Proof 2

October 18, 2013

In this document we present a proof of

$$
\begin{aligned}
INDXS = \ & \forall A : \texttt{Type}. \\
& \quad \forall x : A. \\
& \quad\quad \forall lst : \texttt{List}(A). \\
& \quad\quad\quad \exists indxs : \texttt{List}(\mathbb{N}). \\
& \quad\quad\quad\quad \forall i : \mathbb{N}. \quad i \in_{\mathbb{N}} indxs \\
& \quad\quad\quad\quad\quad\quad \Longleftrightarrow \\
& \quad\quad\quad\quad\quad\quad i < ||lst|| \ \wedge \ x = lst[i] \in A
\end{aligned}
$$

that generates the following program:

```
let rec indexes' x lst n =
  match lst with
    [] -> []
  | h :: t ->
     let l = indexes' x t (n + 1)
     in if x = h then n :: l else l ;;
let indexes x lst = indexes' x lst 0 ;;
```

First we move all our universally quantified variables to our context:

```
● X -- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top
[-]
∀[A:Type]. ∀deq:EqDecider(A). ∀x:A. ∀L:A List.  (∃indxs:{ℕ List| (∀i:ℕ. ((i ∈ indxs) ⟺ (i < ||L||) ∧ (L[i] = x)))})

# BY (UnivCD THENA Auto)

# 1

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List@i
[-]
⊢ ∃indxs:{ℕ List| (∀i:ℕ. ((i ∈ indxs) ⟺ (i < ||L||) ∧ (L[i] = x)))}

# BY
```

Then, we prove the following property instead. Note that the {`n...`} which is the set of natural numbers greater or equal to `n` is necessary to prove that this assertion is true.

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List@i
[-]
⊢ ∃indxs:{ℕ List| (∀i:ℕ. ((i ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))}

# BY Assert ⌈∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))})⌉⋅

# 1 1
.....assertion.....
[-]
⊢ ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))})

# BY

# 1 2

[+],
[-]
⊢ ∃indxs:{ℕ List| (∀i:ℕ. ((i ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))}

# BY
```

We prove that our assertion implies our original goal. This corresponds to making the call `indexes' x lst 0`.

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 2

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List@i
5. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))})
[-]
⊢ ∃indxs:{ℕ List| (∀i:ℕ. ((i ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))}

* BY ((InstHyp ['0'] (-1)⋅ THENA Auto) THEN Repeat (ParallelLast) THEN Auto')
```

As in our first proof, we prove this new formula by induction on `L`:

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1 1
.....assertion.....
[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List@i
[-]
⊢ ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||L||) ∧ (L[i] = x)))})

# BY ((ListInd (-1) THEN (D 0 THENA Auto)) THEN Reduce 0)

# 1 1 1

[+]n,
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < 0) ∧ (bottom() = x)))}

# BY

# 1 1 2

[+]u,v,,n,
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY
```

As in our first proof, the base case is trivial:

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 1

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List
5. n : ℕ@i
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < 0) ∧ (bottom() = x)))}

* BY (With `[]' (D 0)· THEN Auto THEN Auto')
```

In the induction case, we first compute the recursive call `indexes' x t (n + 1)`. This gives us a list `indxs` called `l` in `indexes'`'s definition

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1 1 2

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List
5. u : A
6. v : A List
7. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x)))})
8. n : ℕ@i
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY ((InstHyp ['n + 1'] (-2)· THENA Auto) THEN ExRepD)

# 1 1 2 1

[+]indxs,,
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY
```

As in our first proof, we check whether `x` is equal to `u`, the head of our list `L`. We obtain two subgoals. In the first branch, we get to assume that `x` and `u` are equal and in the other branch we get to assume that they are different.

```
X -- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1 1 2 1

[-]
1. [A] : Type
2. deq : EqDecider(A)@i
3. x : A@i
4. L : A List
5. u : A
6. v : A List
7. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x)))})
8. n : ℕ@i
9. indxs : {n + 1...} List
10. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x))
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY (D 2 THEN (VrBoolCase ⌜deq x u⌝· THENA Auto))

# 1 1 2 1 1

[+]deq,
[+],x,L,u,v,,n,indxs,,,
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY

# 1 1 2 1 2

[+]deq,
[+],x,L,u,v,,n,indxs,,,
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY
```

If `x` is equal to `u` then the list of indexes is `n :: indxs`. After doing some simple reasoning, we are left with two simple subgoals (see the two next figures).

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1 1 2 1 1

[+]A,deq,
[-]
3. \\%6 : ∀x,y:A.  (x = y ⟺ ↑(deq x y))@i
4. x : A@i
5. L : A List
6. u : A
7. v : A List
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x))
12. ↑(deq x u)
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY (InstConcl [`[n / indxs]`]· THEN Auto THEN SpReasoner THEN Decide `i = 0`· THEN Auto)

# 1 1 2 1 1 1

[+],,
[+]i,,,
[-]
⊢ i < (||v|| + 1)

# BY

# 1 1 2 1 1 2

[+],,
[+]i,,,,,
[-]
⊢ ((i + n) = n) ∨ (i + n ∈ indxs)

# BY
```

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 2 1 1 1

[+]A,deq,,x,L,
[-]
6. u : A
7. v : A List
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⟺ (i < ||v||) ∧ (↑(deq v[i] x)))
12. ↑(deq x u)
13. i : ℕ@i
14. (i + n ∈ indxs)
15. ¬(i = 0)
[-]
⊢ i < (||v|| + 1)

* BY ((InstHyp [`i - 1`] (-5)· THENA Auto)
      THEN (Subst `(i - 1) + n + 1 ~ i + n` (-1)· THENA Auto)
      THEN RWO "-1" (-3)
      THEN Auto)
```

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 2 1 1 2

[+]A,deq,,x,L,u,v,
[-]
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⟺ (i < ||v||) ∧ (↑(deq v[i] x)))
12. ↑(deq x u)
13. i : ℕ@i
14. i < (||v|| + 1)@i
15. ↑(deq v[i - 1] x)
16. ¬(i ≤ 0)
17. ¬(i = 0)
[-]
⊢ ((i + n) = n) ∨ (i + n ∈ indxs)

* BY ((OrRight THENA Auto)
      THEN (InstHyp [⌜i - 1⌝] (-7)⋅ THENA Auto)
      THEN (Subst ⌜(i - 1) + n + 1 ~ i + n⌝ (-1)⋅ THENA Auto)
      THEN RWO "-1" 0
      THEN Auto)
```

If `x` is not equal to `u` then the list of indexes is `indxs`. After doing some simple reasoning, we are now left with three simple subgoals (see the two three figures). The first and third subgoals are trivial. Let us look at the second one.

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
# top 1 1 2 1 2

[+]A,deq,
[-]
3. \\%6 : ∀x,y:A.  (x = y ⟺ ↑(deq x y))@i
4. x : A@i
5. L : A List
6. u : A
7. v : A List
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⟺ (i < ||v||) ∧ (v[i] = x))
12. ¬↑(deq x u)
[-]
⊢ ∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⟺ (i < (||v|| + 1)) ∧ ([u / v][i] = x)))}

# BY (InstConcl ["indxs"]· THEN Auto THEN SpReasoner THEN Decide "i = 0"· THEN Auto)

# 1 1 2 1 2 1

[+],,
[+]i,,,
[-]
⊢ i < (||v|| + 1)

# BY

# 1 1 2 1 2 2

[+],,
[+]i,,,,
[-]
⊢ u = x

# BY

# 1 1 2 1 2 3

[+],,
[+]i,,,,,
[-]
⊢ (i + n ∈ indxs)

# BY
```

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 2 1 2 1

[+]A,deq,,x,L,
[-]
6. u : A
7. v : A List
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⇐⇒ (i < ||v||) ∧ (↑(deq v[i] x)))@i
12. ¬↑(deq x u)
13. i : ℕ@i
14. (i + n ∈ indxs)@i
15. ¬(i = 0)
[-]
⊢ i < (||v|| + 1)

* BY ((InstHyp [⌜i - 1⌝] (-5)⋅ THENA Auto)
      THEN (Subst ⌜(i - 1) + n + 1 ~ i + n⌝ (-1)⋅ THENA Auto)
      THEN RWO "-1" (-3)
      THEN Auto)
```

In this subgoal we have to prove that `x` is equal to `u`, assuming that `x` is different from `u`. We can prove this, because it turns our that we can get a contradiction from our list of hypotheses. Hypothesis 14 says that `i + n` is in the list `indxs`. We also have that `i` is 0. Therefore `n` is in the list `indxs`. This is not possible because `indxs` is the list of positions of `x` in the tail of our list, which was obtained by applying `indexes'` to `n + 1`, and `n` must be greater or equal to `n + 1` (see hypothesis 10).

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 2 1 2 2

[+]A,deq,,x,L,u,
[-]
7. v : A List
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⇐⇒ (i < ||v||) ∧ (↑(deq v[i] x)))
12. ¬↑(deq x u)
13. i : ℕ@i
14. (i + n ∈ indxs)@i
15. i ≤ 0
16. i = 0
[-]
⊢ u = x

* BY ((Subst ⌜i + n ~ n⌝ (-3)⋅ THENA Auto')
      THEN RepeatFor 2 (D (-3))
      THEN MoveToConcl (-3)
      THEN GenConclAtAddr [1;3]
      THEN Auto')
```

```
-- PRF : vr_test_indx3-1 @edd.markb-al @gateway
* top 1 1 2 1 2 3

[+]A,deq,,x,L,u,v,
[-]
8. ∀n:ℕ. (∃indxs:{{n...} List| (∀i:ℕ. ((i + n ∈ indxs) ⇐⇒ (i < ||v||) ∧ (v[i] = x)))})
9. n : ℕ@i
10. indxs : {n + 1...} List
11. ∀i:ℕ. ((i + n + 1 ∈ indxs) ⇐⇒ (i < ||v||) ∧ (↑(deq v[i] x)))
12. ¬↑(deq x u)
13. i : ℕ@i
14. i < (||v|| + 1)@i
15. ↑(deq v[i - 1] x)
16. ¬(i ≤ 0)
17. ¬(i = 0)
[-]
⊢ (i + n ∈ indxs)

* BY ((InstHyp [⌜i - 1⌝] (-7)· THENA Auto)
      THEN (Subst ⌜(i - 1) + n + 1 ~ i + n⌝ (-1)· THENA Auto)
      THEN RWO "-1" 0
      THEN Auto)
```