Expander2, a Haskell-based prover and rewriter

fldit-www.cs.uni-dortmund.de/~peter/Expander2.html

Peter Padawitz
TU Dortmund

7. Oktober 2011

1. Expander2 components 2
2. O’Haskell types 4
3. Non-simplifying inference rules 9
4. Examples 18
Expander2 components
O’Haskell types

Data types

data Datatype = constructor1 type11 ... type1n1 |
              constructor2 type21 ... type2n2 |
              ...

a = constructor1 term11 ... term1n1
b = constructor2 term21 ... term2n2
Records

struct Record = selector1 :: type1 -> type1'
selector2 :: type2 -> type2'

record = struct selector1 t1 = term1 (non-recursive)
        selector2 t2 = term2 (non-recursive)

oder

record = struct selector1 = selector1
        selector2 = selector2
        where selector1 t1 = term1 (recursive)
        selector2 t2 = term2 (recursive)

a = record.selector1
b = record.selector2
Subtyping

\[
\text{struct } \text{RecordS} < \text{Record} = \text{selectorS1} :: \text{typeS1} \\
\text{selectorS2} :: \text{typeS2}
\]

\[
\text{Action} < \text{Cmd} ()
\]

\[
\text{Request a} < \text{Cmd} a
\]

\[
\text{Template a} < \text{Cmd} a
\]

\[
\text{struct } \text{Methods} = \text{method1} :: \text{type11} \ldots \text{type1n1} \rightarrow \text{Action} \\
\text{method2} :: \text{type21} \ldots \text{type2n2} \rightarrow \text{Request type2}
\]

Supertyping

\[
\text{data } \text{DatatypeS} > \text{Datatype} = \text{constructorS1} \text{ typeS11} \ldots \text{typeS1nS1} | \\
\text{constructorS2} \text{ typeS21} \ldots \text{typeS2nS2} |
\]
Object classes (templates)

class :: type1 -> type2 -> ... -> Template Methods

class x1 x2 ... = template stateVar1 := term1
                stateVar2 := term2
        in struct method1 = action monad_term1 (non-recursive)
                        method2 = request monad_term2 (non-recursive)
                where <local definitions>

oder

class x1 x2 ... = template stateVar1 := term1
                stateVar2 := term2
        in let <local definitions including
        recursive actions or requests>
            method1 = action monad_term1 (recursive)
            method2 = request monad_term2 (recursive)
        in struct ..Methods
                where <local definitions>

a <- class a1 a2 ...
Main program of Expander2

module Ecom where

import Tk

main tk = do
    win1 <- tk.window []
    win2 <- tk.window []
    fix solve1 <- solver tk "Solver1" win1 solve2 "Solver2" enum1 paint1
    solve2 <- solver tk "Solver2" win2 solve1 "Solver1" enum2 paint2
    paint1 <- painter tk "Solver1" solve1 "Solver2" solve2
    paint2 <- painter tk "Solver2" solve2 "Solver1" solve1
    enum1 <- enumerator tk solve1
    enum2 <- enumerator tk solve2
    solve1.buildSolve (0,20) solve1.skip
    solve2.buildSolve (20,20) solve1.skip
    win2.iconify
Non-simplifying inference rules

**Resolution** Let \( p \) be a least predicate. \( AX_p \) is applied to an atom \( pt \):

\[
\frac{pt}{\bigvee_{i=1}^{k} \exists Z_i : (\varphi_i \sigma_i \land \vec{x} = \vec{x}\sigma_i)}
\]

where \( AX_p = \{ pt_1 \leftarrow \varphi_1, \ldots, pt_n \leftarrow \varphi_n \} \),

\((*)\) \( \vec{x} \) is a list of the variables of \( t \),

for all \( 1 \leq i \leq k \), \( t \sigma_i = t_i \sigma_i \) and \( Z_i = \text{var}(t_i, \varphi_i) \),

for all \( k < i \leq n \), \( t \) is not unifiable with \( t_i \).

**Coresolution** Let \( p \) be a greatest predicate. \( AX_p \) is applied to an atom \( pt \):

\[
\frac{pt}{\bigwedge_{i=1}^{k} \forall Z_i : (\varphi_i \sigma_i \lor \vec{x} \neq \vec{x}\sigma_i)}
\]

where \( AX_p = \{ pt_1 \Rightarrow \varphi_1, \ldots, pt_n \Rightarrow \varphi_n \} \) and \((*)\) holds true.
Deterministic narrowing

Let $f$ be a defined function. $AX_f$ is applied to a $\Sigma$-operation $ft$:

$$
\begin{align*}
    r(\ldots, ft, \ldots) \\
    \bigvee_{i=1}^{k} \exists Z_i : (r(\ldots, u_i, \ldots)\sigma_i \land \varphi_i \sigma_i \land \vec{x} = \vec{x} \sigma_i) \lor \\
    \bigvee_{i=k+1}^{l} (r(\ldots, ft, \ldots)\sigma_i \land \vec{x} = \vec{x} \sigma_i)
\end{align*}
$$

where $r$ is a predicate,

$AX_f = \{ \gamma_1 \Rightarrow (ft_1 = u_1 \iff \varphi_1), \ldots, \gamma_n \Rightarrow (ft_n = u_n \iff \varphi_n) \}$,

(**) $\vec{x}$ is a list of the variables of $t$,

for all $1 \leq i \leq k$, $t \sigma_i = t_i \sigma_i$, $\gamma_i \sigma_i \vdash True$ and $Z_i = var(t_i, u_i, \varphi_i)$,

for all $k < i \leq l$, $\sigma_i$ is a partial unifier of $t$ and $t_i$,

for all $l < i \leq n$, $t$ is not partially unifiable with $t_i$. 
Nondeterministic narrowing

Let $\rightarrow$ be a transition predicate. $AX\rightarrow$ is applied to an atom $t \land v \rightarrow t'$:

\[
\begin{align*}
\forall_{i=1}^{k} \exists Z_i : ((u_i \land v)\sigma_i = t'\sigma_i \land \varphi_i\sigma_i \land \vec{x} = \vec{x}\sigma_i) \lor \\
\bigvee_{i=k+1}^{l}((t \land v)\sigma_i \rightarrow t'\sigma_i \land \vec{x} = \vec{x}\sigma_i)
\end{align*}
\]

where $AX\rightarrow = \{\gamma_1 \Rightarrow (t_1 \rightarrow u_1 \Leftarrow \varphi_1), \ldots, \gamma_n \Rightarrow (t_n \rightarrow u_n \Leftarrow \varphi_n)\}$, (**) holds true and $\sigma_i$ is a unifier modulo associativity and commutativity of $\land$.

Elimination of irreducible atoms and terms ("negation as failure")

\[
\begin{array}{ccccc}
pt & qt & r(\ldots, ft, \ldots) & t \rightarrow t' \\
False & True & r(\ldots, (), \ldots) & () \rightarrow t'
\end{array}
\]

where $p \neq \rightarrow$ is a least predicate, $q$ is a greatest predicate, $f$ is a defined function and $pt$, $qt$, $ft$ and $t \rightarrow t'$ are irreducible, i.e., none of the above rules is applicable.
Let \( p : e \) be a least predicate of \( P' \) and \( \psi_p : e \) be a \( \Sigma \)-formula that shall be proved to follow from \( p \).

**Predicate induction**  
A goal \( p \Rightarrow \psi_p \) is applied to \( AX_p \):

\[
\begin{align*}
\forall \varphi \in AX (\varphi[\psi_p/p \mid p \in P'] \Rightarrow \psi_p) \\
\end{align*}
\]

**Equality induction = induction upon a function**

\[
\begin{align*}
f(x) = y \Rightarrow \psi_f(x, y) \\
\forall \varphi \in flat(AX_f) (\varphi[\psi_f/(f(\_ \_ \_ ) = \_ \_ \_ )] \Rightarrow \psi_f(t, u)) \\
\end{align*}
\]

Let \( p : e \) be a greatest predicate of \( P' \) and \( \psi_p : e \) be a \( \Sigma \)-formula that shall be proved to imply \( p \).

**Predicate coinduction**  
A goal \( \psi_p \Rightarrow p \) is applied to \( AX_p \):

\[
\begin{align*}
\forall \varphi \in AX (\psi_pt \Rightarrow \varphi[\psi_p/p \mid p \in P']) \\
\end{align*}
\]
Noetherian induction

Select a list of free or universal induction variables $x_1, \ldots, x_n$ in the conjecture

$$\varphi = (\text{prem} \Rightarrow \text{conc}).$$

Then the induction hypotheses

$$\text{conc}' \iff (x_1, \ldots, x_n) \succ (x'_1, \ldots, x'_n) \wedge \text{prem}'$$

$$\text{prem}' \implies ((x_1, \ldots, x_n) \succ (x'_1, \ldots, x'_n) \Rightarrow \text{conc}')$$

are added to the current theorems.

If $\varphi$ is not an implication, then

$$\varphi' \iff (x_1, \ldots, x_n) \succ (x'_1, \ldots, x'_n)$$

is added.

Primed formulas are obtained from unprimed ones by priming the occurrences of $x_1, \ldots, x_n$.

$\succ$ denotes the induction ordering. Each left-to right application of an added theorem corresponds to an induction step and introduces an occurrence of $\succ$.

After axioms for $\succ$ have been added to the current axioms, narrowing steps upon $\succ$ should remove the occurrences of $\succ$ because the transformation is correct only if $\varphi$ can be derived to $\text{True}$. 
Incremental versions of predicate induction and coinduction

Let $p : e$ be a least predicate of $P'$ and $\psi_p : e$ be a $\Sigma$-formula that shall be proved to follow from $p$.

**Predicate induction**

\[\begin{align*}
& (1) \quad \frac{p \Rightarrow \psi_p}{\bigwedge_{pt \models \varphi \in AX} (\varphi[q_p/p \mid p \in P'] \Rightarrow \psi_p t)} \quad q_p \Rightarrow \psi_p \text{ is added to } AX \\
& (2) \quad \frac{q_p \Rightarrow \delta_p}{\bigwedge_{pt \models \varphi \in AX} (\varphi[q_p/p \mid p \in P'] \Rightarrow \delta_p t)} \quad q_p \Rightarrow \delta_p \text{ is added to } AX
\end{align*}\]

The proof starts by adding to $P$ a predicate $q_p$, first for $\psi_p$ and – when the second rule is applied – for a generalization $\psi_p \land \delta_p$ of $\psi_p$.

Between the applications of (1) resp. (2), coresholution steps upon the added axiom $q_p \Rightarrow \psi_p$ must be confined to redex positions with negative polarity, i.e., the number of preceding negation symbols in the entire formula must be odd. Otherwise the axiom added when (2) is applied might violate the soundness of the coresholution steps.

Coresolution upon $q_p$ at any redex position becomes sound as soon as the set of axioms for $q_p$ is not extended any more.
By inferring \textit{True} from the conclusions of (1) and (2) one shows, roughly speaking, that the predicate \( \psi_p \land \delta_p \) solves the axioms for \( p \). Since \( p \) itself represents the least solution, we conclude \( p \Rightarrow \psi_p \land \delta_p \), in particular the original goal \( p \Rightarrow \psi_p \).

Let \( p : e \) be a greatest predicate of \( P' \) and \( \psi_p : e \) be a \( \Sigma \)-formula that shall be proved to imply \( p \).

\textbf{Predicate coinduction}

\[
\begin{align*}
\text{(1)} & \quad \psi_p \Rightarrow p \\
& \quad \land_{pt \Rightarrow \varphi \in AX} (\psi_p t \Rightarrow \varphi[q_p/p \mid p \in P'])
\end{align*}
\]

\( q_p \Leftarrow \psi_p \) and – only if \( p \) denotes a congruence relation – equivalence axioms for \( q_p \) are added to \( AX \)

\[
\begin{align*}
\text{(2)} & \quad \delta_p \Rightarrow q_p \\
& \quad \land_{pt \Rightarrow \varphi \in AX} (\delta_p t \Rightarrow \varphi[q_p/p \mid p \in P'])
\end{align*}
\]

\( q_p \Leftarrow \delta_p \) is added to \( AX \)
The proof starts by adding to $P$ a predicate $q_p$, first for $\psi_p$ and – when the second rule is applied – for a generalization $\psi_p \lor \delta_p$ of $\psi_p$.

Between the applications of (1) resp. (2), resolution steps upon the added axiom $q_p \iff \psi_p$ must be confined to redex positions with positive polarity, i.e., the number of preceding negation symbols in the entire formula must be even. Otherwise the axiom added when (2) is applied might violate the soundness of the resolution steps.

Resolution upon $q_p$ at any redex position becomes sound as soon as the set of axioms for $q_p$ is not extended any more.

By inferring $True$ from the conclusions of (1) and (2) one shows, roughly speaking, that the predicate $\psi_p \lor \delta_p$ (or its equivalence closure if $p$ denotes a congruence relation) solves the axioms for $p$. Since $p$ itself represents the greatest solution, we conclude $\psi_p \lor \delta_p \Rightarrow p$, in particular the original goal $\psi_p \Rightarrow p$. 
Rewriting upon a defined function \( f \)

\[
\begin{align*}
    c(f(t)) \\
    c(u_1\sigma_1) & <++> \ldots <++> c(u_k\sigma_k)
\end{align*}
\]

where \( \gamma_1 \Rightarrow f(t_1) = u_1, \ldots, \gamma_1 \Rightarrow f(t_n) = u_n \) are the axioms for \( f \) and

\((*)\) for all \( 1 \leq i \leq k \), \( t = t_i\sigma_i \) and \( \gamma_i\sigma_i \vdash True \),

for all \( k < i \leq n \), \( t \) does not match \( t_i \).

Rewriting upon the predicate \( \rightarrow \)

\[
\begin{align*}
    c(t) \\
    c(u_1\sigma_1) & <++> \ldots <++> c(u_k\sigma_k)
\end{align*}
\]

where \( \gamma_1 \Rightarrow t_1 \rightarrow u_1, \ldots, \gamma_1 \Rightarrow t_n \rightarrow u_n \) are the axioms for \( \rightarrow \) and \((*)\) holds true.

Elimination of non-rewritable terms

\[
\begin{align*}
    f(t) \\
    ()
\end{align*}
\]

where \( f \) is a defined function, \( t \) is a normal form
and for all axioms \( \gamma \Rightarrow f(u) = v \) and \( \gamma \Rightarrow u \rightarrow v \), \( t \) and \( u \) are not unifiable.
Examples

-- nat

preds: Nat even odd eq neq
defuncts: div fib loop fibL loop1 loop2 sum
fovars: q r n
hovars: f

axioms:

\[
\begin{align*}
  \text{sum}(0) &= 0 \\
  &\& \text{sum}(\text{suc}(x)) = \text{sum}(x) + x + 1 \\
  &\& (x < y \implies \text{div}(x,y) = (0,x)) \\
  &\& (0 < y \&\& y \leq x \&\& \text{div}(x-y,y) = (q,r) \implies \text{div}(x,y) = (\text{suc}(q),r)) \\
  &\& (0 < y \&\& y \leq x \implies \text{div}(x,y) = \text{case}(\text{div}(x-y,y),(q,r),(\text{suc}(q),r))) \\
  &\& \text{fib}(0) = 0 \\
  &\& \text{fib}(1) = 1 \\
  &\& \text{fib}(\text{suc}(\text{suc}(n))) = \text{fib}(n) + \text{fib}(\text{suc}(n)) \\
  &\& (\text{Nat}(0) \iff \text{True})
\end{align*}
\]
& (Nat(suc(x)) \iff Nat(x))
& even(0)
& (even(suc(x)) \implies odd(x))
& (odd(suc(x)) \implies even(x))
& eq(x)(x)
& (x \neq y \implies neq(x)(y))

-- & div(x,y) = loop(y,0,x)
& (loop(y,q,r) = (q,r) \iff r < y)
& (loop(y,q,r) = loop(y,q+1,r-y) \iff r \geq y)
& (INV(x,y,q,r) \iff x = (y*q)+r)

& fibL(n) = loop1(n,0,1)
& loop1(0,x,y) = x
& loop1(suc(n),x,y) = loop1(n,y,x+y)

& loop2(f)(0)(x) == x
& loop2(f)(suc(n))(x) == f(loop2(f)(n)(x))

& suc(x) >> x
& Nat(0)
& (Nat(suc(x)) \iff Nat(x))

-- & (INV(n,x,y,z) \iff n \geq x \& y = fib(n-x) \& z = fib(n-x+1))
& (x >> y \iff x > y)

conjects:

(sum(x) = y \implies x*(x+1) = 2*y) -- sum1
& (div(x,y) = (q,r) \implies x = (y*q)+r \& r < y) -- div
& (x = (y*q)+r \implies loop(y,q,r) = div(x,y)) -- divloop
& (Nat(x) \implies x+y = y+x) -- comm
& (Nat(x) \implies x+(y+z) = (x+y)+z) -- assoc
& (Nat(x) \implies x < 2**x) -- exp
& (Nat(x) \implies even(x) \mid odd(x)) -- evod
& fibL(x) = fib(x) -- fib
& (Nat(x) \implies suc(x)*x = x**2+x) -- pot
& (Nat(n) \implies loop2(f)(n)$f$x = f$loop2(f)(n)(x)) -- natloop

& div(5,4) = x
& div(5,x) = (1,1)
& Any x y:(x < y \& div(5,3)=(x,y))
terms:
fun((suc(x),y),x+x+y)(6,10) <+>
fun((suc(x),y),fun(z,x+y+z)(5))(suc(z),10) <+>
filter(rel(x,x<5))[1,2,3,4,5,6] <+>
filter(rel(x,Int(x)))[1,2,3.6,4,5,6]

-- sum

Derivation of

sum(x) = y ==> (x*(x+1)) = (2*y)

Adding

(sum0(x,y) ===> (x*(x+1)) = (2*y))

to the axioms and applying FIXPOINT INDUCTION wrt

sum(0) = 0
& \text{(sum(suc(x)) = ((z0+x)+1) \iff sum(x) = z0)}

at position [] of the preceding formula leads to

\text{All x z0:((0*(0+1)) = (2*0)) \&
All x z0:((suc(x)*(suc(x)+1)) = (2*((z0+x)+1)) \iff sum0(x,z0)}

SIMPLIFYING the preceding formula (23 steps) leads to

\text{All x z0:(sum0(x,z0) \implies ((x+(x+(x*x)))+x) = ((z0+x)+(z0+x))}

NARROWING the preceding formula (1 step) leads to

\text{All x z0:((x*(x+1)) = (2*z0) \implies ((x+(x+(x*x)))+x) = ((z0+x)+(z0+x)))}

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (3 steps) leads to

True
Number of proof steps: 4

-- NatEvenOdd

Derivation of

Nat(x) ==> even(x) | odd(x)

Adding

(Nat0(x) ==> even(x) | odd(x))

to the axioms and applying FIXPOINT INDUCTION wrt

Nat(0)
& (Nat(suc(x)) <= Nat(x))

at position [] of the preceding formula leads to

All x:(even(0) | odd(0)) & All x:(even(suc(x)) | odd(suc(x)) <= Nat0(x))
NARROWING the preceding formula (1 step) leads to

$$\text{All } x: (\text{True } | \text{odd}(0)) \& \text{All } x: (\text{even}(\text{suc}(x)) \mid \text{odd}(\text{suc}(x)) \iff \text{Nat0}(x))$$

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

$$\text{All } x: (\text{True } | \text{odd}(0)) \& \text{All } x: (\text{odd}(x) \mid \text{odd}(\text{suc}(x)) \iff \text{Nat0}(x))$$

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

$$\text{All } x: (\text{True } | \text{odd}(0)) \& \text{All } x: (\text{odd}(x) \mid \text{even}(x) \iff \text{Nat0}(x))$$

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to
All \( x : (\text{True} \mid \text{odd}(0)) \) \& \( \text{All} \ x : (\text{odd}(x) \mid \text{even}(x) \iff \text{even}(x) \mid \text{odd}(x)) \)

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

\text{True}

Number of proof steps: 6

-- natloop

Derivation of

\( \text{Nat}(n) \Rightarrow ((\text{loop2}(f)\,n)\,f(x)) = f((\text{loop2}(f)\,n)\,x) \)

Adding

\( \text{Nat0}(n) \Rightarrow ((\text{loop2}(f)\,n)\,f(x)) = f((\text{loop2}(f)\,n)\,x)) \)
to the axioms and applying FIXPOINT INDUCTION wrt

Nat(0)
& (Nat(suc(x)) <= Nat(x))

at position [] of the preceding formula leads to

\[ \text{All } x \text{ x0 } f: \quad (\text{Nat0}(x) \Rightarrow f((\text{loop2}(f)$x)$f(x0)) = f(f((\text{loop2}(f)$x)$x0))) \]

The reducts have been simplified.

NARROWING the preceding formula (1 step) leads to

\[ \text{All } x \text{ x0 } f:\quad (\text{All } f0 \text{ x1}:(((\text{loop2}(f0)$x)$f0(x1)) = f0((\text{loop2}(f0)$x)$x1)) \Rightarrow f((\text{loop2}(f)$x)$f(x0)) = f(f((\text{loop2}(f)$x)$x0))) \]

The axioms were MATCHED against their reducts.
The reducts have been simplified.
SUBSTITUTING \( f \) FOR \( f_0 \) to the preceding formula leads to

\[
\forall x \quad x_0 \quad f: \\
(\forall x_1: (((\text{loop2}(f)\ x)\ f(x_1)) = f((\text{loop2}(f)\ x)\ x_1)) \implies \\
f((\text{loop2}(f)\ x)\ f(x_0)) = f(f((\text{loop2}(f)\ x)\ x_0)))
\]

The reducts have been simplified.

SUBSTITUTING \( x_0 \) FOR \( x_1 \) to the preceding formula leads to

\[
\forall x \quad x_0 \quad f: \\
(((\text{loop2}(f)\ x)\ f(x_0)) = f((\text{loop2}(f)\ x)\ x_0)) \implies \\
f((\text{loop2}(f)\ x)\ f(x_0)) = f(f((\text{loop2}(f)\ x)\ x_0)))
\]

The reducts have been simplified.

REPLACING THE SUBTREES at position \([0,1,0]\) of the preceding formula leads to

\[
\forall x \quad x_0 \quad f: \\
(((\text{loop2}(f)\ x)\ f(x_0)) = f((\text{loop2}(f)\ x)\ x_0)) \implies \\
f((\text{loop2}(f)\ x)\ f(x_0)) = f(f((\text{loop2}(f)\ x)\ x_0)))
\]
The reducts have been simplified.

REPLACING THE SUBTREES at position [0,1,1,0] of the preceding formula leads to True

The reducts have been simplified.

Number of proof steps: 6
-- list

specs:    nat
preds:    P any zipAny sorted part NOTsorted
copreds:  all zipAll ~
defuncts:  F bag map foldl sum product flatten ext scan zip zipWith
evens odds mergesort split merge isort insert
fovars:   ys xs x y z s s' s1 s2 z1 z2 p
hovars:   F P

axioms:

    x:s >> s
& (s >> s' <=<=> s >> s1 & s1 >> s')
& bag(x:s) = x^bag(s)
& bag(s++s') = bag(s)^bag(s')
& map(F)[] = []
& map(F)(x:s) = F(x):map(F)(s)
& foldl(F)(x)[] = x
& foldl(F)(x)(y:s) = foldl(F)(F(x,y))(s)
& sum(s) = foldl(+)(0)(s)
product(s) = foldl(*)(1)(s)
flattten[] = []
flattten(s:p) = s++flattten(p)
ext(F)(s) = flatten(map(F)(s))
scan(F)(x)[] = [x]
scan(F)(x)(y:s) = x:scan(F)(F(x,y))(s)
zip[] [] = []
zip(x:s)(y:s') = (x,y):zip(s)(s')
zipWith(F)[] [] = []
zipWith(F)(x:s)(y:s') = F(x,y):zipWith(F)(s)(s')
(any(P)(x:s) <=== P(x) | any(P)(s))
(all(P)(x:s) ===> P(x) & all(P)(s))
(zipAny(P)(x:s)(y:s') <=== P(x,y) | zipAny(P)(s)(s'))
(zipAll(P)(x:s)(y:s') ===> P(x,y) & zipAll(P)(s)(s'))
(x `in` s <=== any(eq(x))(s))
(x `NOTin` s <=== all(neq(x))(s))
part([x],[[x]])
(part(x:y:s,[x]:p) <=== part(y:s,p))
(part(x:y:s,(x:s'):p) <=== part(y:s,s':p))
evans[] = []
evans(x:s) = x:odds(s)
& odds[] = []
& odds(x:s) = evens(s)
& (mergesort(x:y:s) = merge(mergesort(x:s1),mergesort(y:s2)))
  \iff split(s) = (s1,s2)
& mergesort[] = []
& mergesort[x] = [x]
& (split(x:(y:s)) = (x:s1,y:s2) \iff split(s) = (s1,s2))
& split[] = ([],[])
& split[x] = ([x],[])
& (merge(x:s,y:s') = x:merge(s,y:s') \iff x <= y)
& (merge(x:s,y:s') = y:merge(x:s,s') \iff x > y)
& merge([],s) = s
& merge(s,[]) = s
& isort[] = []
& isort[x] = [x]
& isort(x:s) = insert(x,isort(s))
& insert(x,[]) = [x]
& (insert(x,y:s) = x:y:s \iff x <= y)
& (insert(x,y:s) = y:insert(x,s) \iff x > y)
& sorted([])
& sorted([x])
\& (\text{sorted}(x:y:s) \iff x \leq y \& \text{sorted}(y:s))
\& (s \sim s' \implies \text{bag}(s) = \text{bag}(s'))

\text{theorems:}

\text{NOT}\text{sorted}(s) \iff \text{Not}\text{sorted}(s)
\& (\text{sorted}(s) \& \text{sorted}(s')) \implies \text{sorted}(\text{merge}(s,s'))
\& (\text{sorted}(s) \implies \text{sorted}(\text{insert}(x,s)))
\& (\text{split}(s) = (s_1,s_2) \implies s \sim s_1++s_2)
\& (s \sim \text{merge}(s_1,s_2) \iff s \sim s_1++s_2)
\& (s \sim \text{insert}(x,s') \iff s \sim x:s')
\& (\text{sorted}(x:s) \implies \text{sorted}(s))
\& (\text{sorted}(x:s) \& \text{sorted}(y:s') \& x \leq y \& \text{sorted}(s_1) \& s_1\sim(s++y:s') \implies s \sim x:s')
\& (x > y \implies y \leq x)
\& y:x:s++s' \sim x:s++y:s'
\& s'++x:s \sim x:s++s'

\text{conjects:}

(part(s,p) \implies s = \text{flatten}(p))
\&
(\text{mergesort}(s) = s' \implies \text{sorted}(s'))
\&
\[
\begin{align*}
\text{mergesort}(s) = s' & \implies s \sim s' \quad & \\
\text{isort}(s) = s' & \implies \text{sorted}(s') \quad & \\
\text{isort}(s) = s' & \implies s \sim s' \quad & \\
\text{merge}(s_1, s_2) = s & \land \text{sorted}(s_1) \land \text{sorted}(s_2) \\
& \implies \text{sorted}(s) \land s \sim s_1++s_2 \quad & \\
\text{map}(F)(s) = s' & \implies \lg(s) = \lg(s') \quad & \\
\text{zip}(\text{evens}(s), \text{odds}(s)) = s & \\
\end{align*}
\]

-- prem subsumes conc:
All \( x \ s \ z: \)
\[
\begin{align*}
\text{sorted}(x:s) & \land \text{All } s': (\text{NOTsorted}(s') \mid x:s = s') \\
& \implies \text{NOTsorted}(z++[x]) \mid x:s = z++[x])
\end{align*}
\]

terms: \( \text{merge}([1,3,5],[2,4,6,8]) \)

-- partflattten

Derivation of

\[
\text{part}(s,p) \implies s = \text{flatten}(p)
\]
Adding

\[ \text{part0}(s,p) \implies s = \text{flatten}(p) \]

to the axioms and applying FIXPOINT INDUCTION wrt

\[ \text{part}([x],[[x]]) \]
& (\text{part}(x:(y:s),[x]:p) \iff \text{part}(y:s,p))
& (\text{part}(x:(y:s),(x:s'):p) \iff \text{part}(y:s,s':p))

at position [] of the preceding formula leads to

All x y s p s':
\( ([x] = \text{flatten}([[x]]) \land \)
All x y s p s':
\( (x:(y:s)) = \text{flatten}([x]:p) \iff \text{part0}(y:s,p) \land \)
All x y s p s':
\( (x:(y:s)) = \text{flatten}((x:s'):p) \iff \text{part0}(y:s,s':p))

NARROWING the preceding formula (1 step) leads to
All \(x\ y\ s\ p\ s':\)

\([x] = ([x]++\text{flatten}[])\) \&

All \(x\ y\ s\ p\ s':\)

\((x:(y:s)) = \text{flatten}([x]:p) <=== \text{part0}(y:s,p)\) \&

All \(x\ y\ s\ p\ s':\)

\((x:(y:s)) = \text{flatten}((x:s'):p) <=== \text{part0}(y:s,s':p)\)

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

All \(x\ y\ s\ p\ s':\)

\([x] = ([x]++[])\) \&

All \(x\ y\ s\ p\ s':\)

\((x:(y:s)) = \text{flatten}([x]:p) <=== \text{part0}(y:s,p)\) \&

All \(x\ y\ s\ p\ s':\)

\((x:(y:s)) = \text{flatten}((x:s'):p) <=== \text{part0}(y:s,s':p)\)

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to
All x y s p s:
  ([x] = ([x]++[])) &

All x y s p s:
  ((x:(y:s)) = ([x]++flatten(p)) \iff part0(y:s,p)) &

All x y s p s:
  ((x:(y:s)) = flatten((x:s'):p) \iff part0(y:s,s':p))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (7 steps) leads to

All y s p:(part0(y:s,p) \implies (y:s) = flatten(p)) &

All x y s p s:
  (part0(y:s,s':p) \implies (x:(y:s)) = flatten((x:s'):p))

NARROWING the preceding formula (1 step) leads to

All y s p:((y:s) = flatten(p) \implies (y:s) = flatten(p)) &

All x y s p s:
  (part0(y:s,s':p) \implies (x:(y:s)) = flatten((x:s'):p))
The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

All x y s p s':
(part0(y:s,s':p) ==> (x:(y:s)) = flatten((x:s'):p))

NARROWING the preceding formula (1 step) leads to

All x y s p s':
((y:s) = flatten(s':p) ==> (x:(y:s)) = flatten((x:s'):p))

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

All x y s p s':
((y:s) = (s'++flatten(p)) ==> (x:(y:s)) = flatten((x:s'):p))

The axioms were MATCHED against their redices.
NARROWING the preceding formula (1 step) leads to

All x y s p s':
  ((y:s) = (s'++flatten(p)) ==> (x:(y:s)) = ((x:s')++flatten(p)))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (3 steps) leads to

True

Number of proof steps: 11

-- partflattenN

Derivation of

part(s,p) ==> s = flatten(p)
SELECTING INDUCTION VARIABLES at position [0,0] of the preceding formula leads to

All p: (part(!s,p) ==\implies!s = flatten(p))

NARROWING the preceding formula (1 step) leads to

All p: (Any x: (!s = [x] & p = [[x]])) | 
              Any x y s p0: 
                 (part(y:s,p0) & !s = (x:(y:s)) & p = ([x]:p0)) | 
              Any x y s s' p0: 
                 (part(y:s,s':p0) & !s = (x:(y:s)) & p = ((x:s'):p0)) \implies 
                 !s = flatten(p))

SIMPLIFYING the preceding formula (17 steps) leads to

All x: (!s = [x] \implies [x] = flatten([x])) & 
All p0 s y x: 
    (!s = (x:(y:s)) & part(y:s,p0) \implies (x:(y:s)) = flatten([x]:p0)) & 
All p0 s' s y x: 
    (!s = (x:(y:s)) & part(y:s,s':p0) \implies (x:(y:s)) = flatten((x:s'):p0))
NARROWING the preceding formula (1 step) leads to

All x:(!s = [x] ==> [x] = ([x]++flatten([]))) &
All p0 s y x:
  (!s = (x:(y:s)) & part(y:s,p0) ==> (x:(y:s)) = flatten([x]:p0)) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

NARROWING the preceding formula (1 step) leads to

All x:(!s = [x] ==> [x] = ([x]++[])) &
All p0 s y x:
  (!s = (x:(y:s)) & part(y:s,p0) ==> (x:(y:s)) = flatten([x]:p0)) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All x:(!s = [x] ==> [x] = (x:[])) &
All p0 s y x:
  (!s = (x:(y:s)) & part(y:s,p0) ==> (x:(y:s)) = flatten([x]:p0)) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All x:(!s = [x] ==> x = x & [] = []) &
All p0 s y x:
  (!s = (x:(y:s)) & part(y:s,p0) ==> (x:(y:s)) = flatten([x]:p0)) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s y x:
  (!s = (x:(y:s)) & part(y:s,p0) ==> (x:(y:s)) = flatten([x]:p0)) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

Applying the INDUCTION HYPOTHESIS

part(s,p) ===> (!s >> s ==> s = flatten(p))
at position [0,0,0,1] of the preceding formula leads to

All p0 s y x:
  (!s = (x:(y:s)) & (!s >> (y:s) ==> (y:s) = flatten(p0)) ==> (x:(y:s)) = flatten([x]:p0)) &

All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

NARROWING the preceding formula (1 step) leads to

All p0 s y x:
  (!s = (x:(y:s)) & (!s >> (y:s) ==> (y:s) = flatten(p0)) ==> (x:(y:s)) = ([x]++flatten(p0))) &

All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to
All p0 s y x:
(!s = (x:(y:s)) & ((x:(y:s)) >> (y:s) ==> (y:s) = flatten(p0)) ==> (x:(y:s)) = ([x]+flatten(p0))) &

All p0 s' s y x:
(!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

NARROWING at position [0,0,0,1,0] of the preceding formula (1 step) leads to

All p0 s y x:
(!s = (x:(y:s)) & (True ==> (y:s) = flatten(p0)) ==> (x:(y:s)) = ([x]+flatten(p0))) &

All p0 s' s y x:
(!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s y x:
(!s = (x:(y:s)) & (y:s) = flatten(p0) ==> (x:(y:s)) = ([x]+flatten(p0)))

All p0 s' s y x:
(!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s y x:
  (flatten(p0) = (y:s) & !s = (x:(y:s)) ==> (x:(y:s)) = ([x]++(y:s))) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s y x:
  (flatten(p0) = (y:s) & !s = (x:(y:s)) ==> (x:(y:s)) = (x:(y:s))) &
All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s' s y x:
  (!s = (x:(y:s)) & part(y:s,s':p0) ==> (x:(y:s)) = flatten((x:s'):p0))
Applying the INDUCTION HYPOTHESIS

\[ \text{part}(s,p) \implies (!s >> s \implies s = \text{flatten}(p)) \]

at position \([0,0,1]\) of the preceding formula leads to

All \(p0\) s' s y x:

\[ (!s = (x:(y:s)) \& (!s >> (y:s) \implies (y:s) = \text{flatten}(s':p0)) \implies (x:(y:s)) = \text{flatten}((x:s'):p0)) \]

NARROWING the preceding formula (1 step) leads to

All \(p0\) s' s y x:

\[ (!s = (x:(y:s)) \& (!s >> (y:s) \implies (y:s) = (s'++\text{flatten}(p0))) \implies (x:(y:s)) = \text{flatten}((x:s'):p0)) \]

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

All \(p0\) s' s y x:
NARROWING the preceding formula (1 step) leads to

All p0 s' s y x:
(!s = (x:(y:s)) & (y:s) = (s'++flatten(p0)) ==> (x:(y:s)) = flatten((x:s'):p0))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s' s y x:
(!s = (x:(y:s)) & (y:s) = (s'++flatten(p0)) ==> (x:(y:s)) = ((x:s')++flatten(p0)))

NARROWING the preceding formula (1 step) leads to

All p0 s' s y x:
(!s = (x:(y:s)) & (y:s) = (s'++flatten(p0)) ==> (x:(y:s)) = ((x:s')++flatten(p0)))
The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s' s y x:

(!s = (x:(y:s)) & (y:s) = (s'++flatten(p0)) ==> (x:(y:s)) = (x:(s'++flatten(p0))))

SIMPLIFYING the preceding formula (1 step) leads to

All p0 s' s y x:

((s'++flatten(p0)) = (y:s) & !s = (x:(y:s)) ==> (x:(y:s)) = (x:(y:s)))

SIMPLIFYING the preceding formula (1 step) leads to

True

Number of proof steps: 25
Derivation of

\[ \text{zip}(\text{evens}(s), \text{odds}(s)) = s \]

Adding

\[
(\text{zip}_0(z_3, z_4, z_5) \implies (z_3 = \text{evens}(s) \land z_4 = \text{odds}(s) \implies z_5 = s))
\]

to the axioms and applying FIXPOINT INDUCTION wrt

\[
(\text{zip}[][]) = [] \\
\land ((\text{zip}(x:s)$(y:s')) = ((x,y):z_6) \iff (\text{zip}(s)$s') = z_6)
\]

at position [] of the preceding formula leads to

All \( x \ s \ y \ s' \ z_6: \)

\[
((\text{zip}[][]) = []) \land
\]

All \( x \ s \ y \ s' \ z_6: \)

\[
((\text{zip}(x:s)$(y:s')) = ((x,y):z_6) \iff (\text{zip}(s)$s') = z_6)
\]
SIMPLIFYING the preceding formula (5 steps) leads to

All x s y s':
  ((zip(x:s)$(y:s')) = ((x,y):(zip(s)$s')))

NARROWING the preceding formula (1 step) leads to

All x s y s':
  (((x,y):(zip(s)$s')) = ((x,y):(zip(s)$s')))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

True

Number of proof steps: 4
-- LTL

preds: P Q true false hatom F `U` Head

copreds: G `W` `R` `H` isPath isPathL NatStream

defuncts: blink evens odds zip

fovars: at s s'
hovars: P Q

axioms:

    (true$s <==> True)
    & (false$s <==> False)
    & (hatom(at)$s <==> at -> head$s)

    & (F(P)$s <= P$s | F(P)$tail$s)               -- finally
    & (G(P)$s ==> P$s & G(P)$tail$s)              -- generally
    & ((P `U` Q)$s <= Q$s | P$s & (P `U` Q)$tail$s) -- until
    & ((P `W` Q)$s ==> Q$s | P$s & (P `W` Q)$tail$s) -- weak until
    & ((P `R` Q)$s ==> Q$s & (P$s | (P `R` Q)$tail$s)) -- release
    & ((P `H` Q)$s ==> P$s & ((P `H` Q)/(Q `H` P)/G(Q)$tail$s)) -- alternate
    & ((P->Q)$s <= G(not(P)/F(Q)$s))               -- leads to
& (isPath$s ===> head$s -> head$tail$s & isPath$tail$s)
& (isPathL$s ===> Any x: (head$s,x) -> head$tail$s & isPathL$tail$s)

& (NatStream(x:s) ===> Nat(x) & NatStream(s))

& head$x:s == x
& tail$x:s == s

& head$blink == 0
& tail$blink == 1:blink

& (blink = 1:blink <==> False) -- used in fairblink2 and
-- notfairblink2

& head(evens(s)) == head(s)
& tail(evens(s)) == odds(tail(s))

& head(odds(s)) == head(tail(s))
& tail(odds(s)) == odds(tail(tail(s)))

& head(zip(s,s')) == head(s)
& tail(zip(s,s')) == zip(s',tail(s))

& (not(F(P))) <=> G(not(P))
& (not(G(P))) <=> F(not(P))
& (not(P`\neg`Q) <=> not(P)`\neg`U`\neg`not(Q))

& (s ~ s' ==> head(s) = head(s') & tail(s) ~ tail(s'))

**theorems:**

(F(Q)$s <= (true`\neg`Q)$s)
& (G(P)$s <= (P`\neg`false)$s)
& ((P`\neg`U`\neg`Q)$s <= (P`\neg`W`\neg`Q)$s & F(Q)$s)
& ((P`\neg`W`\neg`Q)$s <= (P`\neg`U`\neg`Q)$s | G(P)$s)

**conjects:**

G(F$(=0).head)(blink)  --> True  (fairblink0)
& Not(G(F$(=0).head)(blink))  --> True  (notfairblink0)
& G(F$(=2).head)(blink)  --> False  (fairblink2)
& Not(G(F$(=2).head)(blink))  --> True  (notfairblink2)
& G(F$(=!x).head)(blink)  --> !x=0 | !x=1 (fairblinkx)
& G(F$(=0).\text{head})(\mu s.(0:1:s)) \quad \rightarrow \text{True} \quad \text{(fairblinkmu)}
& \text{NatStream}(\mu s.(1:2:3:s)) \quad \rightarrow \text{True} \quad \text{(natstream)}
& \text{NatStream}(1:2:3:!s) \quad \rightarrow !s = (3:!s) \mid !s = (2:(3:!s))
& \quad \rightarrow !s = (1:(2:(3:!s)))
& \quad \quad \text{(natstreamSol)}

& \text{zip(evens$s, odds$s)} \sim s

-- fairblink

Derivation of

$G(F((=0).\text{head}))$\text{blink}$

Adding

$(G0(z0)z1 \iff z0 = F((=0).\text{head}) \land z1 = \text{blink})$

to the axioms and applying COINDUCTION wrt

$(G(P)$s $\implies P(s) \land G(P)$tail(s))
at position [] of the preceding formula leads to

All P s: \( P = F(\langle =0 \rangle.\text{head}) \land s = \text{blink} \implies P(s) \land G0(P)\text{$\text{tail}(s)$} \)

SIMPLIFYING the preceding formula (6 steps) leads to

\( F(\langle =0 \rangle.\text{head})\text{$\text{blink}$} \land G0(F(\langle =0 \rangle.\text{head}))\text{$(1:blink)$} \)

NARROWING the preceding formula (1 step) leads to

\( (\langle =0 \rangle.\text{head}\text{$\text{blink}$} \land F(\langle =0 \rangle.\text{head}\text{$\text{tail}(\text{blink})$}) \land G0(F(\langle =0 \rangle.\text{head}))\text{$(1:blink)$} \)

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (6 steps) leads to

\( G0(F(\langle =0 \rangle.\text{head}))\text{$(1:blink)$} \)

Adding
\( \text{G0}(z2) \leq z2 = F((=0).\text{head}) \land z3 = (1:\text{blink}) \) 

to the axioms and applying COINDUCTION wrt

\( \text{G}(P) \Rightarrow P(s) \land \text{G}0(P) \Rightarrow \text{tail}(s) \) 

at position \([]\) of the preceding formula leads to

\( \text{All } P \ s : (P = F((=0).\text{head}) \land s = (1:\text{blink}) \Rightarrow P(s) \land \text{G0}(P) \Rightarrow \text{tail}(s)) \) 

SIMPLIFYING the preceding formula (6 steps) leads to

\( F((=0).\text{head}) \Rightarrow (1:\text{blink}) \land \text{G0}(F((=0).\text{head})) \Rightarrow \text{blink} \) 

NARROWING the preceding formula (1 step) leads to

\( (\text{=}0).\text{head} \Rightarrow (1:\text{blink}) \mid F((=0).\text{head}) \Rightarrow \text{tail}(1:\text{blink}) \) \land \text{G0}(F((=0).\text{head})) \Rightarrow \text{blink} \)

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (7 steps) leads to
F((=0).head)$blink & G0(F((=0).head))$blink

NARROWING the preceding formula (1 step) leads to

(((=0).head)$blink | F((=0).head)$tail(blink)) & G0(F((=0).head))$blink

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (6 steps) leads to

G0(F((=0).head))$blink

NARROWING the preceding formula (1 step) leads to

F((=0).head) = F(rel(SEC0,SEC0 = 0).head) & blink = (1:blink) |
F((=0).head) = F(rel(SEC0,SEC0 = 0).head) & blink = blink

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to
True

Number of proof steps: 12

-- zipEvensOddsS

Derivation of

zip(evens(s),odds(s)) ~ s

Adding

\[(z0 \sim 0 z1 \iff z0 = \text{zip}(\text{evens}(s), \text{odds}(s)) & z1 = s)\]

to the axioms and applying COINDUCTION wrt

\[(s \sim s' \iff \text{head}(s) = \text{head}(s') & \text{tail}(s) \sim \text{tail}(s'))\]

at position [] of the preceding formula leads to
All s s': (Any s0: (s = zip(evens(s0), odds(s0)) & s' = s0) ==> head(s) = head(s') & tail(s) ~0 tail(s'))

SIMPLIFYING the preceding formula (12 steps) leads to

All s0: (zip(odds(s0), odds(tail(s0))) ~0 tail(s0))

Adding

(z2 ~0 z3 <= z2 = zip(odds(s0), odds(tail(s0))) & z3 = tail(s0))

to the axioms and applying COINDUCTION wrt

(s ~ s' ==> head(s) = head(s') & tail(s) ~ tail(s'))

at position [0] of the preceding formula leads to

All s0: All s s': (Any s0: (s = zip(odds(s0), odds(tail(s0))) & s' = tail(s0))
   head(s) = head(s') & tail(s) ~0 tail(s'))
SIMPLIFYING the preceding formula (12 steps) leads to

All s0: \((\text{zip} (\text{odds} (\text{tail} (s0)), \text{odds} (\text{tail} (\text{tail} (s0)))) \sim 0 \text{ tail} (\text{tail} (s0)))\)

NARROWING the preceding formula (1 step) leads to

All s0: \((\text{Any } s1: (\text{zip} (\text{odds} (\text{tail} (s0)), \text{odds} (\text{tail} (\text{tail} (s0)))) = \text{zip} (\text{odds} (s1), \text{odds} (\text{tail} (s1))) \& \text{tail} (\text{tail} (s0)) = \text{tail} (s1)) \mid \text{Any } s: (\text{zip} (\text{odds} (\text{tail} (s0)), \text{odds} (\text{tail} (\text{tail} (s0)))) = \text{zip} (\text{evens} (s), \text{odds} (s)) \& \text{tail} (\text{tail} (s0)) = s))\)

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (2 steps) leads to

True

Number of proof steps: 6