Programming Problem
for the
2001 Comprehensive Exam

Out: Monday 1/22/00 at 9:00 am
Due: Friday 1/26/00 at 5:00 pm

Department of Computer Science
Brown University
Providence, RI 02912

1.0 The Task

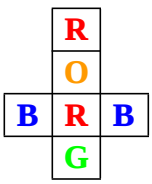
This year you are going to write the core of a program that could play many of the 30 games that compose On-Sets.

On-Sets is a game of set theory. It is loosely based on the well-known (in some elite circles) game of Wff 'N Proof, "the game of modern logic". Wff 'N Proof originated in the 1960s and was quite popular in math departments (there was very little CS in those days) in the late 60's. Playing it is supposed to raise your IQ score by 20 points. On-Sets is a somewhat simpler game that involves elementary set theory (i.e. basic set operations) rather than formal proofs.

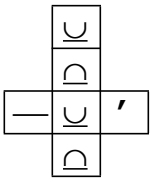
The objective of On-Sets is to construct a pair of formulas (a *restriction* and an *equation*) that define a subset of a specific size. A deck of cards is used to define the original set. Cubes are used to provide the elements of the formulas. There are actually 30 different games in the On-Sets package. These vary in complexity and scope. What you are going to do in this assignment is to create a basic framework that is suitable for playing any of the games.

2.0 Game Basics

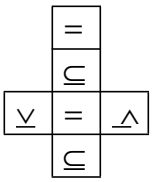
The game equipment is fairly simple. It consists of 18 cubes, 15 cards, 2 playing mats, and a 88 page instruction manual that primarily teaches basic set theory. The 18 cubes are divided into 4 groups. The first 8, the *color cubes*, all look like this:



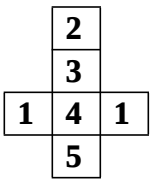
where **R** stands for a red dot, **O** for an orange dot, **G** for a green dot, and **B** for a blue dot. The second group consists of 4 cubes of *set operators*. These look like:



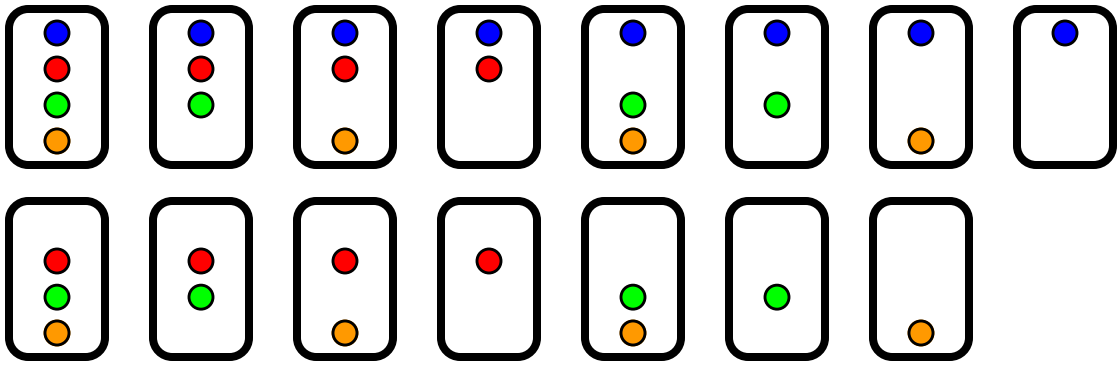
The symbols here are the operators for union, intersection, set difference, and set compliment. The third group consists of 3 cubes of *relational operators*. These look like:



The symbols here are the operators for set equality, set containment, the set of all displayed cards (⊻), and the empty set (⊼). The final group of 3 cubes comprise the *number cubes*. These contain numbers and look like:



The fifteen cards each contain between one and four colored circles:



(Note that the top color is always blue, the second color red, the third green, and the fourth orange.)

The two playing mats (which are logically one) are used to organize the cubes for the various games. They define 7 areas:

- *Resources*: the initial, unplayed cubes;

- *Forbidden*: cubes that can't be used;
- *Permitted*: cubes that may be used;
- *Required*: cubes that must be used;
- *Restriction*: a Boolean expression that restricts the set of cards under consideration;
- *Equation*: a set expression that defines a subset of the cards under consideration after restrictions of a given size; and
- *Goal*: a numeric expression that defines the size of the target set.

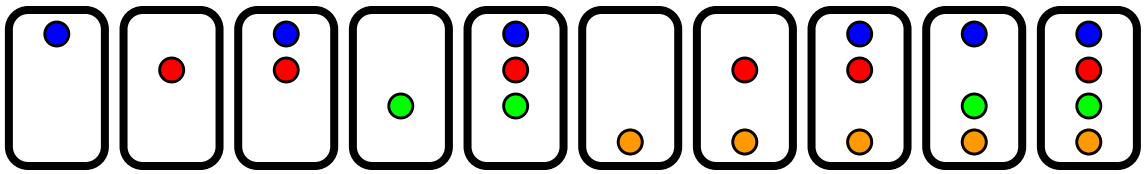
Finally, the instruction book provides the rules for the basic game (the simplest), the advanced game (the most complex) and 28 games in between. Primarily, however, it provides the reader with the basic fundamentals of set theory. (Because of this, and the fact that the pages are small, it is fairly easy and quick to read.)

3.0 Playing On-Sets

Almost all the various On-Sets games are played the same way. Each game consists of three stages: *setting up*, *playing*, and *going-out*.

Setting up involves first selecting the cubes that are going to be used in the game and the number of cards to be used, then playing the cubes and dealing the cards, and finally using the number cubes to determine the goal of the game. The steps involved in setting up are:

- First, the cards are shuffled and 6 or more cards are dealt face up on the table. These cards denote the *base set*. The number of cards can be anywhere between 6 and 15 and is chosen by the dealer.
- Second, a shaking collection of cubes is selected. This consists of 3 or 4 set-operator cubes, 0-3 relational-operator cubes, one more color cube than the total number of set and relational operator cubes, and 3 number cubes.
- Third, the selected collection of cubes is rolled so that each of the cubes has one of its faces selected.
- Fourth, all the cubes that are not number cubes are placed in the Resources area of the mats.
- Finally, the player doing the setup chooses between 1 and 3 of the number cubes and arranges them to form a *goal*. If one is used, its number is the goal. If two are used, they can be placed next to one another to indicate the sum of the two numbers is the goal, or they can be placed one above the other to indicate the product of the two numbers is the goal. If all three cubes are used, the goal can be any combination of addition or multiplication of the three cubes. Because there are no parenthesis cubes, On-Sets lets the player define arbitrary groupings using spacing. This is equivalent



Resources: ≜ ⊆ ∩ — ∪ **R R O B B O**

Required:

Forbidden: **3 1**

Permitted:

Restriction:

Equation: **Goal:** **3**

FIGURE 1. Sample initial configuration.

to permitting the use of arbitrary parenthesis in defining the expression. Note also that some of the simpler games only allow the use of addition in forming the goal.

The number cubes that are used are arranged in the result portion of the equation area on the mats. Any remaining number cubes are placed in the Forbidden area. The goal that is set here is fixed for the remainder of the game.

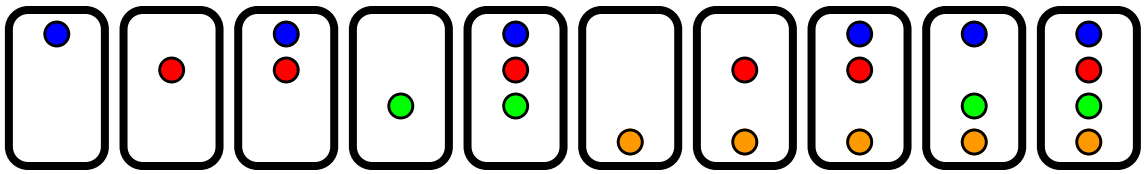
Figure 1 shows an example of an initial configuration. Here there are ten cards, two relational-operator cubes, three set-operator cubes, six color cubes, and three number cubes. The goal was set to three using a single cube.

Playing involves moving cubes from the Resources area to one of the areas Forbidden, Permitted, or Required. Normally, only one cube is moved on a turn, however, it is permissible to move two cubes if one moves the first to the Forbidden area (and claims a “bonus” before doing so).

Figure 2 shows the previous example after a series of such plays have been made. The specific plays here involved:

Moving ⊆ to Permitted.
Moving **O** to Forbidden.
Moving **R** to Required.
Moving **O** to Required.
Moving — to Forbidden.
Moving **R** to Required.
Moving ∪ to Required.

Going-Out involves creating an equation and an optional restriction using zero or more of the cubes that have been moved out of the Resources area to



Resources: △ ∩ **B B**

Required: ∪ **R R O**

Forbidden: **3 1** — **O**

Permitted: ⊆

Restriction:

Equation: **Goal: 3**

FIGURE 2. Configuration after a series of moves.

either the Required or Permitted areas to define a set. The set is defined by forming an equation that specifies a subset of the dealt cards whose size matches the goal set by the number cubes and, optionally, an expression that restricts the set of cards to be considered in constructing the equation. This is done in steps as follows:

- First a restriction formula *may* be defined in the Restriction area of the playing mat. If such a formula is defined, then it must include all the cubes that have been placed in the Required area and may include any of the cubes from the Permitted area or the one cube chosen from Resources. No other cubes can be used. Note that the restriction is a Boolean statement (and not a set-valued statement) and hence must contain one of the operators = or ⊆. Note also, that the only way of using this operator is at the outermost level of the expression.

  This formula is then used to create a subset of the original set of cards. Note that in the formula, the color cubes refer to the subset of the cards that have the corresponding color showing. Thus a color cube with blue showing indicates the subset of the original cards that have a blue dot on them. The compliment operator is the finite compliment (i.e. all cards from the original set that are not in the set that is its operand). Again, because there are no parenthesis cubes, arbitrary groupings are allowed.

  The subset that is specified by a given restriction formula is the maximal subset of the original set that makes the formula true.

  From the configuration of Figure 2, one could define the restriction:

  (**R** ∪ **R**) ⊆ **O**

  This says that the set of cards containing a red dot must be contained in the set of cards containing an orange dot. This can be achieved by ensuring that there are no cards that contain a red dot by not an orange one. The maximal such set is achieved by eliminating the second, third and fifth cards from

the original set. The remaining seven cards would then be considered when building the equation below. Note that the union subexpression is needed here because both red cubes must be used (since they are in the Required area).

- Second, a set equation is formed in the Equation area of the mat. This equation must use all the cubes that were played to the Required area other than those showing = or ⊆; and can use any of the cubes that had been played to the Permitted area or the one cube chosen from Resources. It can not use any cubes from the Forbidden area. Note that this will involve reusing the cubes that might have been used previously in the restriction formula. If a restriction was played, the set expression is evaluated with respect to the restricted subset that resulted; otherwise the expression is evaluated with respect to the dealt cards. The size of the set that results from this equation must contain exactly the number of elements specified by the goal. Again, the player is free to use arbitrary parenthesis in forming the equation.

  Given the example of Figure 2, one could use the restriction cited above along with the equation:

  R ∩ (**O** ∪ **R**)

  Here the intersection operator is the one chosen from the Resources area in order to make the move. As an alternative, the player could have chosen to not create a restriction and to define the equation:

  O ∩ (**R** ∪ **R**)

The actual game also contains a complex set of rules dealing with challenges. These are used to ensure that the game proceeds legally and that it starts out and stays possible to eventually go out. They also deal with what cubes can be used after a challenge is successfully made so the game can continue. Finally, the game rules also deal with scoring, with points being granted for going out and for successfully challenging, and points being deducted for incorrect challenges and incorrect moves. Since we are going to have the computer play, we will assume that all moves are legal and will ignore these rules (and the corresponding strategies) for now.

If this brief summary explanation is unclear, you should refer to the complete On-Sets manual that accompanies this assignment.

4.0 Computer On-Sets

Your assignment is to write a computer program that does most of the work of playing On-Sets. The program will set up a game according to a given specification; assist in playing the game by finding all the legal moves at each point, letting the user choose one, making that move, and then continuing; and assist in going out by listing possible going out moves at each point.

In the following, we use the term *configuration* to describe the state of the game. This includes the cards that have been dealt, the cubes that have been thrown, the location of each cube (i.e. Resources, Required, Forbidden, or Permitted), the goal (i.e. how the number cubes are used), and, if the game is solved, the restriction and equation used. A configuration is *legal* if it is a solution or if there is some sequence of moves from that configuration that results in a valid equation or equation-restriction pair that defines a subset whose size matches the given goal.

Your program should do the following:

1. Let the user specify the number and types of cubes to be used and the number of cards to be dealt in the game. This can be done either on the command line or interactively. The user should also be able to specify whether the number cubes can be joined using both addition and multiplication (normal/advanced game) or only with addition (simple game). Note that the number of color cubes is determined by the number of set-operator cubes and the number of relational-operator cubes.
2. Deal the cards and throw the cubes. This should use an appropriate random number generator. (Hint: you might want to use a fixed or reproducible seed for the random number generator for debugging purposes, but should use a random seed in general.)
3. Determine the legal initial configurations. The user should be shown the various groupings of the number cubes and should be able to select one of these groupings as the goal for the game. For simplicity, you can show the number expression in a readable form. (Hint: either postfix or prefix notation will make things simpler, but you can use whatever you want.) Note that this is non-trivial since your program must ensure that each configuration is legal, i.e. that there is a equation or equation-restriction pair that defines a subset of the dealt cards whose size matches the goal size.
4. Output the current configuration. This is needed for debugging and understanding your program. Your output can be as fancy or functional as you desire. The only constraint is that it must be understandable.
5. Produce the list of *legal moves* from the current configuration. Moves are of two types, *going-out* or *moving* cubes. A going out move involves taking one cube from the Resources area and using it and the other cubes that have been played to form a equation and optional restriction yielding a set of the given goal size. (Of course, this must be done according to the rules of the game, using all required cubes and no forbidden ones.)
   The second type of move involves moving one cube from the Resources area to either the Required, Permitted, or Forbidden areas. Your program should list all such legal moves. This only needs to be done if there are no going-out moves available. Again, note that your program will have to check that each such move yields a legal configuration.
6. Display the list of moves that was found and let the user choose which one to make. Your program should then use the user's choice to update the current configuration. If the move made was a going-out move, the program

can terminate. Otherwise, the program should print out the new configuration, and repeat steps 4-6. Again, your output can be as fancy or functional as you desire as long as it is understandable.

To simplify matters, you can output the resultant formulas in prefix or postfix form. (Hint: using prefix or postfix internally might simplify matters.) Also, when you list the moves, you only need to list equivalent moves, not all the moves. For example, if there are two blue cubes available in the resources area, you need only list one move of a blue cube to a region on the mat, not both such moves. Similarly, if two formulas are obviously equivalent, only one has to be listed in a going-out move. (Or course, you can list all if you so desire.)

Your program should be coded relatively efficiently. You should be able to find the set of valid configurations, legal moves, or going-out methods in under a minute (even interpreted — well under is quite possible). If your program takes significantly longer than this, you probably have the wrong algorithms or data structures and should rethink the design.

To keep your sanity, you might want to get a version of your program working without the use of restrictions first, and then, once this is working, add in the use of restrictions. Note, however, that if you do this you should design the overall system so that it can handle restrictions so that they will be easy to add later on.

5.0 Mechanics

You may write this program in any programming language available on any of the Department of Computer Science's systems. (For grading purposes, your program will have to be able to be run on one of our systems.) You can use any libraries that are considered part of that language. For example, the STL is considered part of C++; all classes in java.* or javax.* are considered part of Java. You may use any machines that you normally have access to at Brown or you may use your own personal machine. If you have any questions as to what is or isn't valid, you should ask Dr. Reiss before proceeding.

You should hand in your program (source and executable), sample runs, and whatever output you have to demonstrate your program works. You should also include a brief write-up that describes the algorithms used and how to run your program. Handins should be done electronically if at all possible into the directory `/map/aux1fred/comps/handins/###` where `###` is the random number you assign yourself. You should have permissions to create this directory and should protect it accordingly. All directories will be locked at the moment the exam is due and you will be graded on what is there at the time.

Grading will be based on your choice of data structures and algorithms as well as on the correctness, performance, style, and readability of your program and documentation.

A physical version of the On-Sets game will be available for you to look at and borrow. If you think it will help, you may play On-Sets with others to get a feel for how the actual game goes. If you do so, you should be careful not to discuss how your program may or may not work with those you are playing with.

All work that you hand in must be your own. You should not discuss your work or ideas with anyone else. You should not share code or ask others for advice on your code. You should not read other's code (either those here or code obtained elsewhere) that duplicates in any way what you are to write. You may ask Dr. Reiss questions either in person or via email. Any answers or information that should be known to all will be posted in the comps news group which is your responsibility to read.

Good luck and have fun.