Discussion Problems: "Miscellaneous Problems"
Declarative Methods (JHU 600.325/425)
Prof. Jason Eisner

1. Each worker at the damaged Fukushima Daiichi nuclear plant is required to limit his or her radiation exposure to 30 mSv (millisieverts) in any given consecutive 3-hour period.[1] This is important in scheduling teamwork shifts; here we consider the case of scheduling a single worker.

   In this problem, we will talk about bounds propagation—however, you are free to substitute generalized arc consistency, which will get the same answers in this case.

   Let Akiko's exposure in each of the hours 1, 2, 3, 4, 5, be denoted by variables A, B, C, D, E. Thus,

   ```
   [A,B,C,D,E] :: [0..infinity],
   A+B+C #=< 30,
   B+C+D #=< 30,
   C+D+E #=< 30
   ```

   (a) [3 points] Run bounds propagation on the constraints above. What are the new (reduced and bounds-consistent) domains of A, B, C, D, E?

   (b) [2 points] The cleanup supervisor would like Akiko to do work that would expose her to a total of 65 mSv. To find out whether this work can be safely scheduled within a 5-hour period, what constraint do you add?

   (c) [3 points] Adding the constraint from part (b) makes the system UNSAT (i.e., the work cannot be safely scheduled). But we have to detect that. What are the new domains of A, B, C, D, E after you run bounds propagation with this added constraint? Does this detect that the problem is UNSAT?

[1] I made this up; I couldn't find out what the actual short-term limits are. Their yearly maximum has been raised to 250 mSv.

(d) [4 points] Suppose you do a step of variable elimination and eliminate variable A, which is mentioned by just these 3 constraints:

```
A :: [0..infinity],
A+B+C #=< 30,
new constraint from part (b)
```

I claim that the new constraints inferred by variable elimination will be exactly

```
B+C #=< 30,
D+E #>= 35
```

I'll prove my claim in the official solution set. To prove it yourself, which of the following would you have to show? *(circle <u>all</u> that apply)*

i. If an assignment to (A,B,C,D,E) satisfies the 3 old constraints on A, then (B,C,D,E) must satisfy the 2 new constraints above.

ii. If an assignment to (A,B,C,D,E) satisfies all of the old constraints, then (B,C,D,E) must satisfy the 2 new constraints above.

iii. If an assignment to (B,C,D,E) satisfies the 2 new constraints above, then it can be extended into an assignment to (A,B,C,D,E) that satisfies the 3 old constraints on A.

iv. If an assignment to (B,C,D,E) satisfies the 2 new constraints above, then it can be extended into an assignment to (A,B,C,D,E) that satisfies all of the old constraints.

v. none of the above

(e) Now consider generalized arc consistency (or bounds propagation) on the new set of constraints after A has been eliminated, namely

```
[A,B,C,D,E] :: [0..infinity],
B+C+D #=< 30,
C+D+E #=< 30,
new constraint from part (b),
B+C #=< 30 (inferred by variable elimination),
D+E #>= 35 (inferred by variable elimination)
```

It is now not hard to see that the constraints are UNSAT (since C+D+E $\leq 30 <$ $35 \leq$ D+E, yet C > 0). However, the question is whether bounds propagation will prove that.

i. [2 points] What can bounds propagation immediately conclude about D and E from the facts that `D+E #>= 35` and `D, E :: [0..30]`?

ii. [4 points] The following reduced domains can be justified by some additional steps of bounds propagation. Are the domains bounds-consistent yet, or will more steps of bounds propagation reduce them further?
If they are bounds-consistent, pick one constraint and explain why they are bounds-consistent with respect to that particular constraint. If not, exhibit a constraint with respect to which they are not bounds consistent.

```
B :: [0..25],
C :: [0..20],
D :: [5..30],
E :: [5..30]
```

2. Recall that 3-SAT is NP-complete. 3-SAT is the satisfiability problem for boolean CNF formulas in which each clause has ≤ 3 literals.

   Recall also that 2-SAT can be solved in polynomial time, for example by the DPLL algorithm.

   Let's define two new problems:

   "$2\frac{1}{2}$-SAT" is the satisfiability problem for boolean CNF formulas where exactly half the clauses have exactly 3 literals and the other half have exactly 2 literals. (The number of clauses must be even.) For example,

   $$(A \vee B) \wedge (\neg A \vee C \vee D) \wedge (\neg B \vee C \vee \neg E) \wedge (\neg C \vee \neg E)$$

   "2-and-a-few-SAT" is the satisfiability problem for boolean CNF formulas where up to 3 clauses have exactly 3 literals but all other clauses have exactly 2 literals.

   You don't have to give completely formal proofs below, but say enough to convince us that you have figured out precisely what encodings to use for the reductions. Make sure to read the above definitions *carefully*, or you will probably be unable to get full credit.

   (a) [7 points] Show that 2-and-a-few-SAT can be solved in polynomial time. You can do this by reducing it to another easy problem, such as 2-SAT. In other words, assume a fast solver for 2-SAT, and use it to solve 2-and-a-few-SAT fast.

(b) [8 points] Show that $2\frac{1}{2}$-SAT is NP-complete. You can do this by reducing another hard problem (an NP-complete problem such as 3-SAT) to $2\frac{1}{2}$-SAT. In other words, show that a fast solver for $2\frac{1}{2}$-SAT is unlikely because it would yield a fast solver for the other hard problem.

3. The Peano integers have the form `z`, `s(z)`, `s(s(z))`, ... They represent the natural numbers. Here is a definition of addition from class:

```
add(z,B,B).
add(s(A),B,s(C)) :- add(A,B,C).
```

(a) [3 points] The above definition ensures that `add(`i, j, k`)` is provable iff $i + j = k$, assuming that i, j, and k are all Peano integers.

But what if i, j, and k are *not* all Peano integers (i.e., at least one of them is some other kind of ground term)? Might `add(`i, j, k`)` be provable even so? Defend your answer.

(b) [3 points] If n is a Peano integer, let `pos(`n`)` and `neg(`n`)` be *signed Peano integers* that represent $+n$ and $-n$. For example, `pos(s(s(z)))` represents +2 and `neg(s(s(z)))` represents -2. (Notice that `pos(z)` and `neg(z)` both represent 0.)

Define a predicate `negate/2` such that `negate(`x`,`y`)` is provable iff x and y have opposite signs (`pos` vs. `neg`) but are otherwise the same number. You may assume that x and y are both signed Peano integers.

(c) [6 points] Define a predicate `signed_add/3` such that if i, j, and k are all signed Peano integers, then `signed_add(`i, j, k`)` is provable iff $i + j = k$. Your definition may make use of `add` and/or `negate`.

4. In class, we implemented a Prolog predicate `deletefirst(Z,Xs,Ys)` that is true iff the list `Ys` is obtained by deleting the *first* occurrence of `Z` from the list `Xs`.

   Here was the code:

```
deletefirst(X,[X|Xs],Xs) :- !.
deletefirst(Z,[X|Xs],[X|Ys]) :- deletefirst(Z,Xs,Ys).
```

   The first two arguments to `deletefirst` are assumed to be *ground*; this is important because the code is impure Prolog, using a cut.

   (a) [3 points] Why doesn't the code include a base case that matches `deletefirst(Z,[],Ys)`?

   (b) [4 points] Suppose you mistakenly called `deletefirst` with non-ground `Z`, e.g., `deletefirst(Z,[1,2,3],Ys)`. What set of answers would you get? What answers are "missing" from this set?

   (c) [8 points] Give a pure Prolog implementation of `reverse(As,Bs)`. What is the runtime when `As` is a ground list of length n and `Bs` is a free variable?

(d) [6 points] Give a Prolog implementation of `deletelast(Z,Xs,Ys)`, based on calling `reverse` and `deletefirst`. Assume that Z is ground, that `Xs` is a ground list of length n, and that `Ys` is free. What is the runtime?

(e) [8 points] Give a different Prolog implementation of `deletelast(Z,Xs,Ys)`, in which you solve the problem directly without using `reverse`. You may use the cut, and you may use `member`. You can do this in 2 or 3 lines. What is the runtime?

(f) [10 **extra credit** points] Give an answer to the previous problem that runs in $O(n)$ time.

Hint: The basic idea is to solve the problem recursively, and have the recursive call return an extra value saying whether it managed to delete `Z`. You may have to define some extra predicates besides `deletelast`, to help out.