HSR_{minDepth}(n \in \mathbb{N}, k \in \mathbb{N}, q \in \mathbb{N}) =$
  $\forall d \in DTs.t.correct(d, n, k) : depth(d) \geq q.$

- An algorithm computes a minimum depth decision tree:

  $HSR_{minDepthTreeAlgo}(a \in \mathbb{N} * \mathbb{N} => DT) =$
  $\forall n, k \in \mathbb{N} : HSR_{minDepth}(n, k, depth(a(n, k)))$

- An algorithm that computes the depth of the minimum depth decision tree.

  $HSR_{minDepthExpr}(e \in \mathbb{N} * \mathbb{N} => \mathbb{N}) =$
  $\forall n, k \in \mathbb{N} : HSR_{minDepth}(n, k, e(n, k))$

- An algorithm that computes the minimum depth decision tree in linear time (constant c).

  $HSR_{minDepthTreeLinAlgo}(c, n_{min}, n_{max} \in \mathbb{N}) =$
  $\exists a \in \mathbb{N} * \mathbb{N} => DT$
  $\forall n, k \in \mathbb{N} s.t. n_{min} \leq n \leq n_{max}$
  $let(q, r) = RT(a(n, k))$
  $r \leq c \cdot n \land HSR_{minDepth}(n, k, q)$

  RT is an interpreter that runs the algorithm and returns its result in the first element of the return pair and the resource consumption in the second element of the return pair. Standard RT functions are provided and can be user defined. RT could measure the wall clock time, count virtual machine instructions or just simply count the number of loop iterations of an algorithm consisting of a single loop.

  Note that we use an additional binding construct called let. We have now the claim expression, $\forall$, $\exists$ and let as binding constructs. The let binding construct does not change the nature of the Semantical game.

THE FOLLOWING ABOUT HSR IS OUTDATED.

HSR is a traditional decision problem, involving evaluated versions of the predicate logic formula.

$HSRLabClaim(c, n_{min}, n_{max}) =$
$\exists\ algorithm\ DT(\mathbb{N}, \mathbb{N})$
$\forall n, k \in \mathbb{N} \land k < n \land n_{min} < n < n_{max}$
$RT(DT, n, k)) < c \cdot n \land DT(n, k)$ is correct $\land$
$depth(DT(n, k))$ is minimum

RT is the running time of the algorithm. Correctness is expressed using the binary search tree property and that the tree must be tilted as determined by k. The minumum property is expressed by: $\neg(\exists q_1 < q$ and a decision tree $dt : depth(dt) = q_1 \land dt$ is correct).

Example HSRLabClaim:

- $HSR(c = 3, n_{min} = 8, n_{max} = 10^6, n = 25, k = 2, q = 5)$

  This claim says that there is an algorithm to compute the correct and minimum decision tree in time $3 \cdot n$ for n between n_{min} and n_{max}. And that when the algorithm is applied to $n = 25$ and $k = 2$ a decision tree of depth 5 is constructed which is correct and minimum.

We define a second HSR lab, called HSR2. We need a simpler predicate logic formula where the goal is just to go for the correct minimum decision trees.

$HSR2LabClaim(n, k, q)$
$\exists\ algorithm\ DT(\mathbb{N}, \mathbb{N})$
$\forall n, k \in \mathbb{N}$
$\exists q \in \mathbb{N} :$
$k < n \land q < n \land DT(n, k)$ is correct $\land$
$depth(DT(n, k)) = q$ is minimum

- HSR2LabClaim Example:

  $HSR2LabClaim(n = 25, k = 2, q = 5)$. n and k are input parameters and q is an output parameter.

$HSR3LabClaim(n, k, q) =$
$\forall n, k \in \mathbb{N} \land k < n$
$(\exists ep \in CorrectExperimentalPlan(n, k) : depth(ep) = q$

$\land$ (minimum depth)

$\neg(\exists ep_2 \in CorrectExperimentalPlan(n, k) :$
$depth(ep_2) < q)$
)

Here the lab designer does not introduce the concept of a decision tree. Instead it is called a correct experimental plan.

The HSR example shows how labs are organized. They consist of a set of claims that are symbolic evaluations of the predicate logic expression of the lab.

2.4 Counting problems

2.4.1 *One IPLF: #Sat*

Counting problem. #P-complete.

$CountingSatAssignments(S \in CNF) =$
$\exists n \in \mathbb{N}$: n is the number of all satisfying assignments of S.

In order to win, we have to enumerate all satisfying assignments. If we miss some, our claim can be refuted.

2.5 Robotics Computational Problems

$\forall$ noisy sensors that measure partial information
$\forall$ models of the system

$\exists$ state that is most consistent with noisy sensors

3. PROBLEM SOLVING WITH FSCP

An important goal of FSCP is to make the workers better problem solvers by sharpening their intuition through interaction with others.

The problems to be solved: Distinguish true/false claims and substantiate your decision by computing an assignment for the quantified variables jointly with your opponent.

While doing this there is the need to spawn new labs that help with the solution of the current lab. The scholars get a significant new move: define a new lab. But they must substantiate their reduction and show how the old lab reduces to the new one. The reduction lives in a lab with claims that are subject to refutation.

Labs about labs

Claims about claims

We need higher order predicate logic.

Definition: **Weakly/Strongly Solving a lab** means

- **Weakly**: Avoid Contradictions.

  Sharpen our intuition about the claims in the labs.

  Guess a defense strategy for the true claims and, guess a refutation strategy for the false claims.

- **Strongly** Prove Strategy Correct.

  After you have found the defense and refutation strategies, prove them correct.

The two items are successively harder but already the first helps to sharpen our intuition about the lab. Already when at least one scholar can avoid all contradictions, we consider the lab solved.

The weak reductions are useful during problem solving. We can try solution approaches and try to refute them. This is useful for theorem proving where the theorem prover often needs human help.

To sharpen our intuition about a lab it is beneficial to decompose the lab into one or more component labs so that a solution to the component labs will result in a solution for the original lab.

Lab reductions are a useful tool in this process. Finding the right reductions often requires innovation.

Informally, lab L2 is a reduction of lab L1 (L1 reduces to L2) ($L1 < L2$), if a solution for L2 (which we can use as a black box) implies a solution for L1.

Lab reductions are based on claim reductions. A typical use of claim reductions is to start with a claim C, reduce it through a sequence reductions to a claim C_n and then solve C_n directly.

For the formal definition we use a Karp-style reduction with an explicit transformation. A **lab reduction** $L1 < L2$ is a mapping f from L1 to L2 which has the following properties:

- f is a computable function $f : L1.Model- > L2.Model$. We extend f to claims: $f : L1.Claim- > L2.Claim$ and to game histories associated with the claims.

- LabReductionProperty

  $\exists f \in$ computable functions
  $\forall C \in L_1.Claim$
  $\forall h \in$ GameHistories(C)
  $(defend(f(h), f(C)) \Rightarrow defend(h, C)\ and$
  $refute(f(h), f(C)) \Rightarrow refute(h, C))$

  old:

  For all claims c in L1.Claim for all game histories w of the game associated with c: w is a defense of c in L1 iff f(w) is a defense of f(c) in L2 and, w is a refutation of c in L1 iff f(w) is a refutation of f(c) in L2.

Notes: (1) f might not be one-to-one and in that case the inverse function f^{-1} does not exist. (2) The definition of lab reduction does not refer to true or false claims or to defense or refutation strategies.

Which Lemma is more informative?

Lemma: (implied by LabReductionProperty) If L1 < L2, then if L2 has a winning strategy to defend, then L1 has a winning strategy to defend and if L1 does not have a winning strategy to defend, then L2 does not have a winning strategy to defend.

Lemma: (implied by LabReductionProperty) If L1 < L2, then if L2 has a winning strategy to defend, then L1 has a winning strategy to defend and if L2 has a winning strategy to refute, then L1 has a winning strategy to refute.

3.1 Small Steps

Find the right place: Incremental approach

A successful refutation of claim c is viewed as a small step towards a proof of the negation of c. If the proponent is perfect, the successful refutation counts as a proof of !c because the perfect proponent would have found a way to defend if such a defense of !c exists.

A successful defense of claim c is viewed as a small step towards a proof of c. If the opponent is perfect, the successful defense counts as a proof of c because the perfect opponent would have found a way to refute if such a refutation of c exists. Restriction: if the opponent is not perfect, it is possible that c is false and the defense happened because the opponent made a mistake.

3.2 Lab Reduction Examples

3.2.1 Safe Haven

The informal lab description: A safe haven in a directed graph is a vertex that can be reached from any vertex in the graph. Using pseudo-code, develop an algorithm that will compute the number of safe havens in a directed graph in linear time.

Solution ideas: reverse edges, computing strongly-connected component graph of G reduces problem to a DAG, find nodes without predecessors, There must be exactly one node without predecessor in strongly connected component graph.

Lab1: Graph G=(V,E), A structure in the vocabulary (E) of one binary relation is a finite directed graph (V,E). Find the number of safe havens.

$SH(G \in Graph, j \in \mathbb{N}) =$
$\exists shs \in Set(V)$
$\forall v_1 \in V$
$\exists v_2 \in V$
$(\ \exists path\ p : v_1 - > v_2 \iff v_2 \in shs\ ) \wedge size(shs) = j.$

Lab2: Find the number of all nodes from which all other nodes can be reached.

The set of all nodes from which all other nodes can be reached is called a basis of the graph.

This definition comes from Udi Manber's book: Introduction to Algorithms - A creative approach by Addison Wesley:

Exercise 7.92, page 260: A **vertex basis** of a directed graph G=(V,E) is a minimum-size subset B of V with the property that, for each vertex v on V, there is a vertex b in B such that there is a path of length 0 or more from b to v. Prove the following two claims, and then use them to design a linear time algorithm to find a vertex basis in general directed graphs.

a. a vertex that is not on a cycle and has a nonzero indegree cannot be in any vertex basis.

b. A DAG has a unique vertex basis, and it is easy to find.

$Base(G, j)$ means that graph G has a base of size j.

$Base(G \in Graph, j \in \mathbb{N}) =$
$\exists b \in Set(V)$
$\forall v_1 \in V$
$\exists v_2 \in V$
$(\exists path\, p : v_2 -> v_1 \iff v_2 \in b\ ) \wedge size(b) = j.$

Lab3 (reduction) Lab1 < Lab2: reverse all edges.

G1=(V,E1), G2=(V,E2). $\exists f : G1 -> G2 :$
$\forall e = (v_1, v_2) \in E1$
$\exists$ a reverse edge $er = (v_2, v1) \in E2$.

Lab4: Given a DAG G=(V,E) find all nodes from which all other nodes can be reached. This is the same as Lab 2 but Lab 4 is simpler: we only need to deal with directed graphs. This is simpler: we find all nodes without a predecessor. If there is more than one, the base is empty. If there is exactly one, it is the base.

Lab5 (reduction) Lab2 < Lab4: Construct graph of strongly connected components using Tarjan's algorithm. If a node in the DAG is selected, select all nodes in the corresponding strongly connected component.

Lab6: DAG G=(V,E). Find all nodes without predecessor.

Lab7 (reduction) Lab4 < Lab6 if there is exactly one node v without predecessor, from v all nodes in the DAG can be reached.

Lab6 is solved directly using a subalgorithm of Topological Ordering.

3.2.2 NP-Completeness

Lab1: Want to convince ourselves: scholars S1 and S2 problem X is NP-complete

spawned by S1: Lab2: We are already convinced: S1 and S2 problem Y is NP-complete

spawned by S2: Lab3: we are convinced: S1 and S2 X is in NP

spawned by S1: Lab 4 (Reduction): claim is of the form: Exists T ForAll sY in Y: T is polynomial ... transformation T provided by S1: Y < X (Karp) provides transformation from Y to X Consider an arbitrary instance sY of Y. Transform sY using T into sX=T(sY) in polynomial time: (1) if sY in Y, then sX=T(sY) is in X. (2) if sX in X, then sY in Y.

Scholar S1 of Lab1 used the option to spawn an additional Lab that helps in the resolution of Lab1. This is a creative step because it is important to find Y that is known to be NP-complete and that is "close to" X.

A refutation in Lab4 takes the form: find a sY so that (1) or (2) don't hold.

Once Lab2, Lab3 and Lab4 show strong evidence that their claims are true, the claim in Lab1 must also be true.

3.2.3 HSR

Lab1

HSR(n,k)=q

Lab2 M(k,q)=n

Lab3: Reduction Lab1 < Lab2 Transformation T:

Lab4: Modified Pascal Triangle claim: Exists alg A in P to compute values defined by recurrence: $M(k, k) = 2^k M(0, q) = 1 M(k, q) =$ ... Lab5: Reduction Lab2 < Lab4 Transformation T: identity

3.2.4 BFS Lab

Need a better lab name.

Solve the lab through a reduction.

The reduced problem uses a layered graph with n/2 layers. BFS provides the reduction.

s x1

3.2.5 Local versus Global Satisfaction

"Local versus global" is an old theme in computer science and mathematics [4].

Here we illustrate how lab reductions create a connection between different structures and how we can take knowledge from the simpler lab back to the more complex-appearing lab.

Our

```
plan
 reductions to illustrate
 1. from formulas to continuous
    use biased coin, linearity of expectation
 2. From general formulas to symmetric formulas
 3. subset of relations is sufficient
    eliminate noise
```

We investigate combinatorial optimization problems of the following form: Given a sequence of Boolean constraints, find an assignment which satisfies as many as possible. Constraint satisfaction problems appear in many applications.

Maximization problems of this type are naturally formulated as maximum ψ-satisfiability problems [Schaefer (1978)]. ψ is a finite set of logical relations $R_1, \ldots, R_m$ which are used to express the constraints. A ψ-formula S with n variables is a finite sequence of clauses each of the form $R_i(x_1, \ldots, x_{r_i})$. r_i is the rank of R_i and $x_1, \ldots, x_{r_i}$ are a subset of the variables of S. The maximum ψ-satisfiability problem consists of finding, for any ψ-formula S, a Boolean assignment to the n variables satisfying the maximum number of the clauses.

Let τ_ψ be the fraction of the clauses which can be satisfied efficiently in any ψ-formula S.

The ψ-formula lab, called ψ-FormulaLab, has the purpose to approximate τ_ψ and to find efficient algorithms that find such assignments.

The ψ saddle point lab, called ψ-SaddlePointLab, has the purpose to find saddle points in polynomials that depend on ψ. The ψ-SaddlePointLab is a reduction of the ψ-FormulaLab so that if we start with a ψ-formula and translate it to a ψ-polynomial, a solution to the polynomial provides a solution to the formula. We can find good assignments to the ψ-formulas by maximizing polynomials.

But we also want to find "hard" ψ-formulas where only a small fraction can be satisfied, i.e., a fraction close to τ_ψ. This problem we also want to solve in the ψ-SaddlePointLab. A challenge is to map the SaddlePointLab saddle point back into a formula in the FormulaLab. There are many ways to assign a formula to a $\vec{t}$ vector $\vec{t} = (t_{R_1}, \ldots, t_{R_m})$. Fortunately, there is a hardest one, namely a symmetric ψ-formula. If $\vec{t}$ contains irrational numbers, we can find a symmetric ψ-formula whose satisfaction ratio is close to what the SaddlePointLab predicted.

Exercise for reader (solution in [**?**]: The following theorem analyzes subclasses of the regular satisfiability problem.

Theorem 4.1 Let $F(p, q)$ be the class of propositional formulas in conjunctive normal form which contain in each clause at least p positive or at least q negative literals ($p, q > 1$). Let α be the solution of $(1 - x)^p = x^q$ in $(0, 1)$ and let $\tau_{p,q} = 1 - \alpha^q$. Then the fraction $\tau_{p,q}$ of the clauses in any formula $\in F(p, q)$ can be satisfied in time $O(|S||\,\text{clauses}(S)|)$. The set of formulas $\in F(p, q)$ which have an assignment satisfying the fraction $\tau' > \tau_{p,q}$ of the clauses is NP-complete (τ' rational).

Reduction to a continuous min-max-problem

Which fraction can be satisfied in any formula?

Probabilistic approach: biased coin with bias x. Linearity of expectation.

Lab 1 is about formulas

Lab 2 is about polynomials in x in interval [0,1].

Mapping f: formula -> polynomial interpretation -> rational number

(S,J) -> (t,x)

if have a rational $vect$ and x that yield f there is a formula $S = T^{-1}(vect)$. (T(S)= t) and an assignment J to S that yield f. (J sets the fraction x of the clauses to true).

Why? consider gcd of all elements in t.

Need more: symmetry.

defense: can get g in f(S) => can satisfy g in S

There is something special about refutation: cannot get g+eps in f(S) => cannot satisfy g+eps in S

This holds for symmetric formulas but not in general for unsymmetric ones. S may be big if eps is small.

There is a subset of formulas

In the following we sketch how the computation of τ_ψ can be simplified to a discrete minimax problem involving polynomials (a more detailed explanation is in [Lieberherr (1982)]).

τ_ψ is by definition the fraction of the clauses which can be satisfied in all ψ-formulas. First we consider ψ-formulas with at most n variables and let $\tau_{n,\psi}$ be the fraction of clauses which can be satisfied in all such formulas.

For computing $\tau_{n,\psi}$ we determine the worst-case formulas, i.e. the formulas where the smallest fraction of the clauses can be satisfied (by the optimal assignment) among all ψ-formulas with n variables. It is easy to prove that these formulas are symmetric, i.e. they are invariant under permutations of the variables, up to a permutation of the clauses.

Fortunately the worst-case formulas have a nice structure and therefore it is easy to compute an optimal assignment for them. For computing an optimal assignment for a symmetric formula we only have to compute the maximum of a polynomial. This polynomial can be derived by elementary combinatorial analysis.

In this section we prove a theorem which simplifies the computation of τ_ψ to the solution of a continuous minimax problem which does not involve a limit operation. Let $\psi = \{R_1, R_2, \ldots, R_m\}$ be a finite set of relations and let S be a symmetric ψ-formula in which the fraction t_{R_i} of the clauses contains clauses involving relation R_i. In order to compute $\tau_{n,\psi}$ we have to find the worst assignment to the parameters $t_{R_1} \ldots t_{R_m}$ which makes the optimal fraction of satisfiable clauses as small as possible.

$\sum_{i=1}^{m} t_{R_i} = 1, t_{R_i} \geq 0 \; (1 \leq i \leq m))$

t_R	is the fraction of clauses containing relation R
$r(R)$	is the rank of R
$q_s(R)$	is the number of satisfying rows in the truth table of R which contains s ones
$(\alpha)_\beta$	$\frac{\alpha!}{\beta!(\alpha-\beta)!}$, where α, β are positive integers, $\alpha \geq \beta$.

Let

$$\tau'_\psi \;=\; \min_{\substack{t_{R_i} \text{ real} \\ 1 \leq i \leq m}} \; \max_{\substack{0 \leq x \leq 1 \\ \text{real}}} \sum_{i=1}^{m} t_{R_i} \cdot appSAT_x(R_i),$$

$$\sum_{i=1}^{m} t_{R_i} = 1, t_{R_i} \geq 0$$

$$appSAT_x(R) \;=\; \sum_{s=0}^{r(R)} q_s(R) x^s (1-x)^{r(R)-s}.$$

Let S be a ψ-formula containing relation R_i $(1 \leq i \leq m)$ for the fraction t_{R_i} of the clauses. Let $\vec{t} = (t_{R_1}, \ldots, t_{R_m})$. Let

$$appmean_x(\vec{t}) = \sum_{i=1}^{m} t_{R_i} appSAT_x(R_i).$$

How does this relate to the title of this subsection, local versus global?

Consider a conjunctive normal form (CNF) as a sequence of clauses (repetition of clauses is allowed). A CNF is a special kind of ψ-formula. Now consider a property of CNFs that can be considered local: assume that any subset of k clauses is satisfiable. We call such CNFs k-satisfiable. Now consider a global property implied by the local property: which fraction of the clauses can be satisfied in any k-satisfiable CNF? For k=1, the answer is obviously 1/2. For k=2, the answer is, using the reductions mentioned above and the reductions mentioned in the next subsection, $g = (\sqrt(5) - 1)/2$. For general k, the limit is surprisingly 3/4. See the paper by Trevisan on Local versus Global Satisfiability [36].

Another important reduction in this context is noise elimination.

3.2.6 Semantic Games

Put elsewhere. We need a good background on semantic games because they are important to our work. From: http://www.polkfolk.com/docs/Ref-Library/Pierce/Springer,

Semantic games may be viewed as a special class of extensive forms of games that show the flow of semantic information and the distribution of the strategic actions of the players during the actual playing of a game. Variations in the information structure of the players give rise to different kinds of logics, including the IF (independence-friendly) logics introduced in Hintikka and Sandu (1989).

Semantic games seek to establishing when the propositions are true in a model and when they are false in a model.

Among the early ludents was Ernst Zermelo (1871ÂŰ1953), who showed that for a two-player strictly competitive gamewith finitely-many possible positions, a player can avoid losing for only a finite number of moves (if his opponent plays correctly), if and only if the opponent is able to force a win (Zermelo, 1913). The modern received version of the theorem states that every finite, strictly competitive perfect-information two-player game is determined: either player 1 or player 2 has a winning strategy.

In the twentieth century, game interpretations of logic were used, at least occasionally if not systematically, by a number of logicians. Skolem introduced what are known as Skolem functions (Skolem, 1920), and onemay viewthem as winning strategies in the relevant logical game.

These theories are not, in fact, based on purely semantic ideas. An application inwhich theGTS framework is, however, put into practical use is in the construction of knowledge bases (Jackson, 1987).

3.2.7 Noise Elimination

Fix a property L (for Local) on ψ-formulas. Consider a mapping f from ψ-formulas satisfying L to ϕ-formulas satisfying L where the set of relations ϕ is a subset of ψ. Consider a property p over formulas and we want to show $p(\phi$-formulas satisfying $L)$ implies $p(\psi$-formulas satisfying $L)$. Here it is apparent that $p(\phi$-formulas satisfying $L)$ is simpler because we have to deal with fewer relations.

To make the illustration concrete, we consider $\psi =$ all or-relations and $\phi =$ the two or relations A *or* B and $!A$ and $p =$ has an assignment satisfying the fraction g of the clauses.

The transformation f exploits the peculiarities of L. f has to keep the property L invariant and we want f to have the property that if in $f(S)$ we can satisfy the fraction g1 of the clauses then we can satisfy at least fraction g1 of the clauses in S.

The claim for the reduction lab is:
$\exists f\ \forall S \in \psi-$formulas $\forall J \in Assignment$
$(fsat(S, J) \geq fsat(f(S), f(J)))$.

Modify our view of reductions: they take many forms: we offer some standard reductions and give the corresponding predicate logic claims.

LabReductionProperty
$\exists f \in$ computable functions
$\forall C \in L_1.Claim$
$\forall h \in$ GameHistories(C)
$(defend(f(h), f(C)) \Rightarrow defend(h, C)\ and$
$refute(f(h), f(C)) \Rightarrow refute(h, C))$

Standard reduction:
$\exists f \in$ computable functions
$\forall C \in L_1.Claim$
$\forall h \in$ GameHistories(C)
$(defend(h, C) \iff defend(f(h), f(C))\ and$
$refute(h, C) \iff refute(f(h), f(C)))$

Standard many-one reduction??
$\exists f \in$ computable functions
$\forall C \in L_1.Claim$
$(C \iff f(C))$

Important requirement for reduction labs: it ensures that a solution that is always correct for L2 can be converted into a solution that is always correct for L1.

There are many different reductions, all expressible as interpreted predicate logic formulas, that satisfy this criterion.

New View: Reduction labs can use any claim C involving the mapping function f in an existential quantifier. There are a lot of different concepts of reduction. But there must be an implication claim which says that C => LabReductionProperty.

Illustration with mapping from JACM paper. f is a renaming (substitute some x by !x) plus a shortening of clauses (deleting of literals). In the original CNF it is easier to satisfy clauses because the clauses are longer.

defense game history: Have assignment satisfying g in L2, have assignment satisfying g in L1. Note: only one direction.

refutation game history: In L2 exists S where cannot satisfy g+eps implies: in L1 exists S where cannot satisfy g+eps. Note only one direction.

4. BOOTSTRAPPING CROWDSOURCING

Start with a crowdsourcing platform CP, have a set of labs IM_{CP} (for IMprove) to have the crowd find better crowdsourcing algorithms for CP, put new algorithms from IM_{CP} into crowdsourcing platform. CP = CP improved by results from IM_{CP}. Iterate until fixpoint is reached.

```
start with initial CP;
repeat
  results_IM_CP = run IM_CP;
  CP = CP improved by results_IM_CP;
until CP at fixpoint
```

Notes:

1. Any crowdsourcing platform should support this bootstrapping possibility. This requirement enforces a level of sophistication in the crowdsourcing platform. 2. We assume that the improvements don't change the interface of CP but only the algorithms. We could also allow interface changes to CP. 3. What are the knowledge bases for crowdsourcing? 4. CP could get worse if improvements are not really improvements.

5. MORE RELATED WORK

SCG/FSCP is a design for a social media platform where people can join various labs and make contributions in them. A subset of users will create labs both to solve a new problem as well as to solve one of the existing labs.

This paper:
http://www.ics.uci.edu/ singhv/Publications_files/Mechanism_Design_Social_M
Vivek Singh UCI et al.

studies mechanism design for incentivizing social media contributions. Is this paper useful to justify the design of SCG/FSCP.

If not, why is the SCG/FSCP design good? Is an other mechanism design principle applicable?

Do we fill a whole described in: [32]

6. LEARNING-BASED EVALUATON

6.1 Scholar Assessment with SCG

SCG has an natural assessment approach implied by the Scientific Method.

6.1.1 A perfect master teacher is available (gold standard)

input: claim; output: true, false

input: true claim; output: interaction objects that lead to defense

input: true claim, prefix of interaction objects; output: does prefix lead to defense?

MAKE GENERIC

input: false claim. output: first step in refute(c,P,O) that leads to refutation.

input: false claim. Partial elaboration of refute(c,P,O) with next step to be made by O. output: step by O that leads to refutation.

input: true claim. Partial elaboration of refute(c,P,O) with next step to be made by P. output: step by P that leads to defense.

The above perfect master teacher capabilities can be used to guide and assess the scholar.

6.1.2 No perfect master teacher

We still have the blame assigned based on the refutation protocol outcome. The scholar who gets into a contradiction gets blamed (see table **??**).

reason for loss (e.g., proposed claim refutation) not easy to find claim could be false and properly attacked (error in propose) claim could be false and improperly attacked and improperly defended (error in propose,provide and solve) claim could be true but not properly defended (error in provide or solve)

don't know in which situation we are. How does SCG help?

Yes, SCG helps: reason: (oB column in Figure **??**.

6.2 Learning Science and SCG

This is finer-grained than just counting the number of contradictions a scholar creates. What is the connection?

I understand your concerns about incorporating learning scientists. I believe, SCG has very good learning science built in. Below is a description how learning happens and how it is measured in SCG.

In an SCG lab, learning happens during the elaboration of the refutation protocol for a claim. When a claim is defended or refuted, there is a sequence S of instances and solutions which has been produced by the refutation protocol. If the claim is defended, the claim predicate evaluates to true for S. The sequence S contains a surprise for the opponent of the claim because the opponent's intention was to make the predicate false. This surprise is the crystallization point for learning. The scholar playing the role of the opponent is encouraged to ask and answer the following questions: (O1) Why is my prediction wrong that I will successfully refute? (O2) What is the general pattern behind the clever construction that my partner used to defend the claim? Can I interfere with the clever construction? Can I reconstruct it from S? (O3) Can I defend the claim against a partner, successfully? (O4) Can I improve my approach to trying to refute the claim in a second attempt? (O5) Do I still believe that I can refute the claim? (O6) Did I make a mistake? Was there a second or third mistake? Do a blame assignment.

The proponent of the claim is pleased with winning but is not off the hook: (P1) Did I win by accident? Has the opponent made a mistake which made me win this time but not against a better partner? (P2) How do I repeat my success even when the opponent plays differently? (P3) Have I a systematic defense strategy? (P4) Works my systematic defense strategy in all cases?

Emotions of the proponent when she wins: joy, I found a clever construction to defend. Emotions of the opponent when he loses: disappointment, I will try to figure out your clever construction and maybe change my mind about trying to refute.

SCG offers the following approach to measure learning in a lab for a given scholar:

this does not look right for generalized SCG: Instead look for

- Transitions from contradiction to no contradiction (00):

  Refer to Table **??**. The scholar learned to avoid to be pushed into a contradiction.

  target is reached consistently

  learner gets learning points

  vv01 -> vv00 s1 learned to defend

  vv10 -> vv00 s2 learned to defend

  ff10 -> ff00 s2 learned to refute

  ff01 -> ff00 s1 learned to refute

- Transitions from one contradiction to dual contradiction:

  The scholar learned to push the contradiction to the other scholar.

  vf10 -> vf01 s2 learned to refute

  fv10 -> fv01 s2 learned to defend

  vf01 -> vf10 s1 learned to defend

  fv01 -> vf10 s1 learned to refute

OLD from old SCG:

- unsuccessful => successful

  (1) Defense attempts are unsuccessful (dau)=> defense attempts are successful (das). Scholar learned to recognize, correctly, defensible claims.

  (2) Refutation attempts are unsuccessful (rau) => refutation attempts are successful (ras). Scholar learned to recognize, correctly, refutable claims.

  Amount learned: das-dau + ras-rau

- change of mind

  (1) Claim C was unsuccessfully defended in role Proponent (udP) => claim C is successfully refuted consistently in role Opponent (sdO). Scholar initially believed claim C was true but then developed the skill to refute C.

  (2) Claim C was unsuccessfully refuted in role Opponent (urO) => claim C is successfully defended in role Proponent consistently (sdP). Scholar initially believed claim C was false but then developed the skill to defend C.

  Amount learned: sdO-udP + sdP-urO

7. INDEPENDENCE-FRIENDLY LOGIC

We want to make the point that our approach to progress-guaranteed community building for solving computational problems is not hardwired to predicate logic but also works for other logics. We illustrate here the generalization to independence-friendly logic proposed by Hintikka and Sandu.

What we rely on is that each interpreted formula ϕ with model M has an associated game $G(\phi, M, ver, fal)$. We don't use the fact that the verifier ver has a winning strategy to defend or the falsifier fal has a winning strategy . Indeed, it is perfectly fine to have indeterminate claims, i.e., claims that are neither true nor false.

Consider the following claim:

$\forall x \in X$
$\forall y \in Y(x)$
$\exists z \in Y(x)$
$quality(x, z) \geq quality(x, y)$

This claim is obviously true: choose $z = y$.

But now, lets assume that we dont know y when we choose z. The game becomes more interesting. If the verifier chooses the highest quality z, she is guaranteed to win. But the task is now more challenging.

In Independence-Friendly logic, we can express more interesting claims by expressing games with incomplete information, as we did in the example above: we did not know y when we chose z. As shown by Hintikka ([]), IF-logic also has semantical games and we can use our Contradiction-Agreement game to get the progress property for the corresponding scientific community.

8. INTRODUCTION

See: http://www.eecs.harvard.edu/ parkes/cs286r/spring02/papers/stoc01.pdf

Games, Algorithms and the Internet by Papadimitriou

We do crowdsourcing formal science using games based on logic -> Crowdsourcing, Formal Science, Games and Logic.

Crowdsourcing contests have received a lot of attention in recent years. A crowdsourcing system is a generic system that enlists a crowd of users to help solve a problem defined by the system owners [12].

This paper presents the Formal Science Crowdsourcing Platform (FSCP), a highly configurable platform for constructing crowdsourcing systems for formal scientific knowledge. FSCP represents formal scientific knowledge as a set of claims. A claim is a predicate logic formula where all nonlogical symbols (i.e. constants, functions and predicates) are interpreted in a particular domain. Logical connectives are interpreted using the Game Theoretic Semantics (GTS) of Hintikka to yield two-person, zero-sum games, called refutation games (a.k.a semantical games). The two players are called the verifier and the falsifier. The existence of a winning strategy for the verifier means that the formula is true, the existence of a winning strategy for the falsifier means that the game is false.

In FSCP, owners specialize the platform by creating labs. A lab is a crowdsourcing system that focuses a crowd of scholars on examining a family of claims to separate the true claims from false ones. Claim families are constructed by partially interpreting nonlogical symbols of a predicate logic formulas in a particular domain. Individual claims are obtained by completing the interpretation in the same domain. Scholars contribute to the lab by participating in refutation games that are syntactically derived from claims. By doing so, scholars provide evidence to the truth likelihood of individual claims in the lab. Furthermore, playing those games helps building the knowledge and intuition of individual scholars regarding the critical constructions of examples and counter examples embedded in the current winning strategies. The FSCP evaluates the performance of individual scholars as well as the truth likelihood of individual claims based on the history of all refutation games played in a particular lab.

8.1 FSCP Applications

FSCP has several **applications**, including:

1. problem solving and research in formal science. Funding agencies, such as NSF, define, in collaboration with interested researchers, labs that define the problem to be solved. Through playing the game, NSF builds a knowledge base of refutable claims and refutation attempts. Furthermore, the self-evaluating nature of FSCP will fairly evaluate the contributions of scholars and the collaborative nature will lead to productive team work. Newcomers can contribute by participating in a long-running lab (dozens of years).

2. teaching (traditional, online and massively open online) courses in STEM areas. To teach a particular problem solving skill, we design a lab for the problem. Playing FSCP challenge the students' self-image about their ability to solve the lab's problem. Thus, encouraging students to acquire the desired problem solving skill. The self-evaluating nature of FSCP helps lifting much of the evaluation from the teacher and allows stronger students to give precisely targeted feedback to weaker students.

3. software development for computational problems. A computational task is defined by a lab where the role of a scholar is played by an avatar (software). Competitions are held, and the winning avatars will contain the best (within this group of competing avatars) algorithms for the computational task.

8.2 Organization

This paper is organized as follows: In section 9 we present FSCP while in section **??** we present a novel approach for evaluating scholars and an approach for computing the truth likelihood of claims. In section 10, we present our experience with SCG, a very close predecessor of FSCP. In section 11, we present some of the related work. Section 12 concludes the paper.

9. THE FSCP PLATFORM

A lab in FSCP consists of a claim family and a number of scholars. We begin by describing how claim families are specified in FSCP. Then we describe how refutation games and substantiation games are derived from claim families. Then we describe how scholars interact with the FSCP. Finally, we describe few approaches to derive scholar interactions in a lab.

9.1 Claim Families

A claim family consists of a logical formula and a model that provides an interpretation of all predicates mentioned in the formula. A claim consists of an assignment of values from the model to all free variables in the formula. Figure 1 shows the ClaimFamily and Claim structures.

```
final class ClaimFamily{
  final Formula f;
  final Model m;

  final class Claim{
    final Assignment g;
    ...
  }
}
```

Figure 1: ClaimFamily Structure

9.1.1 Formulas

A Formula is either a simple Predicate, a Compound formula, a Negated formula, or a Quantified formula. A Compound formula consists of two subformulas, left and right and a Connective which is either an And or an Or connective. A Quantified formula consists of a Quantification and a subformula. A Quantification consists of a Quantifier, two identifiers representing the quantified variable name and type, and an optional Predicate further restricting the values the quantified variable can take. A Quantifier can be either a ForAll, an Exists, or Free which we use to declare free variables in a formula. Figure 2 shows the grammar for a formula expressed using the class dictionary notation [8].

```
Formula = Predicate | Compound | Negated |
    Quantified.
Predicate = <name> ident "(" <args>
    CommaList(ident) ")".
Compound = "(" <left> Formula <connective>
    Connective <right> Formula ")".
Connective = And | Or.
And = "and".
Or = "or".
Negated = "(" "not" <formula> Formula ")".

Quantified = <quantification> Quantification
     <formula> Formula.
Quantification = "(" <quantifier> Quantifier
     <var> ident "in" <type> ident <
    optionalQuantificationPredicate> Option(
    QuantificationPredicate) ")".
QuantificationPredicate = "where" <pred>
    Predicate.
Quantifier = ForAll | Exists | Free.
ForAll = "forall".
Exists = "exists".
Free = "free".
```

Figure 2: Formula Language

9.1.2 Models

Models are used to interpret the types and predicates in a given formula. Figure 3 shows the Model interface. It has three methods.

wellFormedTypeName checks whether a given type name is supported by the model. wellFormed checks whether a given value is a well formed value of a given type in the model. executePredicate executes a predicate provided by the model.

```
interface Model {
  boolean executePredicate(Assignment g,
      Predicate pred);
  boolean wellFormed(String value, String
      type);
  boolean wellFormedTypeName(String type);
}
```

Figure 3: Model Structure

An example of a model is the SaddlePointModel shown in Figure 4. SaddlePointModel provides one type z1 which is a floating point number between 0 and 1 inclusive. wellFormedTypeName returns true only for "z1". wellFormed returns true for well formed values of type z1. The code for executePrediate supports a single predicate p(z1 x,z1 y,z1 q). This model can be used to interpret for the formula (free q in z1) (forall x in z1) (exists y in z1) p(x,y,q).

```
class SaddlePointModel implements Model{
  public boolean wellFormedTypeName(String
      type) {
    return type.equals("z1");
  }
  public boolean wellFormed(String value,
      String type) {
    try{
      float v = Float.parseFloat(value);
      return v>=0 && v<=1;
    }catch(Exception e){
      return false;
    }
  }
  boolean executePredicate(Assignment g,
      Predicate pred) {
    if(pred.getName().getName().equals("p"))
        {

      float x = ...
      float y = ...
      float q = ...

      return (x*y + (1-x)*(1-y*y)) >= q;

    }else{
      throw new RuntimeException("Unknown␣
          predicate␣"+ pred.toString());
    }
  }
}
```

Figure 4: Sample Model

9.2 Scholars

The Scholar interface describes the inputs that FSCP collects from the crowd. The method decide is used to collect a decision from a scholar regarding whether (s)he wants to verify or falsify the given formula under the given model and assignment. Typically, a scholar would want to be the verifier of claims (s)he believes are true and be the falsifier of claims (s)he believes false. The method choose is used to collect an object from the given model for the quantification variable. Finally, the method propose is used to collect an assignment for free variables in the given formula other than the excluded assignments. It is possible to implement the Scholar interface such that it forwards requests to human scholars via email or a web interface for example. It is also possible to provide a self sufficient implementation that does not rely on human scholars. We call such Scholar implementations, avatars.

```
public interface Scholar {
  public enum Role {
    VERIFIER,
    FALSIFIER
  }
  String getName();
  Role decide(Formula f, Model m, Assignment
      g);
  String choose(Quantified f, Model m,
      Assignment g);
  Assignment propose(Formula f, Model m,
      Collection<Assignment> excluded);
}
```

Figure 5: Scholar Interface

9.3 Scholar Interaction

In FSCP, the interaction between scholars is centered around claims. Two scholars can interact by participating in a substantiation game. Substantiation games build on refutation games which we start explaining before we move to substantiation a refutation game or a substantiation game.

9.3.1 Refutation Games

Two scholars taking opposite positions on a specific claim c can participate in a refutation game denoted as c.RG(verifier, falsifier) where verifier is the scholar trying to support c and falsifier is the scholar disputing c.

Given a claim c and two scholars, a verifier ver and a falsifier fal. Let ϕ be the formula and M be the model of c's enclosing ClaimFamily. Let g be c's assignment to the free variables in ϕ. We define the refutation game of ver and fal centered around c c.RG(ver, fal) to be $G(\phi, M, g, ver, fal)$ which is a two-player, zero-sum game defined as follows:

1. If $\phi = R(t_1, ..., t_n)$ and $M, g \models R(t_1, ..., t_n)$, ver wins; otherwise fal wins.
2. If $\phi = !\psi$, the rest of the game is as in $G(\psi, M, g, fal, ver)$.
3. If $\phi = (\psi \wedge \chi)$, fal chooses $\theta \in \{\psi, \chi\}$ and the rest of the game is as in $G(\theta, M, g, ver, fal)$.
4. If $\phi = (\psi \vee \chi)$, ver chooses $\theta \in \{\psi, \chi\}$ and the rest of the game is as in $G(\theta, M, g, ver, fal)$.
5. If $\phi = (\forall x : p(x))\psi$, fal chooses an element a from M such that $p(a)$ holds, and the rest of the game is as in $G(\psi, M, g[x/a], ver, fal)$. If fal fails to do so, it loses.

s1	s2	ref game	tested
v	v	c.RG(s1, s2)	s1
v	v	c.RG(s2, s1)	s2
f	f	c.RG(s1, s2)	s2
f	f	c.RG(s2, s1)	s1

Table 1: Scholar Under Test

6. If $\phi = (\exists x : p(x))\psi$, ver chooses an element a from M such that $p(a)$ holds, and the rest of the game is as in $G(\psi, M, g[x/a], ver, fal)$. If ver fails to do so, it loses.

The definition of G is adopted from the Game Theoretic Semantics (GTS) of Hintikka [22], [37]. We slightly modified Hintikka's original definition to handle the quantification predicate in our language. The result of a refutation game consists of a record $\mathrm{RGHistory}$ of the two scholars verifier and falsifier, the winner, the assignment g, and a $\mathrm{timestamp}$.

9.3.2 Substantiation Games

FSCP further extend the potential for interaction between scholars by allowing scholars to participate in test games even if they are taking the same positions on a specific claim c. Two scholars $\mathrm{s1}$ and $\mathrm{s2}$ taking two, not necessarily contradictory, positions $\mathrm{r1}$ and $\mathrm{r2}$ on claim c can participate in a substantiation game $\mathrm{c.SG(s1, r1, s2, r2)}$. If the two scholars hold contradictory positions on c, the substantiation game reduces to a refutation game. Otherwise, the substantiation game reduces to two refutation games $\mathrm{c.RG(s1, s2)}$ and $\mathrm{c.RG(s2, s1)}$ in which the two scholars teach each other. Given the two positions and the game, Table 1 can be used to identify the scholar being tested. It is important to identify the scholar under test for scholar evaluation purposes. The result of a substantiation game is a list of either one $\mathrm{RGHistory}$ record or two $\mathrm{TestHistory}$ records. A $\mathrm{TestHistory}$ extends $\mathrm{RGHistory}$ records with a $\mathrm{underTest}$ field.

9.4 Labs

An FSCP lab is a crowdsourcing system that consists of a $\mathrm{ClaimFamily}$ and a number of $\mathrm{Scholars}$. Furthermore, based on its goal, a lab also provides:

1. system wide interaction mechanisms for scholars,
2. an evaluation mechanism for its scholars,
3. a mechanism for combining scholars' contributions.

We discuss system wide interaction mechanisms below and discuss scholar evaluation and combination of scholar contributions in Section **??**.

It is possible to build several system wide interaction mechanisms on top of substantiation games and the $\mathrm{Scholar}$ interface. We give here three examples:

9.4.1 Battleship

Scholars independently propose claims to the system. When two scholars propose the same claim, the system collects their position and then engages them in a substantiation game.

9.4.2 Guided Search

The system chooses a claim and two scholars, then it collects their positions on that claim and engages them in a substantiation game. The system repeats until it reaches its goal. For example, suppose that we are building a crowdsourcing system to find the critical point of some free variable (from an ordered domain) in a formula. This is the value such that all claims above it are, for example, false. The system can effectively perform a binary search on the domain of that free variable. At each step in the binary search, the system creates a claim and chooses two scholars and engage them in a substantiation game.

9.4.3 Scientific Community Game

The Scientific Community Game (SCG) is a precursor to FSCP. The focus of SCG was educational. In SCG, scholars play a soccer-like tournament of binary matches. A match consists of an even number of rounds where scholars participate in binary games with alternating roles. SCG binary games are a precursor to substantiation games. In an SCG binary game, a scholar called the proponent proposes a claim. By doing so, the proponent is implicitly taking the verifier position on the claim it proposed. Then the other scholar, called the opponent, is asked to decide whether it agrees or disputes the claim. In either case, both scholars participated in a refutation game.

Eventually, a scholar ranking is produced as well as a trace of all refutation games. The intent was that scholars learn from the traces and the ranking is used to motivate them.

10. EXPERIENCE WITH THE SCG

The SCG has evolved since 2007. We have used the SCG in software development courses at both the undergraduate and graduate level and in several algorithm courses. Detailed information about those courses is available from the second author's teaching page.

10.1 Software Development

The most successful graduate classes were the ones that developed and maintained the software for SCG Court [1] as well as several labs and their avatars to test SCG Court. Developing labs for avatars has the flavor of defining a virtual world for artificial creatures. At the same time, the students got detailed knowledge of some problem domain and how to solve it. A fun lab was the Highest Safe Rung lab from [20] where the best avatars needed to solve a constrained search problem using a modified Pascal triangle.

10.2 Algorithms

The most successful course (using [20] as textbook) was in Spring 2012 where the interaction through the SCG encouraged the students to solve difficult problems. Almost all homework problems were defined through labs and the students posted both their exploratory and reformatory actions on piazza.com. We used a multi player version of the SCG binary game which created a bit of an information overload. Sticking to binary games would have been better but requires splitting the students into pairs. The informal use of the SCG through Piazza (piazza.com) proved successful. All actions were expressed in JASON which allowed the students to use a wide variety of programming languages to implement their algorithms.

The students collaboratively solved several problems such as the problem of finding the worst-case inputs for the Gale-Shapely algorithm (see the section Example above).

We do not believe that, without the SCG, the students would have created the same impressive results. The SCG effectively focuses the scientific discourse on the problem to be solved.

The SCG proved to be adaptive to the skills of the students. A few good students in a class become effective teachers for the rest thanks to the SCG mechanism.

11. RELATED WORK

11.1 Crowd Sourcing and Human Computation

11.1.1 Dealing with Unreliable Workers

Most crowdsourcing systems must devise schemes to increase confidence in the worker's solutions to tasks, typically by assigning each task multiple times [17]. Larger et AL. present a general model for crowdsourcing tasks. In FSCP, because workers need to justify their answers in a game against another worker, unreliable workers will run into many contradictions and get a low rating. This means that their votes will minimally affect the final result, the knowledge base of true claims.

[9] is related to FSCP scholar ranking. The algorithm is an extended Bradley-Terry model called Crowd-BTU. The paper focuses on finding the quality of annotators in a crowdsourced setting. They study the exploration-exploitation tradeoff which is also relevant to FSCP for labeling claims as true or false.

The "Evaluating the Crowd with Confidence" paper [16] has a title that seems very applicable to FSCP. However, they use a model which is too simple for FSCP. In particular, in FSCP the errors depend on task difficulty, and worker errors are not independent of each other because they play a game.

11.1.2 Rating Systems

We use a rating system for games with wins, losses and draws. This subject has been studied for a long time and there are many applications of rating systems. For example, in chess and other sports, the Elo rating system is used. A good survey and critique of rating systems is given in [6]. Rating systems are a controversial subject and there are many algorithms that can be used. TopCoder [35] uses an Algorithm Competition Rating System to rank the coders.

11.1.3 Combining Worker's Contributions

In FSCP, we use two approaches to combine scholar contributions: (1) During the refutation games, the scholars give each other feedback by trying to drive each other into a contradiction. This is a collaboration which leads potentially to new ideas and knowledge fusion. (2) In FSCP, scholars vote on the truth or falseness of claims when deciding to verify or falsify claims. Furthermore, it is not enough for scholar to just vote but also it is important that they justify their votes through their actions in refutation games. We combine the votes with justifications into an overall vote for whether a claim is true. Related work is [9] and [17] which was already discussed above.

11.1.4 Competitions

There are several websites that organize competitions. What is common to many of those competitions? We believe that the FSCP provides a foundation to websites such as TopCoder.com or kaggle.com.

The FSCP makes a specific, but incomplete proposal of a programming interface to work with the global brain [7]. What is currently missing is a payment mechanism for scholars and an algorithm to split workers into pairs based on their background.

The FSCP is a generic version of the "Beat the Machine" approach for improving the performance of machine learning systems [5].

Scientific discovery games, such as FoldIt and EteRNA, are variants of the FSCP. [10] describes the challenges behind developing scientific discovery games. [3] argues that complex games such as FoldIt benefit from tutorials. This also applies to the FSCP, but a big part of the tutorial is reusable across scientific disciplines.

11.1.5 Crowdsourcing complex tasks

[19] describes a general-purpose framework for solving complex problems through micro-task markets. Engaging in the scientific dialogs of FSCP could be done through a micro-task market. [29] proposes a language to define crowdsourcing systems. Our lab definition approach provides a declarative description of what needs to be crowdsourced.

[21] provides an interesting analysis of several issues relevant to FSCP: how incorrect responses should affect worker reputations and how higher reputation leads to better results.

11.2 Logic and Imperfect Information Games

Logic has long promoted the view that finding a proof for a claim is the same as finding a defense strategy for a claim.

Logical Games [28], [14] have a long history going back to Socrates. The FSCP is an imperfect information game which builds on Paul Lorenzen's dialogical games [18].

11.3 Foundations of Digital Games

A functioning game should be deep, fair and interesting which requires careful and time-consuming balancing. [15] describes techniques used for balancing that complement the expensive playtesting. This research is relevant to FSCP lab design. For example, if there is an easy way to refute claims without doing the hard work, the lab is unbalanced.

11.4 Architecting Socio-Technical Ecosystems

This area has been studied by James Herbsleb and the Center on Architecting Socio-Technical Ecosystems (COASTE) at CMU http://www.coaste.org/. A socio-technical ecosystem supports straightforward integration of contributions from many participants and allows easy configuration.

The FSCP has this property and provides a specific architecture for building knowledge bases in (formal) sciences. Collaboration between scholars is achieved through the scientific discourse implied by the refutation game. The information exchanged gives hints about how to play the game better next time. An interesting question is why this indirect communication approach works.

The NSF workshop report [33] discusses socio-technical innovation through future games and virtual worlds. The FSCP is mentioned as an approach to make the scientific method in the spirit of Karl Popper available to CGVW (Computer Games and Virtual Worlds).

11.5 Online Judges

An online judge is an online system to test programs in programming contests. A recent entry is [30] where private inputs are used to test the programs. Topcoder.com [35] includes an online judge capability, but where the inputs are provided by competitors. This dynamic benchmark capability is also expressible with the FSCP: The claims say that for a given program, all inputs create the correct output. A refutation is an input which creates the wrong result.

11.6 Educational Games

The FSCP can be used as an educational game. One way to create adaptivity for learning is to create an avatar that gradually poses harder claims and makes the scientific discourse more challenging. Another way is to pair the learner with another learner who is stronger. [2] uses concept maps to guide the learning. Concept maps are important during lab design: they describe the concepts that need to be mastered by the students for succeeding in the game.

11.7 Formal Sciences and Karl Popper

James Franklin points out in [13] that there are also experiments in the formal sciences. One of them is the 'numerical experiment' which is used when the mathematical model is hard to solve. For example, the Riemann Hypothesis and other conjectures have resisted proof and are studied by collecting numerical evidence by computer. In the FSCP experiments are performed when the game associated with a claim is elaborated.

Karl Popper's work on falsification [31] is the father of non-deductive methods in science. The FSCP is a way of doing science on the web according to Karl Popper.

11.8 Scientific Method in CS

Peter Denning defines CS as the science of information processes and their interactions with the world [11]. The FSCP makes the scientific method easily accessible by expressing the hypotheses as claims. Robert Sedgewick in [34] stresses the importance of the scientific method in understanding program behavior. With the FSCP, we can define labs that explore the fastest practical algorithms for a specific algorithmic problem.

11.9 Games and Learning

Kevin Zollman studies the proper arrangement of communities of learners in his dissertation on network epistemology [38]. He studies the effect of social structure on the reliability of learners.

In the study of learning and games the focus has been on learning known, but hidden facts. The FSCP is about learning unknown facts, namely new constructions.

11.10 SCG

SCG [25], [23], [24] is a close predecessor of FSCP. The original motivation for the SCG came from the two papers with Ernst Specker: [26] and the follow-on paper [27].

The key difference between FSCP and SCG is that SCG was targeted at evaluation of the scholars while FSCP is targeted at crowdsourcing true claims. FSCP is cleaner: there is a simple concept of self-contradiction and there is no longer a need to have the concept of strengthening a claim explicitly.

12. CONCLUSION AND FUTURE WORK

We presented FSCP, a crowdsourcing platform for formal science. FSCP provides a simple interface to a community that uses the (Popperian) Scientific Method.

We want to extend our model so that we can make claims about claims. For example, we want to have a "macro" for a claim to be optimal. We want to leverage claim relationships across labs and work with lab reductions as a useful problem solving tool.

We see a significant potential in putting the refutation-based scientific method into the cyberinfrastructure and make it widely available. We plan to, iteratively, improve our current implementation based on user feedback.

We see an interesting opportunity to mine the game histories and make suggestions to the scholars how to improve their skills to propose and defend claims. If this approach is successful, FSCP will make contributions to computer-assisted problem solving.

13. ACKNOWLEDGMENTS

We would like to thank Magy Seif El-Nasr, Casper Harteveld, Thomas Wahl and Tugba Koc for their input to the paper.

14. REFERENCES

[1] A. Abdelmeged and K. J. Lieberherr. SCG Court: Generator of teaching/innovation labs on the web. Website, 2011. http://sourceforge.net/p/generic-scg/code-0/110/tree/GenericSCG/ .

[2] E. Andersen. Optimizing adaptivity in educational games. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 279–281, New York, NY, USA, 2012. ACM.

[3] E. Andersen, E. O'Rourke, Y.-E. Liu, R. Snider, J. Lowdermilk, D. Truong, S. Cooper, and Z. Popovic. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 59–68, New York, NY, USA, 2012. ACM.

[4] S. Arora, L. Lovász, I. Newman, Y. Rabani, Y. Rabinovich, and S. Vempala. Local versus global properties of metric spaces. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 41–50, New York, NY, USA, 2006. ACM.

[5] J. Attenberg, P. Ipeirotis, and F. Provost. Beat the machine: Challenging workers to find the unknown unknowns. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[6] J. Beasley. *The Mathematics of Games*. Dover Books on Mathematics. Dover Publications, 2006.

[7] A. Bernstein, M. Klein, and T. W. Malone. Programming the global brain. *Commun. ACM*, 55(5):41–43, May 2012.

[8] B. Chadwick. DemeterF: The functional adaptive programming library. Website, 2008. http://www.ccs.neu.edu/home/chadwick/demeterf/.

[9] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM, Rome, Italy*, 2013.

[10] S. Cooper, A. Treuille, J. Barbero, A. Leaver-Fay, K. Tuite, F. Khatib, A. C. Snyder, M. Beenen, D. Salesin, D. Baker, and Z. 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.

[11] P. J. Denning. Is computer science science? *Commun. ACM*, 48(4):27–31, Apr. 2005.

[12] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011.

[13] J. Franklin. The formal sciences discover the philosophers' stone. *Studies in History and Philosophy of Science*, 25(4):513–533, 1994.

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

[15] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic. Evaluating competitive game balance with restricted play, 2012.

[16] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. Technical report, Stanford University.

[17] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 1953–1961, 2011.

[18] L. Keiff. Dialogical logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2011 edition, 2011.

[19] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut.

Crowdforge: crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM.

[20] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[21] M. Kosinski, Y. Bachrach, G. Kasneci, J. V. Gael, and T. Graepel. Crowd iq: measuring the intelligence of crowdsourcing platforms. In *WebSci'12*, pages 151–160, 2012.

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

[23] K. Lieberherr. The Scientific Community Game. Website, 2009. http://www.ccs.neu.edu/home/lieber/evergreen/specker/scg-home.html.

[24] K. J. Lieberherr and A. Abdelmeged. The Scientific Community Game. In *CCIS Technical Report NU-CCIS-2012-19*, October 2012. http://www.ccs.neu.edu/home/lieber/papers/SCG-definition/SCG-definition-NU-CCIS-2012.pdf.

[25] K. J. Lieberherr, A. Abdelmeged, and B. 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 .

[26] K. J. Lieberherr and E. Specker. Complexity of Partial Satisfaction. *Journal of the ACM*, 28(2):411–421, 1981.

[27] K. J. Lieberherr and E. Specker. Complexity of Partial Satisfaction II. *Elemente der Mathematik*, 67(3):134–150, 2012. http://www.ccs.neu.edu/home/lieber/p-optimal/partial-sat-II/Partial-SAT2.pdf.

[28] M. Marion. Why Play Logical Games. Website, 2009. http://www.philomath.uqam.ca/doc/LogicalGames.pdf.

[29] P. Minder and A. Bernstein. Crowdlang - first steps towards programmable human computers for general computation. In *Proceedings of the 3rd Human Computation Workshop*, AAAI Workshops, pages 103–108. AAAI Press, 2011.

[30] J. Petit, O. Giménez, and S. 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.

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

[32] A. J. Quinn and B. B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.

[33] W. Scacchi. The Future of Research in Computer Games and Virtual Worlds: Workshop Report. Technical Report UCI-ISR-12-8, 2012. http://www.isr.uci.edu/tech_reports/UCI-ISR-12-8.pdf.

[34] R. Sedgewick. The Role of the Scientific Method in Programming. Website, 2010. http://www.cs.princeton.edu/~rs/talks/ScienceCS.pdf.

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

[36] L. Trevisan. On local versus global satisfiability. *SIAM J. Discret. Math.*, 17(4):541–547, 2004.

[37] T. Tulenheimo. Independence friendly logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2009 edition, 2009.

[38] K. J. S. Zollman. The communication structure of epistemic communities. *Philosophy of Science*, 74(5):574–587, 2007.