CSC 373 | Additional Example | Winter 2019

Show that CNF-SAT is polytime self-reducible.

** For maximum benefit, try this for yourself before looking at the answers! And treat each question as a hint: think about it and come up with the answer (and work out the rest of your solution) before looking at the way we wrote it up. **

Q: What does this means exactly? What do we need to set up?
A: Need a precise description of CNF-SAT decision problem and CNF-SAT search problem.

CNF-SAT decision -- from class:
- Input: A propositional formula \phi in CNF.
- Output: Is there is a satisfying assignment for \phi?

Q: CNF-SAT search = ?
A:
- Input: A propositional formula \phi in CNF.
- Output: A satisfying assignment for \phi, if one exists -- the special value NIL otherwise.

Q: What does self-reducibility mean again?
A: That any efficient algorithm for CNF-SAT-D can be used to write an efficient algorithm for CNF-SAT-S.

Q: How do we prove this?
A: Suppose CSD(\phi) is an algorithm that solves CSD in polytime, then write an algorithm CSS(\phi) that solves CSS by making calls to CSD. Argue that CSS is correct and also runs in polytime.

Let’s try to come up with the ideas for algorithm CSS, step-by-step, then we’ll put it all together.

Q: What should CSS(\phi) do first?
A: Check that \phi is satisfiable -- if not, return NIL immediately:

```
if not CSD(\phi):  return NIL
```

Q: Next, CSS must find a satisfying assignment. This is the part that requires the most creativity, so let’s think it through carefully. We have at our disposal a tool that allows us to find out whether or not any formula is satisfiable. But when it reports that a formula is satisfiable, it does not directly give us any information about how to set the variables to make this happen. Is there a way to get information about the values of variables indirectly? Perhaps by asking about different, related formulae?

> Q: Hint: If \phi is satisfiable and x is any variable in \phi, we know that x must be either True or False, and at least one of these choices will satisfy \phi. Is there a way to find out?
> A: If we set x = True inside \phi, and the resulting formula is still satisfiable, then \phi is satisfiable by setting x = True; if \phi is no longer satisfiable, then we must set x = False.
>
> Q: But wait, what does it mean to "set x = True inside \phi"? We have to make sure that the result is still a propositional formula, in order to be able to call CSD on the result.

A: Set x = True and simplify \phi, using propositional equivalences: True \/ A = True; True /\ A = A; False \/ A = A; False /\ A = False. The result of doing this for each clause is denoted \phi[x = True].

To make sure this is clear for everyone, here are some examples. Let \phi = ~x3 /\ (x1 \/ ~x2 \/ x3) /\ (~x5 \/ x4).

```
- \phi[x1 = True]
    = ~x3 /\ (True \/ ~x2 \/ x3) /\ (~x5 \/ x4)
    = ~x3 /\ True /\ (~x5 \/ x4)
    = ~x3 /\ (~x5 \/ x4)

- \phi[x2 = True]
    = ~x3 /\ (x1 \/ False \/ x3) /\ (~x5 \/ x4)
    = ~x3 /\ (x1 \/ x3) /\ (~x5 \/ x4)

- \phi[x3 = True]
    = False /\ (x1 \/ ~x2 \/ True) /\ (~x5 \/ x4)
    = False
```

Q: What are the possible outcomes?
A: Either this will result in some propositional formula \phi’ (like in the first two examples), or it will result in False (if the last literal in some clause is set to False), or it will result in True (if every clause has been satisfied).

Q: What do we do in each case?
A:
- Result = True: \phi is satisfiable by setting x = True and all remaining variables arbitrarily.
- Result = False: \phi is not satisfiable by setting x = True, so set x = False.
- Result = \phi’: Check if \phi’ is satisfiable or not; if it is, set x = True; otherwise, set x = False.

Q: Final algorithm?
A:
```
CSS(\phi):
    if not CSD(\phi):  return NIL
    for each variable x in \phi:
        \phi’ = \phi[x = True]
        if \phi’ = True:
            set all remaining variables in \phi (including x) to True
            exit loop
        else if \phi’ = False or not CSD(\phi’):
            set x = False
            \phi = \phi[x = False]
        else: # \phi’ != True and \phi’ != False and CSD(\phi’)
            set x = True
            \phi = \phi’
```

NOTE: It’s important to update \phi at each iteration, to get a complete satisfying assignment of values to the variables of \phi. For example, (x \/ y) /\ (~x \/ ~y) can be satisfied by setting x = True, and it can also be satisfied by setting y = True, but not both at the same time!

Trace on example \phi = ~x3 /\ (x1 \/ ~x2 \/ x3) /\ (~x5 \/ x4).

```
- before main loop:
  \phi = ~x3 /\ (x1 \/ ~x2 \/ x3) /\ (~x5 \/ x4)
  CSD(\phi) = True
```

```
  - first iteration:
    \phi' = \phi[x1 = True]
          = ~x3 /\ True /\ (~x5 \/ x4)
          = ~x3 /\ (~x5 \/ x4)
    \phi' != True, \phi' != False so call CSD(\phi') [returns True]
    set x1 = True, \phi = \phi'

  - second iteration:
    \phi' = \phi[x2 = True]
          = ~x3 /\ (~x5 \/ x4) -- no change!
    \phi' != True, \phi' != False so call CSD(\phi') [returns True]
    set x2 = True, \phi = \phi'

  - third iteration:
    \phi' = \phi[x3 = True]
          = False /\ (~x5 \/ x4)
          = False
    set x3 = False,
    \phi  = \phi[x3 = False]
          = True /\ (~x5 \/ x4)
          = ~x5 \/ x4

  - fourth iteration:
    \phi' = \phi[x4 = True]
       = ~x5 \/ True
       = True
    set x4 = True and x5 = True and terminate

Final assignment:
x1 = True, x2 = True, x3 = False, x4 = True, x5 = True.
```

Q: Now that we have the algorithm, what's next?
A: Argue correctness and analyze runtime.

Q: Why is the algorithm correct?
A: "\phi is satisfiable" is a loop invariant:
  - The first line ensures it's true at the start,
  - If \phi is satisfiable, either \phi[x = True] or \phi[x = False] (or maybe both) must be satisfiable. The algorithm picks the first one for which this is the case (using calls to CSD to check).

Q: What's the runtime?
A: The algorithm makes at most one call to CSD for each variable in \phi, plus the initial call. The time to simplify \phi[x = True] and, in the worst-case, \phi[x = False] is polynomial (no more than quadratic in the length of \phi). Hence, if T(n) represents the worst-case time of CSD on inputs of size n, the worst-case time of CSS is O(n^2 + n T(n)) and this is polynomial if T(n) is polynomial.