Rewriting in ACL2

When you prove a theorem(lemma) in ACL2 using `defthm`, it gets added into the ACL2 logical world(think of it as a database of things ACL2 can use to prove more theorems), as a rewrite rule. Why? Simply because, ACL2 Theorem prover applies the *substitute Equals for Equals*(along with *Instantiation*) rule of inference using these rewrite(transformation) rules. Remember that in a proof, we start with a formula we want to show valid, and we successively apply the above rules of inference and of course Prop. Ded. to finally obtain `t`, thus completing the proof. This series of transformations are mostly done by ACL2 theorem prover using **rewriting**. So it is essential that you understand how ACL2 rewriter works, if you want to understand how ACL2 theorem prover works. Note that the theorems you prove previously affect the rewriter, and to develop a successfult proof strategy one must carefully choose the lemmas one wants to prove, since some lemmas give good rewrite rules and some dont.

Given an expression(term), here is how the ACL2 rewriter works on it:
Variables and constants are simply returned without any change.
Otherwise, the expression is a function application, lets say it is $(f\ a_1\ a_2 \ldots a_n)$. In most[1] cases it rewrites each argument a_i to get a'_i, and then **applies rewrite rules** to $(f\ a'_1 \ldots a'_n)$. Associated with each function symbol `f` is stored a list of rewrite rules that are derived(shown below) from axioms, and theorems. These rules are stored in reverse chronological order i.e the most recently proved theorem is the first one in the list. The rules are tried one by one and the first one that **matches/fires** produces the result.

A rewrite rule for `f` may be derived from a theorem of the form:

```
(implies (and hyp1 ... hypk)
         (equal (f  b1 ... bn )
                rhs))
```

Continuing our explanation of how the rewriter works: The following shows what it means to "apply rewrite rules" to $(f a'_1 \ldots a'_n)$. Lets say the rewriter is searching through the stored list of rules, and is currently considering wether the above rule can be used to rewrite i.e does it *fire*.

[1] In case `f` is `if`, then the test a_1 is rewritten to a'_1, and then a_2 and/or a_3 are rewritten, depending on wether we can establish $a1_1$ is nil.

For a rewrite rule to **fire**, two conditions must be satisfied:

1. A substitution σ should exist such that

   $$(f\, b_1 \ldots b_n)|_\sigma =_s (f\, a'_1 \ldots a'_n)$$

   That is the instantiation of the conclusion should be syntactically equal to the expression to be rewritten. In this case, we say the rule **matches**. Lets say the instantiated rule is:

   ```
   (implies (and hyp'_1 ... hyp'_k)
            (equal (f a'_1 ... a'_n)
                   rhs'))
   ```

2. To apply the instantiated rule the rewriter must relieve[2] its hypotheses. To do so, rewriting is used recursively to establish each hypothesis in turn. This is called *backchaining*. If all hypotheses are established. the rewriter then recursively rewrites rhs' to get rhs''. So the final answer of rewriting $(f\, a_1 \ldots a_n)$ would be rhs''.

   So when does a rule fire?
   It fires, when it matches *and* its conditions are satisfied.

   See the example in the next page, to understand what I just told you, for now dont worry about relieving the hypotheses, just concentrate on understanding how the terms are rewritten inside out and how rules are matched in reverse chronological order.

   EXAMPLE ON NEXT PAGE:

[2]Recall the analogy I had with the induction hypothesis conclusion being under a lock and to use the derived context to unlock it, which is the same as saying relieving the hypothesis

Example:
Suppose you just completed a session with ACL2s where you proved theorems leading to the following rewrite rules.

```
1. (f (f x)) = (g x x)
2. (f (g (g x y) z)) = x
3. (g x y) = (h y)
4. (g x y) = (h x)
5. (g (h x) y) = (f x)
6. (f (g x y)) = (g x y)
```

Suppose further that these rewrite rules were admitted in the order given above (that is, 1 was admitted first, then 2, then 3, then 4, then 5, then 6).

Show all steps in rewriting:

`(f (g (g a b) c))`

There are two rules, you should remember above all:

(a) Rules are applied in *reverse chronological order*.

(b) Rewriting is done *inside out*, from left to right.

To understand the recursive nature of the rewriter in the steps I am going to show you, keep in mind the following notation. Notation: I will show in overline, the $\overline{term}$ being rewritten and on its way **inside**, and when rewriter is on its way **out** and has to "apply rewrite rules", or rewrite constants/vars , I will show the $\underbrace{term}$ with an underbrace, and the partial result of the rewriting I will show in bold.

$\overline{(f\ (g\ (g\ a\ b)\ c))}$

$(f\ \overline{(g\ (g\ a\ b)\ c)})$ {`Go Inside`}

$(f\ (g\ \overline{(g\ a\ b)}\ c))$ {`Go Inside`}

$(f\ (g\ (g\ \overline{a}\ b)\ c))$ {`Go Inside`}

$(f\ (g\ (g\ \underbrace{a}\ b)\ c))$ {`Nothing inside a`}

$(f\ (g\ (g\ \mathbf{a}\ b)\ c))$ {Constants/Vars are simply returned}

$(f\ (g\ (g\ a\ \overline{b})\ c))$ {Left to right(i.e after rewriting a go to b)}

$(f\ (g\ (g\ a\ \underbrace{b}\ )\ c))$ {nothing inside b}

$(f\ (g\ (g\ a\ \mathbf{b})\ c))$ {Constants/Vars simply returned}

$(f\ (g\ \underbrace{(g\ a\ b)}\ c))$ {Rewriter on its way out}

$(f\ (g\ (\mathbf{h}\ \mathbf{a})\ c))$ {Rule 4 matches and applied(replace lhs with rhs) }

$(f\ (g\ \overline{(h\ a)}\ c))$ {Restart(recursively) rewrite at same position }

$(f\ (g\ (h\ \overline{a})\ c))$ {Go Inside}

$(f\ (g\ (h\ \underbrace{a}\ )\ c))$ {Nothing inside a}

$(f\ (g\ (h\ \mathbf{a})\ c))$ {Constants/Vars returned}

$(f\ (g\ \underbrace{(h\ a)}\ c))$ {On its way out}

$(f\ (g\ (\mathbf{h}\ \mathbf{a})\ c))$ {No rules matched (h a), simple return}

$(f\ (g\ (h\ a)\ \overline{c}))$ { Left to Right}

$(f\ (g\ (h\ a)\ \underbrace{c}\ ))$ { Nothing inside c}

$(f\ (g\ (h\ a)\ \mathbf{c}))$ {Constants/Vars returned}

$(f\ \underbrace{(g\ (h\ a)\ c)})$ {On its way out}

$(f\ (\mathbf{f}\ \mathbf{a}))$ {Rule 5 matches and applied}

$(f\ \overline{(f\ a)})$ {restart rewriting at same position }

$(f\ (f\ \overline{a}))$ {Go Inside out}

$(f\ (f\ \underbrace{a}))$ {Cant go inside anymore}

$(f\ (f\ \mathbf{a}))$ {Constants/Vars returned}

$(f\ \underbrace{(f\ a)})$ {On its way out }

$(f\ (\mathbf{f}\ \mathbf{a}))$ {No rules matched, return it }

$\underbrace{(f\ (f\ a))}$ {On its way out }

$(\mathbf{g}\ \mathbf{a}\ \mathbf{a})$ {Rule 1 matches, and applied }

$\overline{(g\ a\ a)}$ {Restart rewriting at same pos}

$(g\ \overline{a}\ a)$ {Inside out}

$(g\ \underbrace{a}\ a)$ {STOP inside out, now go back}

$(g\ \mathbf{a}\ a)$ {Constants/Vars returned}

$(g\ a\ \overline{a})$ {Left to Right}

$(g\ a\ \underbrace{a})$ {STOP inside out, now go back}

$(g\ a\ \mathbf{a})$ {Constants/Vars returned}

$\underbrace{(g\ a\ a)}$ {On its way out}

$(\mathbf{h}\ \mathbf{a})$ {Rule 4 matches, and applied}

$\overline{(h\ a)}$ {Restart rewriting at same pos}

$(h\ \overline{a})$ {Inside out}

$(h\ \underbrace{a})$ {Nothing inside a}

$(h\ \mathbf{a})$ {Consts returned}

$\underbrace{(h\ a)}$ {On its way out}

(**h a**) {No rules match, FINAL ANSWER }

I showed the above steps for pedagogical reasons, to show you how the machine works, of course you dont need to show all these steps, but you should understand it. In the homework, the following solution will give you full points:

Solution:

```
(f (g (g a b) c))

= {Apply Rule 4(i.e Instantiate Lemma 4 }

(f (g (h a) c))

= {Apply Rule 5 }

(f (f a))

= {Apply Rule 1 }

(g a a)

= {Apply Rule 4}

(h a)
```