SCG Example Labs

Ahmed Abdelmeged
Karl Lieberherr
Structures of SCG

• These examples are to be read with the SCG paper as background.

• The best way to represent Domain, Lab and Claim is to have Domain and Lab as top-level classes and Claim nested inside Lab. Lab has a Domain as field to give all claims access to Domain functionality.

• Instance and Solution are nested inside Domain.
Structures of SCG

• We use a Java-like syntax but the goal is to use only one or two lines per item for those simple introductory labs.
Structures of SCG

**Domain**
Instance
Solution

**valid** (i: Instance, s: Solution)
**quality** (i: Instance, s: Solution)

**Lab**
d : Domain
proto: Protocol
Claim
  claim parameters
  isp (i: d.Instance)
  p (I: d.Instance[], S: d.Solution[])
  stronger (c2: Claim)
  distance (c2: Claim)

1/9/2013
Calculus Lab

• Have your students mastered calculus (minimizing and maximizing functions)?
• The next lab shows a lab to test their skills.
Structures of SCG

Domain
Instance
Solution
valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

Lab
 d: Domain
proto: Protocol
Claim
 claim parameters
isp(i:d.Instance)
p(I:d.Instance[],S:d.Solution[])
stronger(c2: Claim)
distance(c2: Claim)

Example: calculus problem

SaddlePoint
Instance = [0,1]
Solution = [0,1]
valid(i,s) = true
quality(i,s) = i*s + (1-i)*(1-s^2)

SaddlePointLab
SaddlePoint
O:I[0], P:S[1] of I[0]
SaddlePointLabClaim
 q: [0,1]
isp(i)=true
p(I,S)=d.quality(I[0],S[1])> =q
stronger(c2) = q>c2.q
distance(c2) = q-c2.q

new SaddlePointLab.Claim(q=0.6)
Programming an Algorithm

• Have your students understood the Gale-Shapley algorithm?

• Next come two labs where they can demonstrate their skills through their avatar in a full-round-robin tournament.
  – In the first lab they test each other’s programs to see whether they match each other’s best solution.
  – In the second lab, they find worst-case inputs.
Structures of SCG

**Domain**
- Instance
- Solution

**valid**(i: Instance, s: Solution)

**quality**(i: Instance, s: Solution)

---

**Lab**

**d**: Domain
- proto: Protocol
- Claim
  - claim parameter definitions
  - isp(i:d.Instance)
  - p(I:d.Instance[], S:d.Solution[])
  - stronger(c2: Claim)
  - distance(c2: Claim)

---

**Example: GS algorithm**

**GaleShapleyBasic**
- Instance = Preferences
- Solution = Assignment

**valid**(i,s) = s is syntactically correct for i

**quality**(i,s) = s is semantically correct for i
- 1 if true, 0 if false.

**GaleShapleyBasicLab**
- GaleShapleyBasic
- O:[]I[0], P:S[1] of I[0], O:S[2] of I[0]
- GSAtLeastAsGoodAsYouClaim
  - none
  - isp(i)=true
  - p(I,S)=d.quality(I[0],S[1])>=d.quality(I[0],S[2])
  - stronger(c2) = false
  - distance(C2) = 0

- new GSAtLeastAsGoodAsYouClaim()
Structures of SCG

**Domain**
- **Instance**
- **Solution**

**valid**(i: Instance, s: Solution)
**quality**(i: Instance, s: Solution)

---

Example: Worst-Case of GS algorithm

**GaleShapley (GS)**
- **Instance** = Nat (number of people)
- **Solution** = Preferences

**valid**(i,s) = s is syntactically correct for i
**quality**(i,s) = GS iterations for s and i

**GaleShapleyWorstCaseLab**
- **GaleShapley**
  - O:I[0], P:S[1] of I[0]
- **GSWCLClaim**
  - n:Nat, q:Nat
  - **isp**(i)=i=n (singleton)
  - **p**(I,S)=d.quality(I[0],S[1])\geq q
  - **stronger**(c2)=this.q>c2.q
  - **distance**(c2)=this.q-c2.q

```plaintext
new GSWCLClaim(n=10,q=30)
```
Maximum Satisfiability

• The next lab is about a paper by David Johnson in the 1970’s which is covered now in some algorithm text books, like Kleinberg and Tardos.

• The following MaxSat lab covers several skills, such as working with randomized algorithms and then derandomizing them.
Structures of SCG

**Domain**
- Instance
- Solution

**valid** \(i: \text{Instance}, s: \text{Solution}\)

**quality** \(i: \text{Instance}, s: \text{Solution}\)

Lab
- d: Domain
- proto: Protocol
- Claim
  - claim parameters
  - isp\(i:d.\text{Instance}\)
  - p\(I:d.\text{Instance}[],S:d.\text{Solution}[]\)
  - stronger\(c2: \text{Claim}\)
  - distance\(c2: \text{Claim}\)

MaxSatLab
- Satisfiability
- CNF
- Assignment

valid\(i,s\) = all variables in i assigned once

quality\(i,s\) = fraction of satisfied clauses in i

Example: MaxSat

new MSLClaim\(q=1-(1/2^3), k=3\)
Generalized MaxSat

• The next lab is based on a paper by Lieberherr and Specker (2012).
Structures of SCG

**Domain**
- **Instance**
- **Solution**

valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

**Example: Boolean GeneralizedMaxSat = BMaxCSP**

**BooleanCSP**
- Sequence of Boolean constraints
- Assignment

valid(i,s) = all variables in i assigned once
quality(i,s) = fraction of sat. constraints in i

**Lab**
- d: Domain
- proto: Protocol
- Claim
  - claim parameters
  - isp(i:d.Instance)
  - p(I:d.Instance[],S:d.Solution[])
  - stronger(c2: Claim)
  - distance(c2: Claim)

**BooleanMaxCSPLab**
- BooleanCSP
- O:I[0], P:S[1] of I[0]

GenBooleanMaxSatClaim
- q:[0,1], r:{R1,R2,…}
- isp(i)=constraints in i use only r
- p(I,S)=d.quality(I[0],S[1])>=q
- stronger(c2)=q>c2.q
- distance(c2)=q-c2.q

new GenBooleanMaxSatClaim(q=0.618, r={R1,R2})
Local to Global

• The next lab is based on several papers inspired by a JACM paper by Lieberherr and Specker in 1981.
• The lab is about studying how local properties of a conjunctive normal form translate into global properties.
Structures of SCG

**Domain**
- **Instance**
- **Solution**

valid\((i: \text{Instance}, s: \text{Solution})\)

quality\((i: \text{Instance}, s: \text{Solution})\)

**Example: BooleanMaxCSPLocalGlobal**

BooleanCSP
- Sequence of Boolean constraints
- Assignment
valid\((i, s) = \text{all variables in } i \text{ assigned once}\)

quality\((i, s) = \text{fraction of sat. constraints in } i\)

Lab
- **d**: Domain
- **proto**: Protocol
- **Claim**
  - **claim parameters**
  - **isp**(i: d.Instance)

\[ p(I:d.\text{Instance}[], S:d.\text{Solution}[]) \]

stronger(c2: Claim)
distance(c2: Claim)

new BMCLClaim\((q=0.618, r=\{R1,R2,R3,R4\}, k=2)\)
Manufacturing Lab

• The next lab is about an efficient manufacturing problem where raw materials are turned into a product.
• The lab is underspecified in that the details about the isp function are missing.
Structures of SCG

Domain
Instance
Solution
valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

Lab
d: Domain
proto: Protocol
Claim
  claim parameters
  isp(i:d.Instance)
  p(I:d.Instance[],S:d.Solution[])
  stronger(c2: Claim)
  distance(c2: Claim)

Example: Solar Cells

SolarCells
RawMaterials
Product
valid(i,s) = only raw materials used
quality(i,s) = energy efficiency of s for i

SolarCellsLab
SolarCells
O:I[0], P:S[1] of I[0]
SCLClaim
  q:[0,1], k:Nat (raw material parameter)
  isp(i)= ...
p(I,S)=d.quality(I[0],S[1])> q
  stronger(c2)=q>c2.q
  distance(c2)=q-c2.q

new SCLClaim(q=0.7,k=3)
Sums of Arithmetic Sequences

• The next labs deal with skills needed to sum arithmetic sequences efficiently.
• We use the variables and notation introduced on the next slide.
expression in a, d, n using multiplication, addition and division. To simplify, replace \((1+2+ \ldots +n)\) by \(n(n+1)/2\).

\[
\sum_{k=1}^{n} a + dk = (a + d) + (a + 2d) + (a + 3d) + \ldots + (a + nd) = na + (1 + 2 + 3 \ldots + n)d
\]
Constructive / Proof only

• The first lab is constructive in that the solution is an expression which sums the arithmetic sequence efficiently.

• In the second lab, a specific expression is given and the lab only asks for an inductive proof.

• Key to the solution for the constructive lab is a reduced lab where the first n natural numbers are added. Formula is on previous slide.
Structures of SCG

**Domain with name**
- **Instance**
- **Solution**

valid(i: Instance, s: Solution)

quality(i: Instance, s: Solution)

---

**Example: Arithmetic Sequences Sum**

ArithmeticSequences2

i=triple (a,d,n: Nat)

expression in variables a,d,n

valid(i,s) = s uses +,*,/ and vars in i

quality(i,s) = 1 if s is correct for i, 0 otherwise

---

**Lab**

- **d: Domain**
- **proto: Protocol**
- **Claim**

  claim parameters
  isp(i:d.Instance)
  p(I:d.Instance[],S:d.Solution[])  
  stronger(c2: Claim)
  distance(c2: Claim)

---

**ArithmeticSequencesSumLab**

ArithmeticSequences2

O:I[0], P:S[1] of I[0], O:S[2] of I[0]

ASeqSumALAGAYLClaim

i1:Instance

isp(i)=(i=i1)

p(I,S)=d.quality(I[0],S[1])>=d.quality(I[0],S[2])

stronger(c2)=false

distance(c2)=0

---

new ASeqSumLClaim((a=2,d=3,n=100000))
Structures of SCG

Domain with name
Instance
Solution
valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

Lab

d: Domain
proto: Protocol
Claim
claim parameters
isp(i:d.Instance)
p(I:d.Instance[],S:d.Solution[])
stronger(c2: Claim)
distance(c2: Claim)

Example: Arithmetic Sequences Sum

ArithmeticSequencesInduction
sum[k=1..n] 2+3k = 2n+3(n(n+1))/2
sequence of steps: induction proof
valid(i,s) = proof s is syntactically correct
quality(i,s) = 1 iff proof is correct

ArithmeticSequencesInductionLab
ArithmeticSequencesInduction
O:I[0], P:S[1] of I[0]
ASLClaim
i1:Instance
isp(i)=(i=i1) //singleton
p(I,S)=d.quality(I[0],S[1])=1
stronger(c2)=false
distance(c2)=0

new ASLClaim(sum[k=1..n] 2+3k = 2n+3(n(n+1))/2)
Highest Safe Rung

• This is a fun lab which can be formulated in many different ways. Inspired by Kleinberg and Tardos’ Algorithm Design book.
Structures of SCG

Domain with name
Instance
Solution
valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

Example: HighestSafeRung

HighestSafeRung
pair(n,k)
decision tree
valid(i,s) = s is correct for (n,k)
quality(i,s) = depth(s)

Lab

d: Domain
proto: Protocol
Claim
claim parameters
isp(i:d.Instance)
p(I:d.Instance[],S:d.Solution[])
stronger(c2: Claim)
distance(c2: Claim)

HighestSafeRungLab
HighestSafeRung
O:I[0], P:S[1] of I[0]
HSRClaim
n:Nat,k:Nat,q:Nat
isp(i)=(i=(n,k)) // singleton {(n,k)}
p(I,S)=d.quality(I[0],S[1])<=q
stronger(c2)=q<c2.q
distance(c2)=q-c2.q

new HSRClaim(n=25,k=2,q=5)

lower quality is better
Structures of SCG

**Domain with name**
- Instance
- Solution

valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

**Lab with name**
- d: Domain
  - claim parameter definitions
  - instance set predicate
  - refutation predicate
  - protocol
  - stronger(c1,c2: Claim)
  - distance(c1,c2: Claim)

**Claim with name**
- proponent: Scholar
- lab : Lab
  - claim parameter values

Example: LeafCovering

LeafCovering
Set of trees. Set M=subset of GCP of trees. witness (leaf in GCP) of non-coverage by M
valid(i,s) = s is correct for i
quality(i,s) = unused
Structures of SCG

**Domain with name**
- **Instance**
- **Solution**

valid(i: Instance, s: Solution)
quality(i: Instance, s: Solution)

**Lab with name**
- **d**: Domain
- **claim parameter definitions**
- **instance set predicate**
- **refutation predicate**
- **protocol**
- **stronger(c1,c2: Claim)**
- **distance(c1,c2: Claim)**

**Claim with name**
- **proponent**: Scholar
- **lab**: Lab
- **claim parameter values**

Example: LeafCovering

LeafCovering
LeafCoveringProblem: Set of trees. Set M=subset of GCP of trees.
Program
valid(i,s) = s is correct for i
quality(i,s) = unused

LeafCoveringLab
LeafCovering
m: Nat (size of M)
instanceSetP(i,m)= |i.M|=m
quality(i[0],s[1])<=q
O:i[0], P:s[1] of s[0]
c1.q>c2.q
c1.q-c2.q

HSRClaime
- **Alice**
- **HighestSafeRungLab**, 25,2,5
<table>
<thead>
<tr>
<th>claim</th>
<th>action</th>
<th>outcome</th>
<th>P</th>
<th>O</th>
<th>cB</th>
<th>aB</th>
<th>oB</th>
<th>np</th>
</tr>
</thead>
<tbody>
<tr>
<td>P not per</td>
<td>F * a</td>
<td>sO</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T a</td>
<td>sO</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>O not per</td>
<td>F * d</td>
<td>sP</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T d</td>
<td>sP</td>
<td>1</td>
<td>-1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>F a</td>
<td>rP</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T * a</td>
<td>rP</td>
<td>-1</td>
<td>-1</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F d</td>
<td>rO</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P not per</td>
<td>T * d</td>
<td>rO</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SCG Truth Table Interpretation

- no competition: P 0 and O 0 everywhere
- not fair: the player different from the “not perfect” player loses a point
<table>
<thead>
<tr>
<th>claim</th>
<th>dec</th>
<th>out</th>
<th>P</th>
<th>O</th>
<th>cB</th>
<th>aB</th>
<th>oB</th>
<th>Blame Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>F * a</td>
<td>sO</td>
<td>0</td>
<td>0</td>
<td>P</td>
<td>O</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T a</td>
<td>sO</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F * d</td>
<td>sP</td>
<td>1</td>
<td>-1</td>
<td>P</td>
<td>-</td>
<td>O</td>
<td></td>
<td>O did not refute a claim it disputed</td>
</tr>
<tr>
<td>T d</td>
<td>sP</td>
<td>1</td>
<td>-1</td>
<td>-</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F a</td>
<td>rP</td>
<td>1</td>
<td>-1</td>
<td>P</td>
<td>O</td>
<td>O</td>
<td></td>
<td>O failed to support a claim it agreed with</td>
</tr>
<tr>
<td>T * a</td>
<td>rP</td>
<td>1</td>
<td>-1</td>
<td>-</td>
<td>-</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F d</td>
<td>rO</td>
<td>-1</td>
<td>1</td>
<td>P</td>
<td>-</td>
<td>P</td>
<td></td>
<td>P failed to support a claim it proposed</td>
</tr>
<tr>
<td>T * d</td>
<td>rO</td>
<td>-1</td>
<td>1</td>
<td>-</td>
<td>O</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>claim</td>
<td>dec</td>
<td>out</td>
<td>P</td>
<td>O</td>
<td>cB</td>
<td>aB</td>
<td>oB</td>
<td>Blame Justification</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>F *</td>
<td>a</td>
<td>sO</td>
<td>paso</td>
<td>oaso</td>
<td>P</td>
<td>O</td>
<td>-</td>
<td>O did not refute a claim it disputed</td>
</tr>
<tr>
<td>T</td>
<td>a</td>
<td>sO</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>F *</td>
<td>d</td>
<td>sP</td>
<td>pdsp</td>
<td>odsp</td>
<td>P</td>
<td>-</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>d</td>
<td>sP</td>
<td>-</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>a</td>
<td>rP</td>
<td>parp</td>
<td>oarp</td>
<td>P</td>
<td>O</td>
<td>O</td>
<td>O failed to support a claim it agreed with</td>
</tr>
<tr>
<td>T *</td>
<td>a</td>
<td>rP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>d</td>
<td>rO</td>
<td>pdro</td>
<td>odro</td>
<td>P</td>
<td>-</td>
<td>P</td>
<td>P failed to support a claim it proposed</td>
</tr>
<tr>
<td>T *</td>
<td>d</td>
<td>rO</td>
<td>-</td>
<td>O</td>
<td>O</td>
<td></td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>