# Specker Derivative Game: Requirements & Design From Generic to CNF/CSP Versions

Karl Lieberherr

College of Computer & Information Science
Northeastern University, 360 Huntington Avenue
Boston, Massachusetts 02115 USA.
{lieber}@ccs.neu.edu

Date: June 6, 2009, Fall 2009 version

## 1 Introduction

The *Specker Derivative Game* (SDG) is a game of interacting robots that seek to maximize their energy by engaging in energy enhancing interactions. A potential interaction is described by a description of raw materials that may be delivered and a price. When an interaction is entered, the price is paid (energy deducted), raw materials are delivered and finished and the quality achieved for the finished product is paid (energy added).

SDG is interesting because it provides a very simple model of the behavior of living organisms who choose actions that give them positive energy in the long run. Those organisms need to look ahead of what the consequences of their actions might be. When they perform an action in the world, for example, to offer or buy a service, they need to judge whether this action might drain their energy.

We use the following terms as synonyms: service / derivative; life energy / money; price / life energy subtracted for engaging in an activity.

This document describes the Classic SDG game. The description of the Secret SDG game is here: http://www.ccs.neu.edu/home/lieber/evergreen/specker/sdg-home.html.

## 2 Definition

We parameterize the *Specker Derivative Game* (SDG) over the derivatives, raw materials etc. to be used. Then we discuss the generalized satisfiability instance (CSP). This supports better separation of concerns at the requirements level. We can then formulate the important game rules without knowing the details of the derivatives.

### 2.1 Parameterization

The SDG game is parameterized by a tuple $C = (G, Pred, D, R(d, d \in D), O(r, r \in R(d), d \in D), F, Q)$, where $G$ is a set of raw materials, *Pred* is a set of *predicates*, each selecting a subset of $G$, $D$ (*derivatives*) is a set of pairs $d = (t, p)$, where $t \in Pred$ is the predicate $pred(d)$ and $0 \leq p \leq 1$ is the price $price(d)$. $R(d)$ (*raw materials* for derivative $d$) is a set with each element $r \in R(d)$ satisfying predicate $pred(d)$, the predicate of derivative $d$. $O(d)$ is the set of feasible solutions for elements in $R(d)$; we denote a feasible solution for raw material $r$ as $o(r) \in O(d)$. $F$ (*finished products*) is a set of pairs $(r, o(r))$, with $r \in R$ and $o(r) \in O$. $Q$ (*quality*) is a function that maps a finished product $(r, o(r))$ to $[0, 1]$. $Pred$ is expressed in a suitable chosen predicate language described by a grammar and its semantics.

The game is about buying and selling derivatives. When a derivative $d = (t, p)$ is offered, only its predicate $t$ and price $p$ are known. The creator and buyer of a derivative need to do a *min-max* analysis for the predicate $t$, which makes the game interesting. By min-max analysis we mean that it must be known what the worst-case raw material is for a given predicate. The seller must know this to price the derivative properly and the buyer must know this to decide whether the price is right. The buyer of a derivative will be paid the *quality* of the finished product achieved for the raw material produced by the seller, after the derivative was bought.

$SDG(C)$ is a tuple $(P, account, store, config)$, where $P$ is an ordered set of players, *account* is a function that assigns a positive real number to each player. *store* is a function that assigns a pair $(forSale, bought)$ to each player, where *forSale* is a set of derivatives for sale and *bought* is a set of tuples $(d, seller, r, f)$, where $d$ is a derivative, $seller \in P$, $r \in R \cup \{absent\}$ and $f \in F \cup \{absent\}$. $config$ is a tuple $(init, maxTurns, timeslot, mindec)$, which configures the game. This configuration tuple will be later expanded with specifics for the combinatorial structure[1] to be used in the game. *init* is a positive real number, giving the initial amount of the account of each player in $P$. *maxTurns* is the maximum number of turns the game will be played. *timeslot* is the amount of time given to each player. *mindec* is a small real number which specifies the minimum decrement when the price of a derivative is lowered.

### 2.2 Two Player Game Rules: explained with Chess analogy

Note: The two player game rules are listed here to explain the game using a chess analogy, but the game that is implemented uses a different protocol. The game that is implemented also works with only two players.

Without loss of generality, we limit the game to two players, called White and Black, as in chess. White starts the game. The board is a set $G$ of raw materials and a predicate language to express predicates selecting subsets of raw materials.

A mega move White/Black consists of: White's first move is to offer a set of derivatives for sale. Black's next move is to decide which subset of the derivatives on sale to buy. White's next move is to deliver raw material for the bought derivatives. Black's next move is to finish the raw materials that have been delivered. If Black does not buy any derivatives, there is no raw material delivery and no finished products delivered but instead Black reoffers all derivatives for sale by White at a lower price.

After a mega move White/Black comes a mega move Black/White with the roles reversed. An even number of mega moves will be played, determined by the configuration object. The winner of the game is the robot with the most life energy. If both have the same life energy, it is a tie.

The game rules of the next subsection apply, adapted to the the two player situation.

### 2.3 Game Rules

The $SDG(C)$ game has an asynchronous and a synchronous version. Here we define the synchronous version where players operate sequentially; in the asynchronous version, players can buy and sell at any time.

The rules of the synchronous $SDG(C)$ game are:

1. *Main Objective.* The winner of the game is the player with the most money in the player's account at the end of the game. The account value ranks the players. Players with the same account value have the same rank.
2. *Uniform Turns.* Only one player is playing at a given time. When a player is done, the next player is the next element in the ordered set $P$. A player may indicate that s/he is done in a variety of ways, for example by creating a file. In other words, the players take turns in a uniform sequence.
3. *Buy or Re-offer.* To make derivatives more attractive, on each turn, a player must buy at least one derivative offered for sale by other players or re-offer all derivatives for sale by all other players at a lower price. When a derivative is bought the seller is paid by the buyer the price of the derivative.
4. *Offer new derivative.* On each turn, a player must offer a derivative whose predicate does not exist yet in the store, or whose price is lower than the price of all other derivatives of the same predicate in the store.
5. *Timely delivery.* A player must deliver raw materials for derivatives bought from them in the previous round. The predicate of the raw material must match predicate of the derivative.
6. *Obligation to the finished product.* The owner of a derivative is obliged to pay for a finished product based on the quality achieved as soon as the finished product is delivered.
7. *Price lowering.* When lowering the price of a derivative, it must be lowered at least by *mindec*.
8. *Bought once.* A derivative may only be bought once from the store.
9. *Positive account.* Players must maintain a positive account.

[1] CNF or CSP for example.

10. *Time limit.* Players must finish within *timeslot*.
11. *Consequences.* Players that don't comply with the rules are removed from the game. If a player is removed from the game, its derivatives are removed from the store and its bought, but unfinished derivatives are refunded to the buyer.
12. *Completion.* After *maxTurns* rounds, nothing can be bought but raw materials are still delivered and finished until all outstanding obligations are met.

### 2.4 Archiving Concern

We want to be able to archive games for further analysis. We use the following 4 archiving transactions.

1. When a player $p$ offers a derivative $d$, we archive $create(p, d)$.
2. When a player $p$ buys a derivative $d$ from $seller$, we archive $buy(seller, p, d)$.
3. When a player $p$ delivers raw material $r$ for derivative $d$, we archive $deliverR(p, d, r)$.
4. When a player $p$ delivers the finished product $f = (r, o(r))$ for raw material $r$ and derivative $d$, we archive $deliverF(p, d, f)$.

Given a sequence of archived transactions, we can check whether the players followed the rules of the game. For example, a player can only finish raw material for a derivative that was bought previously.

### 2.5 Security Concern

It is important to have integrity of the competitions. We rely on the administrator to enforce the rules of the artificial world. If it fails to provide a safe and equal playground for all robots, the intelligence of the robots is of questionable value and we start mistrusting the game and feel cheated.

A robot, to gain life energy quickly, may try the following:

- Attack the administrator.
  There may be vulnerabilities in the administrator that can be used to gain life energy. An example would be to directly manipulate the account values in one's favor or to manipulate the store.
  These attacks are undesirable unless one wants to strengthen the administrator.
- Attack another robot.
  There may be vulnerabilities in a robot that can be exploited to gain life energy. We distinguish between intelligence and mechanism attacks. An example of an intelligence attack is to trade derivatives for which the attacked robot makes poor decisions. An example of a mechanism attack is to modify a "personal" file of the attacked robot.
  - Intelligence
    This kind of attack is desirable because it improves the quality of the robots. The attacking robot will gain life energy for alerting the attacked robot of its vulnerabilities. But in the long run it will improve the quality of all robots.
  - Mechanism
    In a course where software security is taught, this is also a dersirable kind of attack because it improves the security of the robots.

A good approach to security is to use different security policies for different competitions. And to allow robot submissions for different policies. Although the administrator in principle should be perfect and not contain vulnerabilities, there might be some hidden threats.

There are three kinds of attack modes: Administrator Attack Mode, Robot Attack Mode (Intelligence) and Robot Attack Mode (Mechanism). Each of the attack modes may be on or off is described by a Boolean triple.

Security policy: When an attack mode is off, there is a gentleman's agreement that no attack of that kind takes place. If it does, there is a penalty for the offending robot, if caught. The history files of all competitions are public which allows other robots to look for violators. The resolution of an issue might require an arbitration hearing where the software developers of a robot act as its lawyers that defend the actions of the robot. A student acts as chair person of the arbitration panel and all class members that are

not acting as lawyers act as members of the arbitration panel. In this sense, in class arbitration hearings are used as a security mechanism to resolve security disputes.

(0,0,0) is a very friendly robot that tries to win through uniform intelligence. The robot is not allowed to target individual robots.

(0,1,0) is a friendly debugging robot that exploits intelligence weaknesses of another robot. For example, the robot might always use the same derivatives against the attacked robot because the attacked robot makes bad decisions regarding those derivatives.

(0,0,1) is a friendly debugging robot that exploits mechanism weaknesses of another robot. For example, the attacking robot might modify a temporary file that the other robot left unprotected.

(1,0,0) is an vicious robot that tries to modify the artificial world in its favor.

(1,1,1) is a fierce robot.

Competitions often take place under the (0,1,0) policy. It makes sense to have (0,1,1) policies to make the robot developers aware of security problems in their robot. Competitions using the (1,*,*) security policy don't make sense if the objective is to improve the robots and not the administrator.

However, teams may submit two robots, one for the (0,1,0) or (0,1,1) or (0,0,1) policy and one for the (1,*,*) policy, illustrating an exploit in the administrator. Only the first will be used in the next competition while the second, if revealing a bug in the administrator, will be awarded $ 1 million in life energy.

The security concern in the game can only be partially addressed by the administrator through technical solutions. But a correct and secure administrator are an important part of the solution.

The game has the property that after each round, all information becomes public by exposing it in the history file. This reduces the need for cryptographic techniques.

## 3 Specialization

### 3.1 For CNF

This specialization initiates learning about propositional calculus and basic combinatorics. Algorithms for MAX-SAT are important to play the game well.

$G$ is the set of all conjunctive normal forms that use only two clause types. $Pred = \{r1, r2\}$ where $r1$ and $r2$ are *clause types*. In other words, we express the predicates with clause types. A clause type is a pair $(l, p)$, where $p$ is a set of positive literals with $l$ elements. $R(d)$ is the set of weighted CNF-formulas satisfying the predicate of $d$, where each clause has a positive integer *weight*. $O(r)$ is the set of all assignments for a CNF-formula = raw material $r$. $F$ is a pair $(r, J)$, where $J \in O$ is an assignment for the CNF-formula $r$. $Q(r, J)$ is the weighted fraction of satisfied clauses in CNF-formula $r$ under assignment $J$.

To make the sets finite we add the following configuration tuple:

$$(maxVars, maxClauses, maxWeight, maxClauseLength)$$

### 3.2 For CSP

This specialization initiates learning about Boolean constraint satisfaction. Algorithms for MAX-CSP are important to play the game well.

For this formulation, $G$ is the set of all Boolean weighted CSP-formulae that use only Boolean relations of arity 3. $Pred$ is the set of all subsets of the set of boolean relations of at most arity 3. There are 256 relations of arity 3. There are $2^{256}$ different predicates in $Pred$. $R(d)$ is the set of CSP($Pred$)-formulas that satisfy the predicate of $d$. A derivative is a set of relations in $Pred$ and a price. $O(r)$ is a Boolean assignment $J$ for a CSP($Pred$)-formula $r$. $F$ is a pair $(r, J)$, where $J$ is an assignment to the variables of $r$. $Q(r, J)$ is the weighted fraction of satisfied constraints in $r$ under assignment $J$.

To make the sets finite we add the following configuration tuple:

$$(maxVars, maxConstraints, maxWeight)$$

### 3.3 Simpler Versions for Classic SDG CSP

We equate the Classic SDG CSP game to baseball which has two simpler versions: T Ball and Softball. Softball itself has two versions: slow pitch and fast pitch. Before our robots can play the full version of the Classic SDG CSP game, they need to learn simpler versions to ensure a smooth learning curve both for the robots and their creators.

**Classic SDG CSP T Ball** Playing T Ball is much easier than playing the full game. Classic SDG CSP T Ball has the same rules as Classic SDG, but for all derivatives only one Boolean relation is allowed in the predicate.

Skills needed include: deriving expected fraction of satisfied constraints for a coin with bias b. Find the maximum bias. Generate all combinations of k elements out of n.

**Classic SDG CSP Slow Pitch Softball** Classic SDG CSP Slow Pitch Softball has the same rules as Classic SDG, but for all derivatives the relations defining the predicate must form an implication tree. For example with the standard relational encoding used by

`http://www.ccs.neu.edu/home/lieber/courses/csu670/sp09/source/IR-2.0/doc/`

2 implies 6 implies 22 and 2 implies 10 is an implication tree with root 2. (Or for Classic SDG SAT: (1,1) implies (2,2) implies (3,3) is an implication tree.)

Slow Pitch Softball can be reduced to T Ball by recognizing that all but one relation in a fastpitch softball derivative are noise. Only the root of the tree is important.

An alternative definition of Slow Pitch Softball is that all but one relation can be eliminated using implications.

**Classic SDG CSP Fast Pitch Softball** This is the full version of Classic SDG CSP. We have multiple relations and we need to identify the important ones for determining the break-even price. There may be several implication relationships. For example, you might have a derivative d = (2,6,22,1) which consists of two implication chains: 2 implies 6 implies 22 and the singleton chain 1. An interesting question is whether to price (2,6,22,1) it is sufficient to price (2,1).

We define the simplified form: Level k Independent of Classic SDG CSP Fast Pitch Softball. In Level k Independent we have k relations so that no relation implies any of the others. An example of Level 2 Independent is (using the SAT notation) ((1,1) (2,0)) which has a break-even price of (sqrt(5)-1)/2 = 0.618 ...

Classic SDG CSP Fast Pitch Softball can be reduced to Classic SDG CSP Fast Pitch Softball Level k Independent (for some k) by finding the implications and eliminating the stronger relations.

Note that Classic SDG CSP Fast Pitch Softball Level 1 Independent is the same as T Ball.

We define the simplified form: Level k Reduced of Classic SDG CSP Fast Pitch Softball. We have any number of relations that can be reduced to Level k Independent.

An example: The Classic SDG CSP Fast Pitch Softball derivative (using SAT notation) ((1,1) (2,2) (3,3) (2,0) (3,0)) is Level 2 Reduced. It can be reduced to ((1,1) (2,0)) of Classic SDG CSP Fast Pitch Softball Level 2 Independent.

Therefore, ((1,1) (2,2) (3,3) (2,0) (3,0)) also has a break-even price of (sqrt(5)-1)/2 = 0.618 ...

Note that Classic SDG CSP Slow Pitch Softball is a special case of Classic SDG CSP Fast Pitch Softball Level 1 Reduced.

**SDG Base Ball** This game uses not only classic derivatives but also secret derivatives. It covers a much wider space than Classic SDG CSP because we can use in principle any NP-hard combinatorial maximization problem. One challenge is to define a generic language to define combinatorial maximization problems.

SDG CSP Base Ball is the special case where we stay with CSP but allow secret CSP derivatives. This extension is relatively easy to add to Classic SDG CSP Fast Pitch Softball.

### 3.4 Version for Optimization

This generalizes the game to any standard optimization problem: an optimization problem is the problem of finding the best solution from all feasible solutions.

Note: This is very similar to the parameterization described in 2.1, except that $g$ is added.

More formally, an optimization problem A is a quadruple $(I, f, m, g)$, where

- $I$ is a set of instances. These are our raw materials and in our world we will define them by a class dictionary and a semantic checker. An example would be a class dictionary for weighted CSP instances. The semantic checker would check, for example, that no variable appears more than once in a constraint.
- $f$ is a function that maps an instance $x$ of $I$ to a set of feasible solutions. The set of feasible solutions for a weighted CSP instance with variable set $V$ is the set $literals(V)$.
- $m(x, y)$ is a function that maps an instance $x$ and a feasible solution $y$ of $x$ to a real number. For a weighted CSP instance $m(x, y)$ is the weighted satisfaction ratio of assignment $y$ for instance $x$.
- $g$ is the goal function and is either min or max. For Max CSP, $g$ is max.

A predicate on the raw materials is used to define a derivative. This is best done through a class dictionary and a semantic checker. For a weighted CSP raw material, the predicate might be that the instance contains only a given set of relations.

This abstract way of defining raw materials and predicates makes it much harder for robots to make their buying decisions. Making a buying decision involves analyzing the class dictionary for $I$ and the code for $f, m$ and $g$. It is good to specify the code in a "normalized" way, for example, using DemeterF.

To make the problem tractable, the optimization problem will be a constant for each competition. But different competitions will use different optimization problems. For example, we might change $g$ from max to min.

When the optimization problem is a constant, it is the software developers who need to analyze the definition of the optimization problem and not the robots. The robots only need to be aware of a small number of optimization problems that are competition candidates.

**Pseudo Boolean CSP** We give another example for Optimization: We consider a version of CSP where the constraints are expressed as pseudo Boolean constraints.

The class dictionary is:

```
PseudoBooleanCSP =
  <variables> List(Variable) <constraints> List(PBConstraint).
...
Literal = <kind> Kind <v> Variable.
Kind = Pos | Neg.
Pos = .
Neg = "!".
Assignment = <literals> List(Literal).
```

We need a semantic checker to define the legal instances. The variables in part variables must be exactly the list of variables appearing in constraints.

All List(Literal) of the same length as variables is the set feasible solutions.

...

## 4 Web version

The administrator is the web robot and the robots turn into web applications that respond to the administrator. When it is time for a competition, the administrator broadcasts the constant: the optimization problem for which a competition will be held. The web applications that are ready for the competition with the given optimization problem register with the administrator and the fun begins.

There is the issue that the web applications might not "understand" the optimization problem because there are many different ways to define the same optimization problem. We assume that there is a coordination between the administrator and web applications so that the web applications can check for equality between the optimization objects.

We need a class dictionary for class dictionaries (CDCD-SDG) and a class dictionary (DemeterF-SDG) for defining $f, m$, and $g$. Is CDCD-SDG equal to the cd for cds of DemeterF?

## 5 Reflection on the course

We can recognize elements of the game in real life. You have selected this course based on a a set of features: course description, college requirements, reputation, etc. Based on those features you have selected to take my course. The features correspond to the predicate of a derivative which you buy only by knowing the predicate. Now I deliver raw materials (subprojects) within the constraints of the course features we agreed upon. While we make the raw materials a bit hard for you, we do this with the objective of challenging and expanding your intellectual capabilities in software development. (This is different than in the game where we find hard raw materials to avoid a high payout.) You finish the raw materials by finding solutions to the subprojects and you will be paid by the quality of your finished product (the quality of your robot will strongly influence your grade). Your real payout, however, is the set of new skills you learn about software development.

You can learn about managing software development by learning the theory and by practicing the management of a software development project. It is my belief that without having experienced a project, you won't absorb the theory well. That is why we experience the project of developing algorithmic players.

## 6 Design

The requirements don't specify how the players communicate with each other. There are two options: a centralized versus a decentralized design. We use a centralized design using an administrator. The players and the administrator communicate through XML documents.

## 7 Implementation

We use Java as implementation language and a Java data binding tool to automatically generate parsers and printers from an XML schema.

In the implementation we practice Adaptive Programming (AP). In AP, as little structural information as possible is spread as narrowly as possible. (A good design principle in general not only for structural information.)

## 8 To Integrate

```
What happens if we use the model: derivative = (Pred, price, quality)
where quality is a function that says which quality we must reach to win.

Q:  quality of finished product found by buyer
SQ: quality of the secret finished product of creator
P:  price of derivative

quality may depend on SQ and price.
e.g., quality = SQ * price.

secret:
=======================================
```

```
win = Q >= quality


profit (after paying price P) =

 if win then P*(1 + INTEREST) + INCENTIVE(Q - quality) else 0

 where INTEREST is a small percentage we get paid because
 we had to spend P for a round to get the derivative.
 We set INTEREST = 10% = 0.1.
 INCENTIVE is a factor that determines the reward for
 doing better than what we are supposed to do.

Problem: price not important?

Duc suggests:

profit = 2Q - SQ - P + 0.1

This formula will require a robot to be smart in not
only buying and finishing the product.
It takes into consideration the price as it was intended
to be. So even if the buyer finishes at a higher
quality than the seller, they can still lose
money if they can't beat the price (which means they are stupid in buying).

This formula also provides incentives for robots to actually
play the Secret Derivative variation.
As a buyer, they can earn even more money if their
finishing agent is better than the seller finishing agent.
As a seller, they can lower their loss on the buying back
of the finish product if they can finish much better than the buyer.
Heck, if the buyer is really dumb, they buyer will actually have to pay even more to the seller.
```

related work:

Okamoto, M., Kita, H., Ono, I., Kiga, D., Terano, T., Yamada, T. & Koyama, Y. (2008). Project-Based Learning of Computer Programming Using an Artificial Market System. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008 (pp. 4545-4553). Chesapeake, VA: AACE.

U-MART project: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01419804

**Acknowledgements:** Feng Zhou and Milena Georgieva Dimitrova have given detailed feedback on the requirements and they have written the previous version of the requirements.