INF3230 – Mandatory assignment 2

Vegard Fjogstad

Exercise 62:

2. **{f(x) = g(x), g(b) = f(a)}**

First equation: by lpo-1 we have that x=x. To prove that f(x) >lpo g(x), we need to first prove that f(x) >lpo x. Using lpo-1 we can see that f(x) >lpo x.
Because f(x) >lpo x, we can use lpo-2 to prove that f(x) >lpo g(x), given precedence f > g.

Second equation: g(b) = f(a). g and f are different functions, so we cannot use lpo-3. We will instead use lpo-1 from this step to get b > f(a). Using lpo-2 we have that b >lpo a. For the second equation to hold, b > f must be true.

Combining the precedence from equation 1 with equation 2, we get the following precedence which holds: b > f > g > a

3. **{f(g(x)) = g(f(x))}**

We start with lpo-1, giving us x=x. We show that f(x) >lpo x by using lpo-1. This means that using lpo-2, we show that f(x) >lpo g(x), given precedence f > g.
Because f and g are different functions, we cannot use lpo-3 to prove f(g(x)) >lpo g(f(x)).
We will instead use lpo-2. The assumptions for lpo-2 to hold is f(g(x)) >lpo f(x). We use lpo-3 to prove the assumptions: Lpo-3s assumptions are g(x) > lex lpo (x), which we can show by using lpo-1 g(x) >lpo x, and f(g(x)) >lpo x. The last assumption is proven with lpo-1, because g(x) > lpo x already has been shown to hold.

4. **{g(f(x)) = f(g(x))}**

This problem is basically the same as the one above, the functions f and g have just swapped places. We can use the same logic to get our answer.
Lpo-1 for x=x. Then we show that f(x) >lpo x and g(x) >lpo x by using lpo-1. We can then show that g(x) >lpo f(x) given precedence g > f.
g(f(x)) >lpo f(g(x)):
lpo2->g(f(x))>lpo g(x) → lpo-3: f(x) > lex lpo (x) → lpo-1: f(x) >lpo x
g(f(x)) >lpo x → lpo-1 because f(x) >lpo x holds.

Eksamensoppgave 1 i INF2022 fra 2002

1b.
Eks is well founded when there is no infinite sequences like: *t >eks t' >eks t'' >eks* ..
The definition of a strict partial ordering says that > (don't have the correct symbol in libreoffice) is well founded over a set S if there is no infinite sequence s1 > s2 > s2 > ... of S-elements s1, s2, s3, ...
Eks is an extension of lex. We can express all properties of the >eks relation over terms as properties of >lex over tuples of natural number, i.e w(u) >lex w(b). This is helpful because by proving that an infinite sequence would reach a contradiction with lex, we have proven the same for eks.

The "number of symbols" decreases for each simplification step of >ex, because that is how >ex is defined. We compare this with how >ex is defined through lex. That is, the amount of tuples over natural numbers. Given a precedence, we can compare the values in these specifications with >lex to see who is greater then who. The left hand side is always greater than the right hand side, but the right hand side can never be smaller than 0.
This means that we can not have an infinite derivation w(t) -> w(t') -> w(t'') -> ... because it would give an infinite sequence w(t) >lex w(t') >lex w(t'') >lex etc, which is impossible.
This proves that eks is well founded, because it means that there cannot be an infinite sequence t >eks t' >eks t'' >eks … .

1c.
Eks is a simplification ordering. We can prove this by showing that the term is: *transitive, anti-reflexive, monotonic and subterm-property.* As shown above, eks can be represented using >lex. We will use this to prove the first two points.
Eks is transitive. We show this using >eks writtens a tuples of >lex.
Say we have the terms t, u, v. We write >eks with the help of function w, which is simply the tuples over natural numbers. (Example: precedence f > g > a and term f(g(f,a)) would give w(2,1,1))
To show transitivity, it means that if t >eks u and u >eks v, then t >eks v. In terms of lex, this is w(t) >lex w(u), w(u) >lex w(v) implies w(t) >lex w(v).
Instead of writing the terms as w(t), w(u) and w(v), we swap it with A, B and C. This gives us:
If and only if A >lex B, B >lex C then A >lex C.
Because we know that >lex is transitive, and we see that >eks can be rewritten as terms for lex, we have proven that >eks is transitive.

Eks is anti-reflexive, because by using the definition of >lex we see that both are not reflexive. w(a) >lex w(a) does not hold for, given the properties of lexicographic order. Because of this, we know that >eks is not reflexive aswell.

Eks is monotonic. We show this with the definition of >lex. Using a weight function w we can show that >lex is monotonic if: w(t) >lex w(u) implies that w(f(..., t, ...)) >lex w(f(..., u,...)). A weight function could for example be the amount of tuples in a specification. By >lex definition, we know that if t >lex u, then f(t) >lex f(u) also is correct. Lets show why with an example:
f > g > a > b
t = f(a, b, g(a, a)), u = (g(a, g(b, b)) => w(1, 1, 3, 1) >lex w(0, 2, 1, 2), which means that t >lex u

g(f(a,a,a), t) >lex g(f(a,a,a), u) = > w(1, 1, 3, 1) + w(1, 1, 3, 0) >lex w(0, 2, 1, 2) + w(1, 1, 3, 0)
=> w(2, 2, 6, 1) >lex w(1, 3, 4, 2)
Here, f is the part we add to t and u (1, 1, 3, 0). We can see that when we add the same to both sides,

there's no change in the outcome. This means that f(t) >lex f(u) is correct.

In the general case we could change our parentheses to A, B and C. Let's say that w(1, 1, 3, 1) = A, w(0, 2, 1, 2) = B and w(1, 1, 3, 0) = C.
Because >lex makes sense for natural numbers, if A, B and C are natural numbers then A + C >lex B + C holds if A > lex B. Equation logic says that we can discard C from both sides of the equation because it does not affect the outcome. Seeing how A, B and C are simply >eks rewritten for >lex, we have proven that >eks is monotone.

Eks satisfies the subterm-property. We can prove this because we've already proven that it is monotonic, i.e. a >lex b implies f(a) >lex f(b). Because f(a) >lex f(b), we know that f(a) >lex a also holds, as this is implied when a relation is monotonic. Because eks can be written as lex for tuples over natural numbers w(a), we know that eks also satisfies the subterm property.

1d.

1. We would have to compare the variables that could occur on both the left and right-hand side of the >eks. We do this by comparing the variables symbolic value. If their values are the same, we can simply discard them. (Example: Both sides contains one x variable. We discard it, and only look at the remaining symbols.) If one side contains more of the same variable then the other, then we store this number in case both sides has equal value after checking the symbols. (This could be a weight function, Example: Left hand side contains 2 x's, right hand side only contain 1. We then store the number 2 on the left hand side.) This coupled with checking the "normal" symbols occurrence on both sides, we can check the precedence and if left hand side has a higher "value" than the right hand side, it is terminating. If both sides contain different variables, or a variable is only on one side then the system can still terminate because the given precedence.
2. Using the precedence *b f g a* the specifications are terminating because *f(x) >eks g(x)* holds and *g(b) >eks f(a)* holds. We discard the x variable on both sides using the rules above.

1e.
We need to find an example where >eks can be used to prove termination, but not with >lpo. A way to do this would be to use mpo.

Exercise 74

1. Find terms r1 and r2 such that *{f(g(x)) = r1, g(h(x)) = r2}* is confluent (and terminating).

The way we do this is by finding an r1 and r2 that we can prove local confluence and termination. If both these demands are met, the specification is confluent. We prove local confluence by checking if all critical pairs can be joined. We prove termination by either lpo or weight functions. Proving confluence by showing local confluence and terminations is also known as the Newman's lemma.

The overlap term for the left hand sides are f(g(h(x))). We use this to find r1 = f(x) and r2 = h(x), giving us the specification *{f(g(x)) = f(x), g(h(x)) = h(x)}*. Let's prove this.

Using this, we achieve the following critical pairs:
f(g(h(x))) → Equation 1 → f(h(x))
f(g(h(x))) → Equation 2 → f(h(x))
We see that they share a common term, and the specification is therefore locally confluent.

We prove termination using lpo.

Left hand equation:
Using lpo-1 we get x=x. From this, using lpo-1 we derive that f(x) >lpo x, and finally using lpo-2 we have that f(x) > g(x), with precedence f > g.
To complete the left hand side, we need to look at the whole term. f(g(x)) = f(x). We can use lpo-3 here, because both sides have the same function symbol. Because g(x) >(lpo lex) x and f(g(x)) >lpo x is correct, we can use lpo-3 to show that f(g(x)) >lpo f(x).

Right hand equation:
Lpo-1 for x=x. We use lpo-2 to show that g(x) > h(x), with precedence g > h.

We cannot use the same logic for the right hand side, because g(h(x)) = h(x) are different function symbols. To prove g(h(x)) >lpo h(x), we have to use lpo-2. We've already set precedence g >h, so we need to show that g(h(x)) >lpo x. We can use lpo-1 to prove that g(h(x)) > lpo x. This means that h(x) >lpo x has to hold, which it does using lpo-1.

We have therefore shown that the system is terminating.

We combine the right hand side with the left, and get the following precedence: f > g > h.
We have proved that the specification is terminating, and using Newman's lemma, it's also confluent.

Exercise 1 Confluence

1.

{f(a) = b, f(f(x)) = x}

a and *b* are constants, whereas *x* is a variable. We can not rename these to match the second specification, meaning the specification is not confluent. We prove this by showing that it's either non terminating or locally confluent. First we find the critical pairs.

Equations do not share a variable, so we do not have to rename x to x'.

Equation 1: f(a) = b

Equation 2: f(f(x)) = x

The terms overlap on f. We have to check the overlap term f(f(a)).

$f(f(x)) |_1 f(a)$:

"Equation" 3: mgu {x |-> a}

Now we use these equations to show that we cannot reduce the critical pairs to the same term. We obtain the critical pairs by applying equation 1 and 2 to the overlap term.

f(f(a)) → equation 1 → f(b)

f(f(a)) → equation 2 → a

The critical pairs *{f(b), a}* share no common term, which means that the specification is not locally confluent. It's therefore not confluent.

2.

By adding the equation f(b) = a, the system would be confluent. We can show this by changing the substitutions for the above critical pairs.

f(f(a)) → equation 1 → f(b) → a

f(f(a)) → equation 2 → a

The critical pairs now reduce to the same term, and the specification is therefore locally confluent.

Now we need to prove that the specification is terminating. We can do this by using lpo.

Let's start with the first equation, *{f(a) = b}*:

a and b are constants. The equation is terminating, and we don't need lpo to prove this.

Second equation: *{f(f(x)) = x}*

To prove that f(f(x)) = x is terminating, we use lpo-1. This means that f(x) >lpo x has to hold, which it does because x = x, as shown by lpo-1.

Last equation: *{f(b) = a}*

This equation is also terminating, because both b and a are constants.

This proves that the specification is confluent by Newman's lemma. There's also no conflicts with the previous equations.

New specification: *{f(a) = b, f(f(x) = x, f(b) = a}*