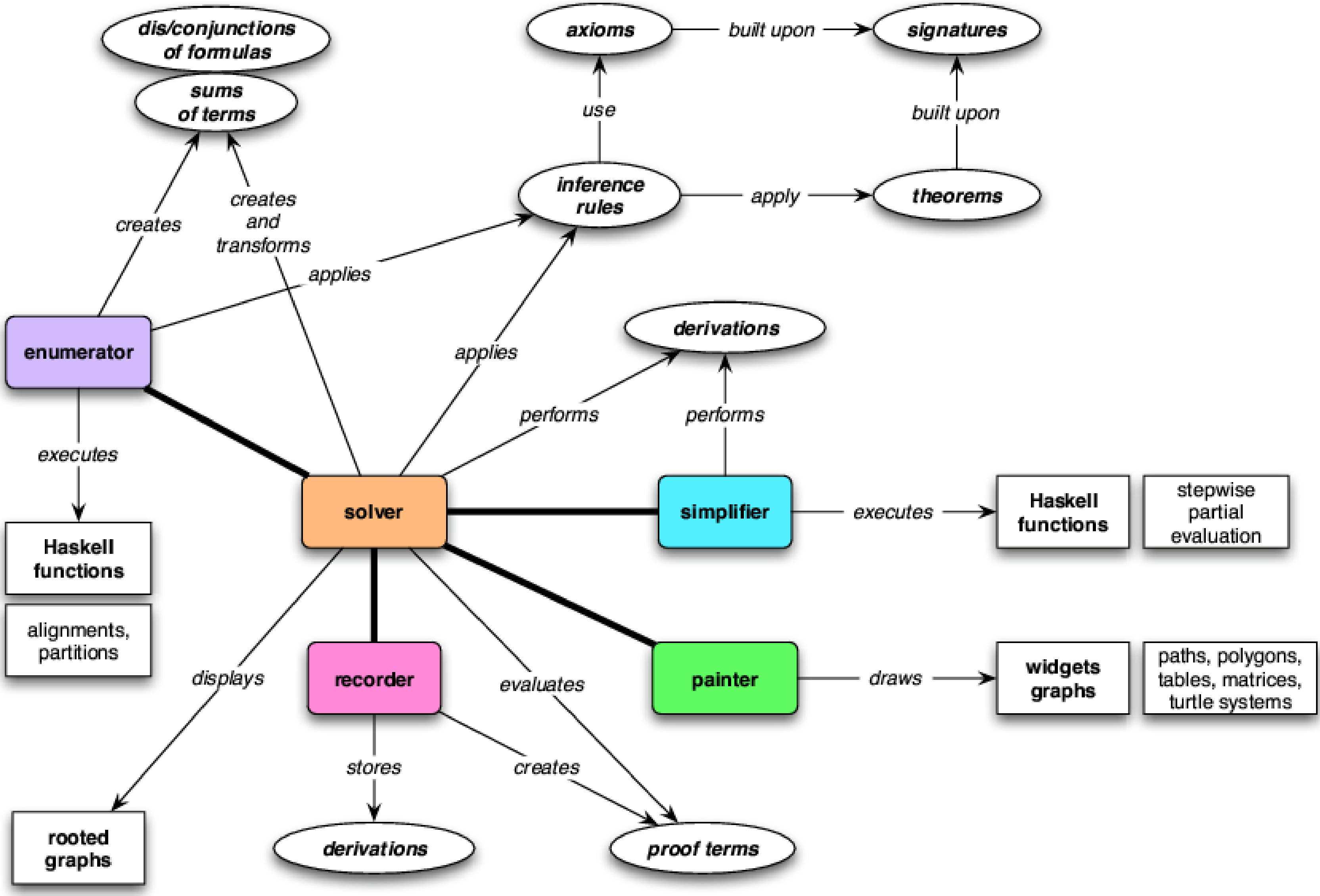
Expander2, a Haskell-based prover and rewriter

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

Peter Padawitz
TU Dortmund

7. Oktober 2011

Expander2 components


dis/conjunctions of formulas
sums of terms
axioms
built upon
signatures
use
built upon
inference rules
apply
theorems
creates
creates and transforms
applies
applies
enumerator
derivations
performs
performs
executes
solver
simplifier
executes
Haskell functions
stepwise partial evaluation
Haskell functions
alignments, partitions
displays
recorder
evaluates
painter
draws
widgets graphs
paths, polygons, tables, matrices, turtle systems
stores
creates
rooted graphs
derivations
proof terms

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

```
struct RecordS < Record = selectorS1 :: typeS1
                          selectorS2 :: typeS2

Action     < Cmd ()
Request a  < Cmd a
Template a < Cmd a

struct Methods = method1 :: type11 ... type1n1 -> Action
                 method2 :: type21 ... type2n2 -> Request type2
```

Supertyping

```
data DatatypeS > Datatype = constructorS1 typeS11 ... typeS1nS1 |
                            constructorS2 typeS21 ... 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:

$$\frac{pt}{\bigvee_{i=1}^{k} \exists Z_i : (\varphi_i\sigma_i \wedge \vec{x} = \vec{x}\sigma_i)} \quad \Updownarrow$$

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 $Z_i = var(t_i, \varphi_i)$,
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:

$$\frac{pt}{\bigwedge_{i=1}^{k} \forall Z_i : (\varphi_i\sigma_i \vee \vec{x} \neq \vec{x}\sigma_i)} \quad \Updownarrow$$

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 Σ-operation ft:

$$\frac{r(\dots,ft,\dots)}{\begin{array}{c}\bigvee_{i=1}^{k} \exists Z_i : (r(\dots,u_i,\dots)\sigma_i \wedge \varphi_i\sigma_i \wedge \vec{x} = \vec{x}\sigma_i) \vee \\ \bigvee_{i=k+1}^{l} (r(\dots,ft,\dots)\sigma_i \wedge \vec{x} = \vec{x}\sigma_i)\end{array}}$$

where r is a predicate,
$AX_f = \{\gamma_1 \Rightarrow (ft_1 = u_1 \Longleftarrow \varphi_1),\dots,\gamma_n \Rightarrow (ft_n = u_n \Longleftarrow \varphi_n)\}$,

(∗∗) $\vec{x}$ is a list of the variables of t,
for all $1 \leq i \leq k$, $t\sigma_i = t_i\sigma_i$, $\gamma_i\sigma_i \vdash \mathit{True}$ and $Z_i = \mathit{var}(t_i, u_i, \varphi_i)$,
for all $k < i \leq l$, σ_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 $\to$ be a transition predicate. $AX_\to$ is applied to an atom $t\ {}^\wedge v \to t'$:

$$\frac{t\ {}^\wedge v \to t'}{\begin{array}{c}\bigvee_{i=1}^{k} \exists Z_i : ((u_i\ {}^\wedge v)\sigma_i = t'\sigma_i \wedge \varphi_i\sigma_i \wedge \vec{x} = \vec{x}\sigma_i) \vee \\ \bigvee_{i=k+1}^{l}((t\ {}^\wedge v)\sigma_i \to t'\sigma_i \wedge \vec{x} = \vec{x}\sigma_i)\end{array}}$$

where $AX_\to = \{\gamma_1 \Rightarrow (t_1 \to u_1 \Longleftarrow \varphi_1), \dots, \gamma_n \Rightarrow (t_n \to u_n \Longleftarrow \varphi_n)\}$, $(**)$ holds true and σ_i is a unifier *modulo associativity and commutativity of* ${}^\wedge$.

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

$$\frac{pt}{\mathit{False}} \qquad \frac{qt}{\mathit{True}} \qquad \frac{r(\dots, ft, \dots)}{r(\dots, (), \dots)} \qquad \frac{t \to t'}{() \to t'}$$

where $p \not\to$ is a least predicate, q is a greatest predicate, f is a defined function and pt, qt, ft and $t \to t'$ 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 Σ-formula that shall be proved to follow from p.

Predicate induction A goal $p \Rightarrow \psi_p$ is applied to AX_p:

$$\frac{p \Rightarrow \psi_p}{\bigwedge_{pt \Leftarrow \varphi \in AX}(\varphi[\psi_p/p \mid p \in P'] \Rightarrow \psi_p t)} \Uparrow$$

Equality induction = induction upon a function

$$\frac{f(x) = y \;\Rightarrow\; \psi_f(x,y)}{\bigwedge_{f(t)=u \Leftarrow \varphi \in flat(AX_f)}(\varphi[\psi_f/(f(\_) = \_)] \Rightarrow \psi_f(t,u))} \Uparrow$$

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

Predicate coinduction A goal $\psi_p \Rightarrow p$ is applied to AX_p:

$$\frac{\psi_p \Rightarrow p}{\bigwedge_{pt \Rightarrow \varphi \in AX}(\psi_p t \Rightarrow \varphi[\psi_p/p \mid p \in P'])} \Uparrow$$

Noetherian induction

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

$$\varphi = (prem \Rightarrow conc).$$

Then the *induction hypotheses*

$$\begin{array}{lcl} conc' & \Longleftarrow & (x_1,\dots,x_n) \gg (x'_1,\dots,x'_n) \ \wedge \ prem' \\ prem' & \Longrightarrow & ((x_1,\dots,x_n) \gg (x'_1,\dots,x'_n) \ \Rightarrow \ conc') \end{array}$$

are added to the current theorems.

If φ is not an implication, then

$$\varphi' \ \Longleftarrow \ (x_1,\dots,x_n) \gg (x'_1,\dots,x'_n)$$

is added.

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

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

After axioms for $\gg$ have been added to the current axioms, narrowing steps upon $\gg$ should remove the occurrences of $\gg$ because the transformation is correct only if φ can be derived to *True*.

Incremental versions of predicate induction and coinduction

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

Predicate induction

$$(1) \qquad \frac{p \Rightarrow \psi_p}{\bigwedge_{pt \Leftarrow \varphi \in AX}(\varphi[q_p/p \mid p \in P'] \Rightarrow \psi_p t)} \qquad q_p \Rightarrow \psi_p \textit{ is added to } AX$$

$$(2) \qquad \frac{q_p \Rightarrow \delta_p}{\bigwedge_{pt \Leftarrow \varphi \in AX}(\varphi[q_p/p \mid p \in P'] \Rightarrow \delta_p t)} \qquad q_p \Rightarrow \delta_p \textit{ is added to } AX$$

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

Between the applications of (1) resp. (2), coresolution 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 coresolution 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 *True* from the conclusions of (1) and (2) one shows, roughly speaking, that the predicate $\psi_p \wedge \delta_p$ solves the axioms for p. Since p itself represents the least solution, we conclude $p \Rightarrow \psi_p \wedge \delta_p$, in particular the original goal $p \Rightarrow \psi_p$.

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

Predicate coinduction

$$(1) \qquad \frac{\psi_p \Rightarrow p}{\bigwedge_{pt \Rightarrow \varphi \in AX} (\psi_p t \Rightarrow \varphi[q_p/p \mid p \in P'])}$$

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

$$(2) \qquad \frac{\delta_p \Rightarrow q_p}{\bigwedge_{pt \Rightarrow \varphi \in AX} (\delta_p t \Rightarrow \varphi[q_p/p \mid p \in P'])}$$

$q_p \Leftarrow \delta_p$ is added to AX

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

Between the applications of (1) resp. (2), resolution steps upon the added axiom $q_p \Leftarrow \psi_p$ 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 $\psi_p \vee \delta_p$ (or its equivalence closure if p denotes a congruence relation) solves the axioms for p. Since p itself represents the greatest solution, we conclude $\psi_p \vee \delta_p \Rightarrow p$, in particular the original goal $\psi_p \Rightarrow p$.

Rewriting upon a defined function f

$$\frac{c(f(t))}{c(u_1\sigma_1)\texttt{<+>}\ldots\texttt{<+>}c(u_k\sigma_k)}$$

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 $\gamma_i\sigma_i \vdash \mathit{True}$,
for all $k < i \leq n$, t does not match t_i.

Rewriting upon the predicate $\rightarrow$

$$\frac{c(t)}{c(u_1\sigma_1)\texttt{<+>}\ldots\texttt{<+>}c(u_k\sigma_k)}$$

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

$$\frac{f(t)}{()}$$

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:

  sum(0) = 0
& sum(suc(x)) = sum(x)+x+1
& (x < y ==> div(x,y) = (0,x))
& (0 < y & y <= x & div(x-y,y) = (q,r) ==> div(x,y) = (suc(q),r))
--& (0 < y & y <= x ==> div(x,y) == case(div(x-y,y),(q,r),(suc(q),r)))
& fib(0) == 0
& fib(1) == 1
& fib(suc(suc(n))) == fib(n)+fib(suc(n))
& (Nat(0) <==> True)
& (Nat(suc(x)) <==> Nat(x))
& even(0)
& (even(suc(x)) <=== odd(x))
& (odd(suc(x)) <=== even(x))
& eq(x)(x)
& (x =/= y ==> neq(x)(y))

-- & div(x,y) = loop(y,0,x)
& (loop(y,q,r) = (q,r) <=== r < y)
& (loop(y,q,r) = loop(y,q+1,r-y) <=== r >= y)
& (INV(x,y,q,r) <=== 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)) <=== Nat(x))

-- & (INV(n,x,y,z) <=== n >= x & y = fib(n-x) & z = fib(n-x+1))
& (x >> y <=== x > y)

conjects:

  (sum(x) = y ==> x*(x+1) = 2*y)                              -- sum1
& (div(x,y) = (q,r) ==> x = (y*q)+r & r < y)                  -- div
& (x = (y*q)+r ==> loop(y,q,r) = div(x,y))                    -- divloop
& (Nat(x) ==> x+y = y+x)                                      -- comm
& (Nat(x) ==> x+(y+z) = (x+y)+z)                              -- assoc
& (Nat(x) ==> x < 2**x)                                       -- exp
& (Nat(x) ==> even(x) | odd(x))                               -- evod
& fibL(x) = fib(x)                                            -- fib
& (Nat(x) ==> suc(x)*x = x**2+x)                              -- pot
& (Nat(n) ==> 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
```

```
& (sum(suc(x)) = ((z0+x)+1) <=== sum(x) = z0)

at position [] of the preceding formula leads to

All x z0:((0*(0+1)) = (2*0)) &
All x z0:((suc(x)*(suc(x)+1)) = (2*((z0+x)+1)) <=== sum0(x,z0))

SIMPLIFYING the preceding formula (23 steps) leads to

All x z0:(sum0(x,z0) ==> ((x+(x+(x*x)))+x) = ((z0+x)+(z0+x)))

NARROWING the preceding formula (1 step) leads to

All x z0:((x*(x+1)) = (2*z0) ==> ((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

All x:(True | odd(0)) & All x:(even(suc(x)) | odd(suc(x)) <=== Nat0(x))

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

All x:(True | odd(0)) & All x:(odd(x) | odd(suc(x)) <=== Nat0(x))

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to

All x:(True | odd(0)) & All x:(odd(x) | even(x) <=== Nat0(x))

The axioms were MATCHED against their redices.

NARROWING the preceding formula (1 step) leads to
```

```
All x:(True | odd(0)) & All x:(odd(x) | even(x) <=== even(x) | odd(x))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (1 step) leads to

True

Number of proof steps: 6


-- natloop

Derivation of

Nat(n) ==> ((loop2(f)$n)$f(x)) = f((loop2(f)$n)$x)

Adding

  (Nat0(n) ===> ((loop2(f)$n)$f(x)) = f((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

All x x0 f:
 (Nat0(x) ==> f((loop2(f)$x)$f(x0)) = f(f((loop2(f)$x)$x0)))

The reducts have been simplified.

NARROWING the preceding formula (1 step) leads to

All x x0 f:
 (All f0 x1:(((loop2(f0)$x)$f0(x1)) = f0((loop2(f0)$x)$x1)) ==>
  f((loop2(f)$x)$f(x0)) = f(f((loop2(f)$x)$x0)))

The axioms were MATCHED against their redices.
The reducts have been simplified.
```

```
SUBSTITUTING f FOR f0 to the preceding formula leads to

All x x0 f:
 (All x1:(((loop2(f)$x)$f(x1)) = f((loop2(f)$x)$x1)) ==>
  f((loop2(f)$x)$f(x0)) = f(f((loop2(f)$x)$x0)))

The reducts have been simplified.

SUBSTITUTING x0 FOR x1 to the preceding formula leads to

All x x0 f:
 (((loop2(f)$x)$f(x0)) = f((loop2(f)$x)$x0) ==>
  f((loop2(f)$x)$f(x0)) = f(f((loop2(f)$x)$x0)))

The reducts have been simplified.

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

All x x0 f:
 (((loop2(f)$x)$f(x0)) = f((loop2(f)$x)$x0) ==>
  f((loop2(f)$x)$f(x0)) = f(f((loop2(f)$x)$x0)))
```

```
The reducts have been simplified.

REPLACING THE SUBTREES at position [0,1,1,0] of the preceding formula lead

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)
& flatten[] = []
& flatten(s:p) = s++flatten(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))
& evens[] = []
& evens(x:s) = x:odds(s)
& odds[] = []
& odds(x:s) = evens(s)
& (mergesort(x:y:s) = merge(mergesort(x:s1),mergesort(y:s2))
   <=== split(s) = (s1,s2))
& mergesort[] = []
& mergesort[x] = [x]
& (split(x:(y:s)) = (x:s1,y:s2) <=== split(s) = (s1,s2))
& split[] = ([],[])
& split[x] = ([x],[])
& (merge(x:s,y:s') = x:merge(s,y:s') <=== x <= y)
& (merge(x:s,y:s') = y:merge(x:s,s') <=== 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 <=== x <= y)
& (insert(x,y:s) = y:insert(x,s) <=== x > y)
& sorted([])
& sorted([x])
& (sorted(x:y:s) <=== x <= y & sorted(y:s))
& (s ~ s' ===> bag(s) = bag(s'))

theorems:

  NOTsorted(s) <=== Not(sorted(s))
& (sorted(s) & sorted(s') ===> sorted(merge(s,s')))
& (sorted(s) ===> sorted(insert(x,s)))
& (split(s) = (s1,s2) ===> s ~ s1++s2)
& (s ~ merge(s1,s2) <=== s ~ s1++s2)
& (s ~ insert(x,s') <=== s ~ x:s')
& (sorted(x:s) ===> sorted(s))
& (sorted(x:s) & sorted(y:s') & x <= y & sorted(s1) & s1~(s++y:s') ===> so
& (x > y ===> y <= x)
& y:x:s++s' ~ x:s++y:s'
& s'++x:s ~ x:s++s'

conjects:

(part(s,p) ==> s = flatten(p))                    &
(mergesort(s) = s' ==> sorted(s'))                &
```

```
(mergesort(s) = s' ==> s ~ s')                        &
(isort(s) = s' ==> sorted(s'))                        &
(isort(s) = s' ==> s ~ s')                            &
(merge(s1,s2) = s & sorted(s1) & sorted(s2)
   ==> sorted(s) & s ~ s1++s2)                        &
(map(F)(s) = s' ==> lg(s) = lg(s'))                   &
zip(evens(s),odds(s)) = s                             &
-- prem subsumes conc:
All x s z:
 (sorted(x:s) & All s': (NOTsorted(s') | x:s = s')
   ==> NOTsorted(z++[x]) | x:s = z++[x])

terms: merge([1,3,5],[2,4,6,8])



-- partflatten

Derivation of

part(s,p) ==> s = flatten(p)
```

```
Adding

  (part0(s,p) ===> s = flatten(p))

to the axioms and applying FIXPOINT INDUCTION wrt

  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))

at position [] of the preceding formula leads to

All x y s p s':
 ([x] = flatten[[x]]) &
All x y s p s':
 ((x:(y:s)) = flatten([x]:p) <=== part0(y:s,p)) &
All x y s p s':
 ((x:(y:s)) = flatten((x:s'):p) <=== part0(y:s,s':p))

NARROWING the preceding formula (1 step) leads to
```

```
All x y s p s':
 ([x] = ([x]++flatten[])) &
All x y s p s':
 ((x:(y:s)) = flatten([x]:p) <=== part0(y:s,p)) &
All x y s p s':
 ((x:(y:s)) = flatten((x:s'):p) <=== 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)) = flatten([x]:p) <=== part0(y:s,p)) &
All x y s p s':
 ((x:(y:s)) = flatten((x:s'):p) <=== 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)) <=== part0(y:s,p)) &
All x y s p s':
 ((x:(y:s)) = flatten((x:s'):p) <=== 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) ==> (y:s) = flatten(p)) &
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 y s p:((y:s) = flatten(p) ==> (y:s) = flatten(p)) &
All x y s p s':
 (part0(y:s,s':p) ==> (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 l

All p:(part(!s,p) ==> !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)) ==>
       !s = flatten(p))

SIMPLIFYING the preceding formula (17 steps) leads to

All x:(!s = [x] ==> [x] = flatten[[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))
```

```
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

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

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

NARROWING the preceding formula (1 step) leads to

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

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
```

```
-- zipEvensOddsL

Derivation of

zip(evens(s),odds(s)) = s

Adding

  (zip0(z3,z4,z5) ===> (z3 = evens(s) & z4 = odds(s) ==> z5 = s))

to the axioms and applying FIXPOINT INDUCTION wrt

  (zip[][]) = []
& ((zip(x:s)$(y:s')) = ((x,y):z6) <=== (zip(s)$s') = z6)

at position [] of the preceding formula leads to

All x s y s' z6:
 ((zip[][]) = []) &
All x s y s' z6:
 ((zip(x:s)$(y:s')) = ((x,y):z6) <=== (zip(s)$s') = z6)
```

```
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`R`Q) <==> not(P)`U`not(Q))

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

theorems:

   (F(Q)$s <=== (true`U`Q)$s)
 & (G(P)$s <=== (P`W`false)$s)
 & ((P`U`Q)$s <=== (P`W`Q)$s & F(Q)$s)
 & ((P`W`Q)$s <=== (P`U`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).head)(mu s.(0:1:s))          --> True         (fairblinkmu)
 & NatStream(mu s.(1:2:3:s))             --> True         (natstream)
 & NatStream(1:2:3:!s)                   --> !s = (3:!s) | !s = (2:(3:!s))
                                         --  !s = (1:(2:(3:!s)))
                                         --               (natstreamSol)

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



-- fairblink

Derivation of

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

Adding

  (G0(z0)$z1 <=== z0 = F((=0).head) & z1 = blink)

to the axioms and applying COINDUCTION wrt

  (G(P)$s ===> P(s) & G(P)$tail(s))
```

```
at position [] of the preceding formula leads to

All P s:(P = F((=0).head) & s = blink ===> P(s) & GO(P)$tail(s))

SIMPLIFYING the preceding formula (6 steps) leads to

F((=0).head)$blink & GO(F((=0).head))$(1:blink)

NARROWING the preceding formula (1 step) leads to

(((=0).head)$blink | F((=0).head)$tail(blink)) & GO(F((=0).head))$(1:blink

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (6 steps) leads to

GO(F((=0).head))$(1:blink)

Adding
```

```
  (G0(z2)$z3 <=== z2 = F((=0).head) & z3 = (1:blink))

to the axioms and applying COINDUCTION wrt

  (G(P)$s ===> P(s) & G(P)$tail(s))

at position [] of the preceding formula leads to

All P s:(P = F((=0).head) & s = (1:blink) ===> P(s) & G0(P)$tail(s))

SIMPLIFYING the preceding formula (6 steps) leads to

F((=0).head)$(1:blink) & G0(F((=0).head))$blink

NARROWING the preceding formula (1 step) leads to

(((=0).head)$(1:blink) | F((=0).head)$tail(1:blink)) & G0(F((=0).head))$bl

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (7 steps) leads to
```

```
F((=0).head)$blink & GO(F((=0).head))$blink

NARROWING the preceding formula (1 step) leads to

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

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (6 steps) leads to

GO(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 ~0 z1 <=== z0 = zip(evens(s),odds(s)) & z1 = s)

to the axioms and applying COINDUCTION wrt

  (s ~ s' ===> head(s) = head(s') & tail(s) ~ 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:(zip(odds(tail(s0)),odds(tail(tail(s0)))) ~0 tail(tail(s0)))

NARROWING the preceding formula (1 step) leads to

All s0:(Any s1:(zip(odds(tail(s0)),odds(tail(tail(s0)))) =
                zip(odds(s1),odds(tail(s1))) &
                tail(tail(s0)) = tail(s1)) |
        Any s:(zip(odds(tail(s0)),odds(tail(tail(s0)))) = zip(evens(s),odd
               tail(tail(s0)) = s))

The axioms were MATCHED against their redices.

SIMPLIFYING the preceding formula (2 steps) leads to

True

Number of proof steps: 6
```