```
trace_inv_induction  THM

⊢  ∀M:sm{i:l}(), I:(M.state→(M.action List)→Prop).
|  (∀x:M.state. M.init(x) ⇒ I(x,nil)) ⇒
|  (∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
|   M.init(s0) ⇒
|   trace_reachable(M;s0;l;x) ⇒
|   I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act]))
|  ⇒
|  (M |= always s,t.I(s,t))  by Auto THEN Unfold `trace_inv` 0
|
1. M : sm{i:l}()
2. I : M.state→(M.action List)→Prop
3. ∀x:M.state. M.init(x) ⇒ I(x,nil)
4. ∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
|  M.init(s0) ⇒
|  trace_reachable(M;s0;l;x) ⇒
|  I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act])
⊢  ∀t:M.action List, s0,s:M.state.
|  M.init(s0) ⇒ trace_reachable(M;s0;t;s) ⇒ I(s,t)
| by BackThruLemma `list_append_singleton_ind` THENA Auto
|
|\
| 1. M : sm{i:l}()
| 2. I : M.state→(M.action List)→Prop
| 3. ∀x:M.state. M.init(x) ⇒ I(x,nil)
| 4. ∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
|    M.init(s0) ⇒
|    trace_reachable(M;s0;l;x) ⇒
|    I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act])
| ⊢  ∀s0,s:M.state.
|    M.init(s0) ⇒ trace_reachable(M;s0;nil;s) ⇒ I(s,nil)
|   by RecUnfold `trace_reachable` 0 THEN Reduce 0 THEN Auto
|
|        THEN RevHypSubst (-1) 0 THENA Auto THEN Easy
|
.\
. 1. M : sm{i:l}()
. 2. I : M.state→(M.action List)→Prop
. 3. ∀x:M.state. M.init(x) ⇒ I(x,nil)
. 4. ∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
. |  M.init(s0) ⇒
. |  trace_reachable(M;s0;l;x) ⇒
. |  I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act])
. ⊢  ∀ys:M.action List, x:M.action.
. |  (∀s0,s:M.state.
. |   M.init(s0) ⇒ trace_reachable(M;s0;ys;s) ⇒ I(s,ys))
. |  ⇒
. |
. |  (∀s0,s:M.state.
. |   M.init(s0) ⇒ trace_reachable(M;s0;ys @ [x];s) ⇒ I(s,ys @ [x]))
. | by Auto
. |
. 1. M : sm{i:l}()
. 2. I : M.state→(M.action List)→Prop
```

```
. 3. ∀x:M.state. M.init(x) ⇒ I(x,nil)
. 4. ∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
. |  M.init(s0) ⇒
. |  trace_reachable(M;s0;l;x) ⇒
. |  I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act])
. 5. ys : M.action List
. 6. x : M.action
. 7. ∀s0,s:M.state.
. |  M.init(s0) ⇒ trace_reachable(M;s0;ys;s) ⇒ I(s,ys)
. 8. s0 : M.state
. 9. s : M.state
. 10. M.init(s0)
. 11. trace_reachable(M;s0;ys @ [x];s)
. ⊢  I(s,ys @ [x])
. | by RWW "trace_reachable_append" (-1) THENA Auto THEN ExRepD
.
. |    ...
. THEN RecUnfold `trace_reachable` (-1) THEN Reduce (-1)
.
. |    THEN RevHypSubst (-1) 0 THENA Auto
. |
. 1. M : sm{i:l}()
. 2. I : M.state→(M.action List)→Prop
. 3. ∀x:M.state. M.init(x) ⇒ I(x,nil)
. 4. ∀s0,x:M.state, act:M.action, x':M.state, l:M.action List.
.    M.init(s0) ⇒
.    trace_reachable(M;s0;l;x) ⇒
.    I(x,l) ⇒ M.trans(x,act,x') ⇒ I(x',l @ [act])
. 5. ys : M.action List
. 6. x : M.action
. 7. ∀s0,s:M.state.
.    M.init(s0) ⇒ trace_reachable(M;s0;ys;s) ⇒ I(s,ys)
. 8. s0 : M.state
. 9. s : M.state
. 10. M.init(s0)
. 11. x1 : M.state
. 12. trace_reachable(M;s0;ys;x1)
. 13. x@0 : M.state
. 14. M.trans(x1,x,x@0)
. 15. x@0 = s
. ⊢  I(x@0,ys @ [x])
.   by AllHyps (InstHyp [s0;x1;x;x@0;ys]) THEN Auto THEN Easy
END
```