Formal Verification of a Doubly Linked List Implementation

A Case Study Using the JIVE System

Marcel Labeth, Jörg Meyer, Peter Müller, and Arnd Poetzsch-Heffter
Fachbereich Informatik
Fernuniversität Hagen, Germany
Email: [Joerg.Meyer,Peter.Mueller,Arnd.Poetzsch-Heffter]@Fernuni-Hagen.de

Abstract

The JIVE system is an interactive program verification tool. It supports the application of a Hoare-style programming logic to verify object-oriented programs w.r.t. behavioral interface specifications. This technical report summarizes the results of a case study with JIVE that illustrates the specification and verification of programs with subtyping, inheritance, dynamic method binding, aliasing, and complex invariants. We present a Java implementation and a formal interface specification of a doubly linked list, summarize the proof obligations stemming from the specification, and present the proofs for these obligations. Furthermore, we briefly report on the experiences with the JIVE system.

Contents

1 Introduction

In the Lopex research project[1], we have developed formal specification and verification techniques for modern object-oriented languages. We investigated so-called logic-based programming environments that support program specification and verification. Based on this research, we developed a prototype for such a tool: the Java Interactive Verification Environment JIVE. The JIVE system provides support for syntactic analysis of programs and specifications and allows for the interactive construction of proofs in a Hoare-style programming logic. Verification is supported by various proof tactics.
To evaluate both the JIVE system and the underlying specification and verification techniques, we did a case study with a nontrivial implementation of a doubly linked list that exploits subtyping, inheritance, dynamic method binding, and aliasing. A doubly linked list is particularly interesting for specification and verification since it has a rather complex invariant and uses destructive updates and sharing of object structures intensely. This technical report presents the results of this case study and summarizes our experiences with the JIVE system.

Overview. The sequel of this report is structured as follows: The foundations and system architecture of the JIVE system are described in the next section. In Sections 3 and 4, we present the implementation of the doubly linked list together with its specification. A summary of proof obligations and the proofs in the programming logic are contained in Section 5 through 8. Our experiences with the JIVE system and future development directions are presented in Section 9.

2 Background

In this section, we present the background of the JIVE system: We sketch the Java subset as well as the specification and verification technique supported by JIVE and briefly describe the system architecture. In particular, we refer to relevant literature.

Programming Language

JIVE supports the sequential kernel of Java including recursive methods, classes, abstract classes, interfaces, thus inheritance and subtyping, static and dynamic binding, aliasing via object references, and encapsulation. Exception handling and some of the primitive types (in particular float and char) are not yet supported (cf. [MMPH97] for a precise description).

Specification Technique

JIVE uses a declarative interface specification technique which is based on the two-tiered approach developed by the Larch project [GH93]. Specifications consist of two major parts:

1. A program-independent specification (*universal specification*) that provides the mathematical vocabulary used to formulate interface specifications. In the JIVE system, universal specifications are formulated in the language of the PVS system [OSR93].

2. Program-dependent interface specifications that relate implementations to universal specifications. We have developed an interface specification language that provides pre-post specifications and type invariants [MMPH97]. Interface specifications may refer to program variables and elements of universal specifications (such as abstract data types). The common language for programs and annotations is called ANJA (Annotated Java).

For a more detailed discussion of the specification technique, confer [PH97, MPH97].

[1] See `www.informatik.fernuni-hagen.de/pi5/english/lopex_en.html`.

Verification Technique

For verification, we use a Hoare-style programming logic. Partial correctness of programs w.r.t. their specifications is shown by translating interface specifications into Hoare triples and proving these triples in the programming logic. The logic is presented and discussed in [PHM98, PHM99].

System Architecture

A detailed description of the JIVE system can be found in [MPH00a]. In the following, we explain the architectural components and the input sources of proof sessions based on the overview given in Figure 1.

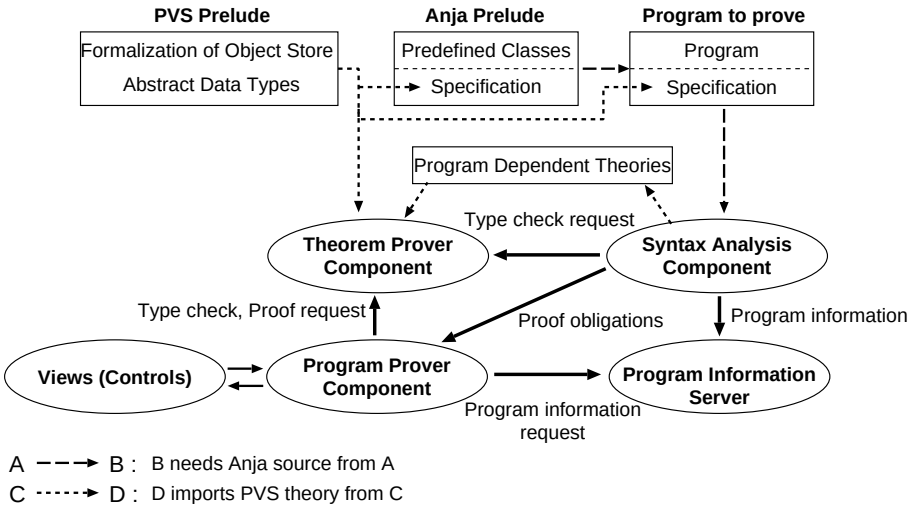

Figure 1: The JIVE architecture

System Components. The architecture is based on five components: (1) The syntax analysis component that reads in and analyzes annotated programs and generates the program proof obligations. (2) The program information server that makes the static program information gathered in the analysis phase available to other parts of the system. (3) The program prover component managing the program proofs. (4) Views to visualize program proofs and to control proof construction. (5) The theorem prover to solve program-independent proof obligations. In our current implementation, we use PVS [COR+95] for general theorem proving.
The program proof component encapsulates the construction of program proofs. It provides a container that stores all information about program proofs and an interface that provides operations to create and modify proofs within this container. Since the contents of the proof container represent the program proof state, it is strongly encapsulated to the rest of the system. Modifications of the proof state can only be achieved by operations of the container interface. Therefore correctness of proofs is ensured by the correctness of the basic container operations.
During program proof construction, various information about the underlying program is needed by the program proof component: The structure of the abstract syntax tree, results of binding and type analysis, and the program text for visualization. Such information is provided by the program information server. In contrast to a compiler frontend, all information computed during static program analysis has to be available online after the analysis.

Proof Setup. Verification of a program is based on three formal texts: (1) The PVS prelude containing two parts: (a) the formalization of the object store (see App. A); (b) the universal specifications used in program annotations (see Section 4 for the universal specifications of the

doubly linked list example). Whereas the former part is program-independent, the latter may be program-dependent. (2) The ANJA prelude containing the specifications of predefined and library classes and interfaces (see [MMPH97]). (3) An ANJA program, i.e. a program in our Java subset together with its interface specification (see Section 3).
From the described sources, the syntax analysis component generates three things: (1) The program proof obligations which need to be proven to guarantee that the program fulfills its specification (see Section 5). They are entered into the proof container. (2) Program-dependent theories formalizing some of the declaration information of the program for the theorem prover (see App. B). (3) The abstract syntax tree decorated with information of the static analysis. It is managed by the program information server.
After syntax and static analysis, the system is set up for interactive proof construction. The user constructs program proofs using basic proof operations and tactics. The views and controllers provide access to the proof state. These proofs are presented in Sections 6 through 8. Program-independent proof obligations (see App. C) are verified with the general theorem prover. The program prover monitors the overall proof process and signals the completion of proof tasks.

3 Implementation and Interface Specification

In this section, we present the annotated implementation of the doubly linked list example. The implementation consists of three classes: `Node` is a universal node class that can be used to implement doubly linked lists, binary trees, etc. `NodeL` is a subclass of `Node` for the implementation of doubly linked lists. It introduces additional methods and refines the specification of `Node`, in particular the class invariant.[2] The list header is implemented by class `DList`. The ANJA implementations of these classes are presented in the following subsections.
The interface specification uses the following abstraction functions to map objects or object structures to values of the universal specification: `aI` and `aB` map Java's `int` and `boolean` values to the corresponding values of the abstract domain (see App. A for a definition). The integer value stored in a `Node` object can be obtained by `ANode`. `ANodeL` and `ADList` map `NodeL` object structures resp. `DList` structures to lists of integer values. See Section 4 for formal definitions of these abstraction functions.

3.1 Class Node

```
decl {  I: int, N: nat, L: list[int], M: list[int], F: Value, S: Store }

class Node {
   protected int  elem;
   protected Node pred;
   protected Node succ;

   /*****************
    * initNode
    *****************/
   public static Node initNode(int i)
   pre  I = aI(i);
   post ANode(result,$) = I;
   {
      Node n;
      n = Node.newNode();
      n.elem = i;
      return n;
   }

   /****************
    * getElem
    ****************/
   public int getElem()
   pre  $=S AND T=this;
   post aI(result) = ANode(T,S);
   pre  $=S;
   post $=S;
   {
      int v;
      v = this.elem;
      return v;
   }
```

[2] We use two node classes to illustrate specification and verification in the context of inheritance.

```
   /****************
    * getPred
    ****************/
   public Node getPred()
   pre  $=S AND T=this;
   post result = S@@loc(T,Node?pred);
   pre  $=S;
   post $=S;
   {
      Node v;
      v = this.pred;
      return v;
   }

   /****************
    * getSucc
    ****************/
   public Node getSucc()
   pre  $=S AND T=this;
   post result = S@@loc(T,Node?succ);
   pre  $=S;
   post $=S;
   pre  ANodeL(this,$)   = L;
   post ANodeL(result,$) = cdr(L);
   {
      Node v;
      v = this.succ;
      return v;
   }
}
```

3.2 Class NodeL

```
class NodeL extends Node
inv  X: wfNodeL(X, $);
{
   /*****************
    * initNodeL
    *****************/
   public static NodeL initNodeL(int i)
   pre  I=aI(i);
   post ANodeL(result,$) = cons(I,null);
   pre  $=S;
   post result = new(S,NodeL) AND
        $=update(S##NodeL, loc(result,Node?elem), i);
   {
      NodeL n;
      n = NodeL.newNodeL();
      n.elem = i;
      return n;
   }
```

```
   /*****************
    * appback
    *****************/
   public int appback(NodeL n)
   req  n/=null AND n/=this AND
        lstNode?(this,$) AND fstNode?(n,$) AND lstNode?(n,$);
   pre  succn(X,$,N)=this AND ANodeL(X,$)=L AND ANodeL(n,$) = M;
   post ANodeL(X,$) = append(L,M);
   pre  $=S AND T=this AND X=n;
   post $=update(update(S, loc(T,Node?succ), X), loc(X,Node?pred), T);
   {
      this.succ = n;
      n.pred    = this;
      return 0;
   }

   /*****************
    * getLast
    *****************/
   public NodeL getLast()
   pre  T=this AND $=S;
   post $=S AND result/=null AND
        (EXISTS (N:nat): succn(T,S,N)=result AND succn(T,S,N+1)=null);
   {
       NodeL f,l;   Node n;
       boolean b;
       f = this;
       n = this.succ;
       l = (NodeL) n;
       b = Operator.notequal(l,null);
       while (b) {
           f = l;
           n = f.succ;
           l = (NodeL) n;
           b = Operator.notequal(l,null);
       }
       return f;
   }
}
```

3.3 Class DList

```
class DList
inv X: wfDList(X, $);
{
   protected NodeL firstNode;
   protected NodeL lastNode;
```

```
/****************
 * init
 ****************/
protected int init()
pre  T=this;
post ADList(T,$) = null;
{
   this.firstNode = null;  return 0;
}

/****************
 * empty
 ****************/
public static DList empty()
pre  TRUE;
post ADList(result,$) = null;
pre  alive(X,$) AND $=S;
post ($==S)(X);
{
   DList d;  d = DList.newDList();
   return d;
}

/****************
 * first
 ****************/
public int first()
req  ADList(this,$) /= null;
pre  ADList(this,$) = L;
post aI(result) = car(L);
pre  $=S;
post $=S;
{
   Node f;  int k;
   f = this.firstNode;
   k = f.getElem();
   return k;
}

/****************
 * create
 ****************/
public static DList create(NodeL f, NodeL l)
req  wf2Nodes(f,l,$);
pre  F=f AND $=S;
post ADList(result,$) = ANodeL(F,S);
pre  alive(X,$) AND $=S;
post ($==S)(X);
{
   DList d;   d = DList.newDList();
   d.firstNode = f;
   d.lastNode  = l;
   return d;
}
```

```
   /****************
    * rest
    ****************/
   public DList rest()
   req  ADList(this,$) /= null;
   pre  ADList(this,$)   = L;
   post ADList(result,$) = cdr(L);
   pre  alive(X,$) AND $=S;
   post ($==S)(X);
   {
      NodeL f,l;    NodeL s;
      DList d;      Node n;
      f = this.firstNode;  n = f.getSucc();
      s = (NodeL)n;        l = f.getLast();
      d = DList.create(s,l);
      return d;
   }

   /****************
    * isempty
    ****************/
   public boolean isempty()
   pre  ADList(this,$) = L;
   post aB(result) = null?(L);
   pre  $=S;
   post $=S;
   {
      NodeL f;    boolean b;
      f = this.firstNode;
      b = Operator.equal(f,null);  return b;
   }

   /****************
    * append
    ****************/
   public int append(int i)
   pre  ADList(this,$) = L AND I=aI(i) AND T=this;
   post ADList(T,$) = append(L,cons(I,null));
   {
      NodeL l;    boolean b;
      NodeL f,h;  int k;
      l = NodeL.initNodeL(i);
      b = this.isempty();
      if(b) {
         this.lastNode = l;  this.firstNode = l;
      } else {
         f = this.firstNode;
         h = f.getLast();
         k = h.appback(l);
         this.lastNode = l;
      }
      return 0;
   }
}
```

4 The Universal Specification

We use the predefined list data type for the abstract list values. The universal specification for the double linked list defines abstraction functions for nodes and lists as well as wellformedness conditions and several auxiliary functions and lemmas.
`wfNodeL` is the essential part of `invNodeL`, which is the invariant of class `NodeL`. In conjunct (1) resp. (2) (see below) it is required that one of the predecessors resp. successors of an `NodeLRef` must be *null*, so that it can't be infinitely long or cyclic. Conjuncts (3) and (4) describe the node structure of doubly linked lists.
The invariant of class `DList` (`invDList` and `wfDList`) specifies that a `DListRef` is either the empty list (see line (5)) or its `firstNode` and `lastNode` are both not null and one successor of `firstNode` is `lastNode` (see (6), (7), (8)). In the latter case one can show that one predecessor of `lastNode` must be `firstNode` (The following lemma holds:
`FORALL (X:NodeLObj, S:Store, n:nat):`
`invNodeL(S) AND succn(X,S,n)/=null => predn(succn(X,S,n),S,n)=X`).
Note that the invariant does not require that a `firstNode` (`lastNode`) really must be the first (last) Node in the list.This is necessary to guarantee that class NodeL's method *appback* preserves the invariant even if two lists share a common node structure: Assume, X is a `DListRef` with $X.firstNode \neq null$ and $X.lastNode = l$; further let $wfDList(X, S) = true$. If you now call `l.appback(n)` with a `NodeLRef` n fulfilling the requirement of *appback*, after execution $X.lastNode$ will stay constant but will have a successor, namely n. Thus you will get $wfDList(X, \$) = false$.
The values of an invariant in two stores should be equivalent if the stores are X-equivalent, a property expected for every invariant [PH97]. The lemmas *wfNodeL1* and *wfDList1* state this property for *invNodeL* and *invDList*.

```
dl: THEORY

  BEGIN

  IMPORTING dlPrelude

  X: VAR Value
  S, env: VAR Store
  L: VAR Location

  NodeObj?(X):  bool = typeof(X) <= ct(Node)
  NodeLObj?(X): bool = typeof(X) <= ct(NodeL)
  DListObj?(X): bool = typeof(X) <= ct(DList)
  NodeRef?(X):  bool = typeof(X) <= ct(Node)   AND X/=null
  NodeLRef?(X): bool = typeof(X) <= ct(NodeL)  AND X/=null
  DListRef?(X): bool = typeof(X) <= ct(DList)  AND X/=null

  NodeObj:  TYPE = (NodeObj?)
  NodeLObj: TYPE = (NodeLObj?)
  DListObj: TYPE = (DListObj?)
  NodeRef:  TYPE = (NodeRef?)
  NodeLRef: TYPE = (NodeLRef?)
  DListRef: TYPE = (DListRef?)

  ANode:  [Value, Store -> int]
  ANodeL: [Value, Store -> list[int]]
  ADList: [Value, Store -> list[int]]
```

```
ANode_ax: AXIOM
  NodeRef?(X) => ANode(X, S) = aI(S@@loc(X,Node?elem))

ANodeL_ax: AXIOM
  NodeLObj?(X) => ANodeL(X, S) =
  IF X=null THEN null
  ELSE cons(aI(S@@loc(X, Node?elem)), ANodeL(S@@loc(X, Node?succ), S))
  ENDIF

ANodeLn((X: NodeObj), (S: Store), (n: nat)): RECURSIVE list[int] =
  IF X=null OR n=0 THEN null
  ELSE cons(aI(S@@loc(X, Node?elem)),
            ANodeLn(S@@loc(X, Node?succ), S, n-1))
  ENDIF
  MEASURE n

ADList_ax: AXIOM
  DListRef?(X) => ADList(X, S) = ANodeL(S@@loc(X, DList?firstNode), S)

fstNode?(X, S): bool =
  NodeRef?(X) => S@@loc(X, Node?pred)=null

lstNode?(X, S): bool =
  NodeRef?(X) => S@@loc(X, Node?succ)=null

predn((X: NodeObj), S, (n: nat)): RECURSIVE NodeObj =
  IF X=null OR n=0 THEN X
  ELSE predn(S@@loc(X, Node?pred), S, n-1)
  ENDIF
  MEASURE n

succn((X: NodeObj), S, (n: nat)): RECURSIVE NodeObj =
  IF X=null OR n=0 THEN X
  ELSE succn(S@@loc(X, Node?succ), S, n-1)
  ENDIF
  MEASURE n

wfNodeL((X: NodeLObj), S): bool =
  (EXISTS (i: nat): predn(X, S, i) = null) AND         % (1)
  (EXISTS (k: nat): succn(X, S, k) = null) AND         % (2)
  (NOT fstNode?(X, S) =>                               % (3)
    typeof(S@@loc(X, Node?pred)) <= ct(NodeL) AND
    S@@loc(S@@loc(X, Node?pred), Node?succ) = X) AND
  (NOT lstNode?(X, S) =>                               % (4)
    typeof(S@@loc(X, Node?succ)) <= ct(NodeL) AND
    S@@loc(S@@loc(X, Node?succ), Node?pred) = X)

wfDList((X: DListRef), S): bool =
  S@@loc(X, DList?firstNode)=null OR                   % (5)
  S@@loc(X, DList?firstNode)/=null AND                 % (6)
  S@@loc(X, DList?lastNode)/=null AND                  % (7)
  EXISTS (n: nat):                                     % (8)
  succn(S@@loc(X, DList?firstNode), S, n)=S@@loc(X,DList?lastNode)
```

```
wf2Nodes(f,l: NodeObj, S: Store): bool =
  f=null OR
  f/=null AND l/=null AND EXISTS (n: nat): succn(f, S, n)=l

invNodeL(S): bool =
  FORALL (X: Value): NodeLRef?(X) AND alive(X, S) => wfNodeL(X, S)

invDList(S): bool =
  FORALL (X: Value): DListRef?(X) AND alive(X, S) => wfDList(X, S)

inv(S): bool =
  invNodeL(S) AND invDList(S)

NodeLObj1: LEMMA
  FORALL (X: Value): NodeLObj?(X) => NodeObj?(X)

NodeLRef1: LEMMA
  FORALL (X: Value): NodeLRef?(X) => NodeRef?(X)

ANodeLn1: LEMMA
  FORALL (X: NodeLObj, S: Store, n: nat):
  succn(X, S, n)=null AND (FORALL (m: nat): NodeLObj?(succn(X, S, m))) =>
  ANodeLn(X, S, n) = ANodeL(X, S)

ANodeLn2: LEMMA
  FORALL (X: NodeObj, S1,S2: Store, n: nat):
  (S1==S2)(X) => ANodeLn(X, S1, n) = ANodeLn(X, S2, n)

predn1: LEMMA
  FORALL (m,n: nat, X: NodeObj, S: Store):
  predn(X, S, n)=null => predn(X, S, n+m)=null

succn1: LEMMA
  FORALL (m,n: nat, X: NodeObj, S: Store):
  succn(X, S, n)=null => succn(X, S, n+m)=null

predn2: LEMMA
  FORALL (n: nat, X: NodeObj, S: Store):
  predn(X, S, n)/=null => FORALL (m: {k:nat | k<=n}): predn(X, S, m)/=null

succn2: LEMMA
  FORALL (n: nat, X: NodeObj, S: Store):
  succn(X, S, n)/=null => FORALL (m: {k:nat | k<=n}): succn(X, S, m)/=null

predn3: LEMMA
  FORALL (n: nat, X: NodeObj, S: Store): predn(X, S, n)/=null =>
  S@@loc(predn(X, S, n), Node?pred) = predn(X, S, n+1)

succn3: LEMMA
  FORALL (n: nat, X: NodeObj, S: Store): succn(X, S, n)/=null =>
  S@@loc(succn(X, S, n), Node?succ) = succn(X, S, n+1)
```

```
  predn4: LEMMA
    FORALL (m,n: nat, X: NodeObj, S: Store):
    predn(predn(X, S, n), S, m) = predn(X, S, m+n)

  succn4: LEMMA
    FORALL (m,n: nat, X: NodeObj, S: Store):
    succn(succn(X, S, n), S, m) = succn(X, S, m+n)

  predn5: LEMMA
    FORALL (n: nat, X: NodeObj, S: Store):
    predn(X, S, n) /= null => reach(X, loc(predn(X, S, n), Node?pred), S)

  succn5: LEMMA
    FORALL (n: nat, X: NodeObj, S: Store):
    succn(X, S, n) /= null => reach(X, loc(succn(X, S, n), Node?succ), S)

  predn6: LEMMA
    FORALL (X: NodeObj, S1,S2: Store):
    (S1==S2)(X) => FORALL (n: nat):  predn(X, S1, n) = predn(X, S2, n)

  succn6: LEMMA
    FORALL (X: NodeObj, S1,S2: Store):
    (S1==S2)(X) => FORALL (n: nat):  succn(X, S1, n) = succn(X, S2, n)

  succn7: LEMMA
    FORALL (X: NodeLObj, S: Store):
    invNodeL(S) => FORALL (n: nat): NodeLObj?(succn(X, S, n))

  wfNodeL1: LEMMA
    FORALL (X: NodeLObj, S1,S2: Store):
    (S1==S2)(X) => (wfNodeL(X, S1) => wfNodeL(X, S2))

  wfDList1: LEMMA
    FORALL (X: DListRef, S1,S2: Store):
    (S1==S2)(X) => (wfDList(X, S1) => wfDList(X, S2))

  ANodeL1: LEMMA
    FORALL (X: NodeLObj, S1,S2: Store):
    invNodeL(S1) AND alive(X, S1) AND (S1==S2)(X) =>
    ANodeL(X, S1) = ANodeL(X, S2)

  reach10: LEMMA
    FORALL (X: NodeObj, L: Location, S: Store):
    typeof(obj(L))=ct(DList) => NOT reach(X, L, S)

END dl
```

5 Proof Obligations

To verify a program w.r.t. its specification, a set of proof obligations is generated. The proof obligations for a specified program are defined by the semantics of the interface specification language [PH97, MPH97]. In this section, we summarize all proof obligations for the list implementation.[3] The proofs can be found in the next sections.

5.1 Class Node

1. ```
   { I=aI(i) }
       Node@initNode(int i)
   { ANode(result,$) = I }
   ```

2. ```
   { INV($) }
       Node@initNode(int i)
   { INV($) }
   ```

3. ```
   { this/=null AND S=$ AND T=this }
     Node@getElem()
   { aI(result) = ANode(T,S) }
   ```

4. ```
   { S=$ }
       Node@getElem()
   { $=S }
   ```

5. ```
   { INV($) }
       Node@getElem()
   { INV($) }
   ```

6. ```
   { this/=null AND S=$ AND T=this }
       Node@getPred()
   { result = S@@loc(T,Node?pred) }
   ```

7. ```
   { S=$ }
       Node@getPred()
   { $=S }
   ```

8. ```
   { INV($) }
       Node@getPred()
   { INV($) }
   ```

9. ```
   { this/=null AND S=$ AND T=this }
       Node@getSucc()
   { result = S@@loc(T,Node?succ) }
   ```

10. ```
    { S=$ }
        Node@getSucc()
    { $=S }
    ```

11. ```
    { this/=null AND ANodeL(this,$)=L }
        Node@getSucc()
    { ANodeL(result,$) = cdr(L) }
    ```

12. ```
    { INV($) }
        Node@getSucc(int i)
    { INV($) }
    ```

[3] For brevity, we omit the parts of preconditions (about invariants, typing, and liveness) if they are not needed in the proofs. That is, the proof obligations presented here are slightly stronger then the obligations actually generated by the JIVE system.

13. { typeof(this)<=ct(Node) AND S=$ AND T=this }
      Node:getElem()
    { aI(result) = ANode(T,S) }

14. { typeof(this)<=ct(Node) AND S=$ }
      Node:getElem()
    { $=S }

15. { typeof(this)<=ct(Node) AND S=$ AND T=this }
      Node:getPred()
    { result = S@@loc(T,Node?pred) }

16. { typeof(this)<=ct(Node) AND S=$ }
      Node:getPred()
    { $=S }

17. { typeof(this)<=ct(Node) AND S=$ AND T=this }
      Node:getSucc()
    { result = S@@loc(T,Node?succ) }

18. { typeof(this)<=ct(Node) AND S=$ }
      Node:getSucc()
    { $=S }

19. { typeof(this)<=ct(Node) AND this/=null AND ANodeL(this,$)=L }
      Node:getSucc()
    { ANodeL(result,$) = cdr(L) }

5.2 Class NodeL

1. { I=aI(i) }
      NodeL@initNodeL(int i)
    { ANode(result,$) = I }

2. { I=aI(i) }
      NodeL@initNodeL(int i)
    { ANodeL(result,$) = cons(I,null) }

3. { $=S }
      NodeL@initNodeL(int i)
    { result=new(S,NodeL) AND
      $=update(S##NodeL, loc(result,Node?elem), i) }

4. { INV($) }
      NodeL@initNodeL(int i)
    { INV($) }

5. { this/=null AND alive(this,$) AND alive(n,$) AND alive(X,$) AND
     invNodeL($) AND n/=null AND n/=this AND
     lstNode(this,$) AND fstNode(n,$) AND lstNode(n,$) AND
     succn(X,$,k)=this AND ANodeL(X,$)=L AND ANodeL(n,$) = M }
      NodeL@appback(NodeL n)
    { ANodeL(X,$) = append(L,M) }

6. { this/=null AND n/=null AND n/=this AND lstNode(this,$) AND
     fstNode(n,$) AND lstNode(n,$) AND $=S AND T=this AND N=n }
      NodeL@appback(NodeL n)
    { $=update(update(S, loc(T,Node?succ), N),loc(N,Node?pred), T) }

7. { this/=null AND alive(this,$) AND alive(n,$) AND n/=null AND n/=this AND
   lstNode(this,$) AND fstNode(n,$) AND lstNode(n,$) AND INV($) }
     NodeL@appback(NodeL n)
   { INV($) }

8. { this/=null AND T=this AND $=S }
     NodeL@getLast()
   { $=S AND result/=null AND
   (EXISTS N: succn(T,S,N)=result AND succn(T,S,N+1)=null) }

9. { INV($) }
     NodeL@getLast()
   { INV($) }

10. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ AND T=this }
    Node:getElem()
    { aI(result) = ANode(T,S) }

11. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ }
      NodeL:getElem()
    { $=S }

12. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ AND T=this }
      NodeL:getPred()
    { result = S@@loc(T,Node?pred) }

13. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ }
      NodeL:getPred()
    { $=S }

14. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ AND T=this }
      NodeL:getSucc()
    { result = S@@loc(T,Node?succ) }

15. { this/=null AND typeof(this)<=ct(NodeL) AND S=$ }
      NodeL:getSucc()
    { $=S }

16. { this/=null AND typeof(this)<=ct(NodeL) AND ANodeL(this,$)=L }
      NodeL:getSucc()
    { ANodeL(result,$)=cdr(L) }

17. { this/=null AND typeof(this)<=ct(NodeL) AND alive(this,$) AND
    alive(n,$) AND alive(X,$) AND invNodeL($) AND n/=null AND
    n/=this AND lstNode(this,$) AND fstNode(n,$) AND lstNode(n,$) AND
    succn(X,$,k)=this AND ANodeL(X,$)=L AND ANodeL(n,$) = M }
      NodeL:appback(NodeL n)
    { ANodeL(X,$) = append(L,M) }

18. { this/=null AND typeof(this)<=ct(NodeL) AND n/=null AND n/=this AND
    lstNode(this,$) AND fstNode(n,$) AND lstNode(n,$) AND
    $=S AND T=this AND N=n }
      NodeL:appback(NodeL n)
    { $=update(update(S, loc(T,Node?succ), N), loc(N,Node?pred), T) }

19. { this/=null AND typeof(this)<=ct(NodeL) AND T=this AND $=S }
      NodeL:getLast()
    { $=S AND result/=null AND
    (EXISTS N: succn(T,S,N)=result AND succn(T,S,N+1)=null) }

5.3 Class DList

1. { this/=null AND alive(this,$) AND T=this }
   DList@init()
   { ADList(T,$)=null }

2. { this/=null AND alive(this,$) AND INV($) }
   DList@init()
   { INV($) }

3. { TRUE }
   DList@empty()
   { ADList(result,$)=null }

4. { alive(X,$) AND S=$ }
   DList@empty()
   { ($==S)(X) }

5. { INV($) }
   DList@empty()
   { INV($) }

6. { this/=null AND ADList(this,$)/=null AND ADList(this,$)=L }
   DList@first()
   { aI(result)=car(L) }

7. { this/=null AND ADList(this,$)/=null AND S=$ }
   DList@first()
   { $=S }

8. { this/=null AND ADList(this,$)/=null AND INV($) }
   DList@first()
   { INV($) }

9. { wf2Nodes(f,l,$) AND alive(X,$) AND S=$ }
   DList@create(NodeL f, NodeL l)
   { ($==S)(X) }

10. { alive(f,$) AND invNodeL($) AND wf2Nodes(f,l,$) AND F=f AND S=$ }
    DList@create(NodeL f, NodeL l)
    { ADList(result,$)=ANodeL(F,S) }

11. { alive(f,$) AND alive(l,$) AND wf2Nodes(f,l,$) AND INV($) }
    DList@create(NodeL f, NodeL l)
    { INV($) }

12. { this/=null AND ADList(this,$)/=null AND ADList(this,$)=L }
    DList@rest()
    { ADList(result,$)=cdr(L) }

13. { this/=null AND ADList(this,$)/=null AND alive(X,$) AND S=$ }
    DList@rest()
    { ($==S)(X) }

14. { this/=null AND ADList(this,$)/=null AND INV($) }
    DList@rest()
    { INV($) }

15. { this/=null AND ADList(this,$)=L }
      DList@isempty()
    { result=null?(L) }

16. { this/=null AND S=$ }
      DList@isempty()
    { $=S }

17. { this/=null AND INV($) }
      DList@isempty()
    { INV($) }

18. { this/=null AND alive(this,$) AND INV($) AND
    ADList(this,$)=L AND I=aI(i) AND T=this }
      DList@append(int i)
    { ADList(T,$)=append(L,cons(I,null)) }

19. { this/=null AND alive(this,$) AND INV($) }
      DList@append(int i)
    { INV($) }

20. { typeof(this)<=ct(DList) AND this/=null AND alive(this,$) AND T=this }
      DList:init()
    { ADList(T,$)=null }

21. { typeof(this)<=ct(DList) AND this/=null AND ADList(this,$)/=null AND
    ADList(this,$)=L }
      DList:first()
    { result=car(L) }

22. { typeof(this)<=ct(DList) AND this/=null AND ADList(this,$)/=null AND S=$ }
      DList:first()
    { $=S }

23. { typeof(this)<=ct(DList) AND this/=null AND ADList(this,$)/=null AND
    ADList(this,$)=L }
      DList:rest()
    { ADList(result,$)=cdr(L) }

24. { typeof(this)<=ct(DList) AND this/=null AND ADList(this,$)/=null AND
    alive(X,$) AND S=$ }
      DList:rest()
    { ($==S)(X) }

25. { typeof(this)<=ct(DList) AND this/=null AND ADList(this,$)=L }
      DList:isempty()
    { result=null?(L) }

26. { typeof(this)<=ct(DList) AND this/=null AND S=$ }
      DList:isempty()
    { $=S }

27. { typeof(this)<=ct(DList) AND this/=null AND this/=null AND alive(this,$)
    AND INV($) AND ADList(this,$)=L AND I=aI(i) AND T=this }
      DList@append(int i)
    { ADList(T,$)=append(L,cons(I,null)) }

6 Verification of Class Node

In this and the next two sections, we present the proofs for the classes `Node`, `NodeL`, and `DList`. All proofs have been checked with the JIVE system. However, since the JIVE system does not provide LaTeX output so far, we manually laid out the proofs as proof outlines [PH97]. The numbering of the proofs refers to the list of proof obligations in Section 5. To make the proofs easier to read, we do not stick to the syntax of the PVS language in the proof outlines. All program-independent proof obligations, that occur e.g. in every application of the weak- or strength-rule, are contained in App. C. We use names such as *Node1_1* to refer to these lemmas.

6.1 Obligation 1

Abreviations:

$S_1 := update(\$, loc(n, Node?elem), i)$

Lemmas:

1.

$\{\$ = S\}$

```
Node@newNode();
```

$\{\$ = S\#\#Node \wedge result = new(S, Node)\}$

———————————————————————————————[static-invocation-rule]

$\{\$ = S\}$

```
Node n; n = Node.newNode();
```

$\{\$ = S\#\#Node \wedge n = new(S, Node)\}$

Proof:

$\{I = aI(i)\}$

$\Longrightarrow$ (Node1_1)

$\{\exists S : \ \$ = S \wedge I = aI(i)\}$

———————————————————————————————↓–[ex-rule]

$\{\$ = S \wedge I = aI(i)\}$

———————————————————————————————↓–[var-rule]

$\{\$ = S \wedge I = aI(I_1)\}$

———————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
Node n; n = Node.newNode();  //lemma 1
```

$\{\$ = S\#\#Node \wedge n = new(S, Node)\}$

———————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#Node \wedge n = new(S, Node) \wedge I = aI(I_1)\}$

———————————————————————————————↑–[var-rule]

$\{\$ = S\#\#Node \wedge n = new(S, Node) \wedge I = aI(i)\}$

$\Longrightarrow$ (Node1_2)

$\{n \neq null \wedge ANode(n, S_1) = I\}$

```
n.elem = i;
```

$\{ANode(n, \$) = I\}$

```
return n;
```

$\{ANode(result, \$) = I\}$

———————————————————————————————↑–[ex-rule]

$\{ANode(result, \$) = I\}$

6.2 Obligation 2

Abreviations and lemmas: same as in obligation 1.

Proof:

$\{inv(\$)\}$
$\Longrightarrow$ (Node11_1)
$\{\exists S : \ \$ = S \wedge inv(\$)\}$
————————————————————————————————————↓–[ex-rule]
$\{\$ = S \wedge inv(\$)\}$
$\Longrightarrow$ (Node11_2)
$\{\$ = S \wedge inv(S)\}$
————————————————————————————————————↓–[inv-rule]
$\{\$ = S\}$

```
Node n; n = Node.newNode();
```

$\{\$ = S\#\#Node \wedge n = new(Node, S)\}$
————————————————————————————————————↑–[inv-rule]
$\{\$ = S\#\#Node \wedge n = new(Node, S) \wedge inv(S)\}$
$\Longrightarrow$ (Node11_3)
$\{n \neq null \wedge inv(S_1)\}$

```
n.elem = i;
```

$\{inv(\$)\}$

```
return n;
```

$\{inv(\$)\}$
————————————————————————————————————↑–[ex-rule]
$\{inv(\$)\}$

6.3 Obligation 3

Proof:

$\{this \neq null \wedge S = \$ \wedge T = this\}$
$\Longrightarrow$ (Node2_1)
$\{aI(\$@@loc(this, Node?elem)) = ANode(T, S)\}$

```
int v; v = this.elem;
```

$\{aI(v) = ANode(T, S)\}$

```
return v;
```

$\{aI(result) = ANode(T, S)\}$

6.4 Obligation 4

Proof:

$\{\$ = S\}$

```
int v; v = this.elem;
```

$\{\$ = S\}$

```
return v;
```

$\{\$ = S\}$

6.5 Obligation 5

Proof:

$\{inv(\$)\}$

```
int v; v = this.elem;
```

$\{inv(\$)\}$

```
return v;
```

$\{inv(\$)\}$

6.6 Obligation 6

Proof:

$\{this \neq null \wedge S = \$ \wedge T = this\}$
$\Longrightarrow$ (Node4_1)
$\{\$@@loc(this, Node?pred) = S@@loc(T, Node?pred)\}$

```
Node v; v = this.pred;
```

$\{v = S@@loc(T, Node?pred)\}$

```
return v;
```

$\{result = S@@loc(T, Node?pred)\}$

6.7 Obligation 7

Proof:

$\{\$ = S\}$

```
Node v; v = this.pred;
```

$\{\$ = S\}$

```
return v;
```

$\{\$ = S\}$

6.8 Obligation 8

Proof:

$\{inv(\$)\}$

```
Node v; v = this.pred;
```

$\{inv(\$)\}$

```
return v;
```

$\{inv(\$)\}$

6.9 Obligation 9

Proof:

$\{this \neq null \wedge S = \$ \wedge T = this\}$
$\Longrightarrow$ (Node8_1)
$\{\$@@loc(this, Node?succ) = S@@loc(T, Node?succ)\}$

```
Node v; v = this.succ;
```

$\{v = S@@loc(T, Node?succ)\}$

```
return v;
```

$\{result = S@@loc(T, Node?succ)\}$

6.10 Obligation 10

Proof:

$\{\$ = S\}$

```
Node v; v = this.succ;
```

$\{\$ = S\}$

```
return v;
```

$\{\$ = S\}$

6.11 Obligation 11

Proof:

$\{this \neq null \wedge ANodeL(this, \$) = L\}$
$\Longrightarrow$ (Node10_1)
$\{ANodeL(\$(loc(this, Node?succ)), \$) = cdr(L)\}$

```
Node v; v = this.succ;
```
$\{ANodeL(v, \$) = cdr(L)\}$
```
return v;
```
$\{ANodeL(result, \$) = cdr(L)\}$

6.12 Obligation 12

Proof:
$\{inv(\$)\}$
```
Node v; v = this.succ;
```
$\{inv(\$)\}$
```
return v;
```
$\{inv(\$)\}$

6.13 Obligation 13

Let P resp. Q be the pre– resp. postcondition of Node's obligation 3.
Proof :
$\{typeof(this) = ct(Node) \wedge P\}$ Node@getElem() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(Node) \wedge P\}$ Node:getElem() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(Node) \wedge P\}$ Node:getElem() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(Node) \wedge P) \Rightarrow P$
$\{P\}$ Node@getElem() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(Node) \wedge P\}$ Node@getElem() $\{Q\}$

Lemma 2:
$ct(NodeL) \leq ct(Node)$
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ Node:getElem() $\{Q\}$
$(typeof(this) < ct(Node) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof(this) \leq ct(NodeL) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof(this) < ct(Node) \wedge P\}$ Node:getElem() $\{Q\}$

6.14 Obligation 14

Let P resp. Q be the pre– resp. postcondition of Node's obligation 4.
Proof :
$\{typeof(this) = ct(Node) \wedge P\}$ Node@getElem() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(Node) \wedge P\}$ Node:getElem() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(Node) \wedge P\}$ Node:getElem() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(Node) \wedge P) \Rightarrow P$
$\{P\}$ Node@getElem() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(Node) \wedge P\}$ Node@getElem() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getElem() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getElem() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getElem() $\{Q\}$

6.15 Obligation 15

Let P resp. Q be the pre– resp. postcondition of Node's obligation 6.
Proof :
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getPred() $\{Q\}$ (Lemma 1)
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this\,) \leq ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$

Lemma 1:
$(typeof\,(this\,) = ct\,(Node\,) \wedge P) \Rightarrow P$
$\{P\}$ Node@getPred() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getPred() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getPred() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getPred() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$

6.16 Obligation 16

Let P resp. Q be the pre– resp. postcondition of Node's obligation 7.
Proof :
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getPred() $\{Q\}$ (Lemma 1)
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this\,) \leq ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$

Lemma 1:
$(typeof\,(this\,) = ct\,(Node\,) \wedge P) \Rightarrow P$
$\{P\}$ Node@getPred() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getPred() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getPred() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getPred() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getPred() $\{Q\}$

6.17 Obligation 17

Let P resp. Q be the pre– resp. postcondition of Node's obligation 9.
Proof :
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this\,) \leq ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

Lemma 1:
$(typeof\,(this\,) = ct\,(Node\,) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getSucc() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getSucc() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

6.18 Obligation 18

Let P resp. Q be the pre– resp. postcondition of Node's obligation 10.
Proof :
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this\,) \leq ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

Lemma 1:
$(typeof\,(this\,) = ct\,(Node\,) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getSucc() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getSucc() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

6.19 Obligation 19

Let P resp. Q be the pre– resp. postcondition of Node's obligation 11.
Proof :
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this\,) \leq ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

Lemma 1:
$(typeof\,(this\,) = ct\,(Node\,) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) = ct\,(Node\,) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$ct\,(NodeL\,) \leq ct\,(Node\,)$
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ NodeL:getSucc() $\{Q\}$
———————————————————————————————[subtype-rule]
$\{typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P\}$ Node:getSucc() $\{Q\}$
$(typeof\,(this\,) < ct\,(Node\,) \wedge P) \Rightarrow$ (NodeVirt1) $(typeof\,(this\,) \leq ct\,(NodeL\,) \wedge P)$
———————————————————————————————[strength-rule]
$\{typeof\,(this\,) < ct\,(Node\,) \wedge P\}$ Node:getSucc() $\{Q\}$

7 Verification of Class NodeL

7.1 Obligation 1

Abreviations:
$S_1 := update(\$, loc(n, Node?elem), i)$

Lemmas:
1.
$\{\$ = S\}$
`NodeL@newNodeL();`
$\{\$ = S\#\#NodeL \wedge result = new(S, NodeL)\}$
————————————————————————————————[static-invocation-rule]
$\{\$ = S\}$
`NodeL n; n = NodeL.newNodeL();`
$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL)\}$

Proof:
$\{I = aI(i)\}$
$\Longrightarrow$ (NodeL1_1)
$\{\exists S : \$ = S \wedge I = aI(i)\}$
————————————————————————————————↓–[ex-rule]
$\{\$ = S \wedge I = aI(i)\}$
————————————————————————————————↓–[var-rule]
$\{\$ = S \wedge I = aI(I_1)\}$
————————————————————————————————↓–[inv-rule]
$\{\$ = S\}$
`NodeL n; n = NodeL.newNodeL(); //lemma 1`
$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL)\}$
————————————————————————————————↑–[inv-rule]
$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL) \wedge I = aI(I_1)\}$
————————————————————————————————↑–[var-rule]
$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL) \wedge I = aI(i)\}$
$\Longrightarrow$ (NodeL1_2)
$\{n \neq null \wedge ANode(n, S_1) = I\}$
`n.elem = i;`
$\{ANode(n, \$) = I\}$
`return n;`
$\{ANode(result, \$) = I\}$
————————————————————————————————↑–[ex-rule]
$\{ANode(result, \$) = I\}$

7.2 Obligation 2

Abreviations and Lemmas: same as in obligation 1.

Proof:
$\{I = aI(i)\}$
$\Longrightarrow$ (NodeL2_1)
$\{\exists S : \$ = S \wedge I = aI(i)\}$
————————————————————————————————↓–[ex-rule]
$\{\$ = S \wedge I = aI(i)\}$
————————————————————————————————↓–[var-rule]
$\{\$ = S \wedge I = aI(I_1)\}$

———————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
NodeL n; n = NodeL.newNodeL();  // lemma 1
```

$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL)\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL) \wedge I = aI(I_1)\}$

———————————————————————————————————↑–[var-rule]

$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL) \wedge I = aI(i)\}$

$\Longrightarrow$ (NodeL2_2)

$\{n \neq null \wedge ANodeL(n, S_1) = cons(I, null)\}$

```
n.elem = i;
```

$\{ANodeL(n, \$) = cons(I, null)\}$

```
return n;
```

$\{ANodeL(result, \$) = cons(I, null)\}$

———————————————————————————————————↑–[ex-rule]

$\{ANodeL(result, \$) = cons(I, null)\}$

7.3 Obligation 3

Abreviations and Lemmas: same as in obligation 1.

Proof:

$\{\$ = S\}$

```
NodeL n; n = NodeL.newNodeL();
```

$\{\$ = S\#\#NodeL \wedge n = new(S, NodeL)\}$

$\Longrightarrow$ (NodeL6_1)

$\{n \neq null \wedge S_1 = update(S\#\#NodeL, loc(n, Node?elem), i) \wedge n = new(S, NodeL)\}$

```
n.elem = i;
```

$\{\$ = update(S\#\#NodeL, loc(n, Node?elem), i) \wedge n = new(S, NodeL)\}$

```
return n;
```

$\{\$ = update(S\#\#NodeL, loc(result, Node?elem), i) \wedge result = new(S, NodeL)\}$

7.4 Obligation 4

Abreviations and Lemmas: same as in obligation 1.

Proof:

$\{inv(\$)\}$

$\Longrightarrow$ (NodeL3_1)

$\{\exists S : \$ = S \wedge inv(\$)\}$

———————————————————————————————————↓–[ex-rule]

$\{\$ = S \wedge inv(\$)\}$

$\Longrightarrow$ (NodeL3_2)

$\{\$ = S \wedge inv(S)\}$

———————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
NodeL n; n = NodeL.newNodeL();
```

$\{\$ = S\#\#NodeL \wedge n = new(NodeL, S)\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#NodeL \wedge n = new(NodeL, S) \wedge inv(S)\}$

$\Longrightarrow$ (NodeL3_3)

$\{n \neq null \wedge inv(S_1)\}$

```
n.elem = i;
```

$\{inv(\$)\}$

```
return n;
```
$\{inv(\$)\}$

———↑–[ex-rule]

$\{inv(\$)\}$

7.5 Obligation 5

Abreviations:
$S_1 := update(\$, loc(n, Node?pred), this)$
$S_2 := update(update(\$, loc(this, Node?succ), n), loc(n, Node?pred), this)$

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ alive(this, \$) \quad \wedge \\ alive(n, \$) \quad \wedge \\ alive(X, \$) \quad \wedge \\ invNodeL(\$) \quad \wedge \\ n \neq null \quad \wedge \\ n \neq this \quad \wedge \\ fstNode?(n, \$) \quad \wedge \\ lstNode?(n, \$) \quad \wedge \\ lstNode?(this, \$) \quad \wedge \\ ANodeL(X, \$) = L \quad \wedge \\ ANodeL(n, \$) = M \quad \wedge \\ succn(X, \$, k) = this \end{array}\right\}$$

$\Longrightarrow$ (NodeL4_1)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ n \neq null \quad \wedge \\ ANodeL(X, S_2) = append(L, M) \end{array}\right\}$$

```
this.succ = n;
```

$$\left\{\begin{array}{l} n \neq null \quad \wedge \\ ANodeL(X, S_1) = append(L, M) \end{array}\right\}$$

```
n.pred = this;
```
$\{ANodeL(X, \$) = append(L, M)\}$
```
return 0;
```
$\{ANodeL(X, \$) = append(L, M)\}$

7.6 Obligation 6

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ n \neq null \quad \wedge \\ n \neq this \quad \wedge \\ fstNode?(n, \$) \quad \wedge \\ lstNode?(n, \$) \quad \wedge \\ lstNode?(this, \$) \quad \wedge \\ \$ = S \quad \wedge \\ T = this \quad \wedge \\ N = n \end{array}\right\}$$

$\Longrightarrow$ (NodeL7_1)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ n \neq null \quad \wedge \\ update(update(\$,\ loc(this, Node?succ),\ n),\ loc(n, Node?pred),\ this) = \\ update(update(S,\ loc(T, Node?succ),\ N),\ loc(N, Node?pred),\ T) \end{array}\right\}$$

```
this.succ = n;
```

$$\left\{\begin{array}{l} n \neq null \quad \wedge \\ update(\$,\ loc(n, Node?pred),\ this) = \\ update(update(S,\ loc(T, Node?succ),\ N),\ loc(N, Node?pred),\ T) \end{array}\right\}$$

```
n.pred     = this;
```

$\{\$ = update(update(S,\ loc(T, Node?succ),\ N),\ loc(N, Node?pred),\ T)\}$

```
return 0;
```

$\{\$ = update(update(S,\ loc(T, Node?succ),\ N),\ loc(N, Node?pred),\ T)\}$

7.7 Obligation 7

Abreviations: same as in obligation 5

Proof:

$$\left\{\begin{array}{l} this \neq null \wedge alive(this, \$) \wedge alive(n, \$) \quad \wedge \\ n \neq null \wedge n \neq this \wedge inv(\$) \quad \wedge \\ lstNode?(this, \$) \wedge lstNode?(n, \$) \wedge fstNode?(n, \$) \end{array}\right\}$$

$\Longrightarrow$ (NodeL5_1)

$\{n \neq null \wedge this \neq null \wedge inv(S_2)\}$

```
this.succ = n;
```

$\{n \neq null \wedge inv(S_1)\}$

```
n.pred = this;
```

$\{inv(\$)\}$

```
return 0;
```

$\{inv(\$)\}$

7.8 Obligation 8

Lemmas:

1.

$\{A = a \wedge B = b \wedge \$ = S\}$

```
Operator@notequal(a,b)
```

$\{aB(result) = (A \neq B) \wedge \$ = S\}$

———————————————————————————————————————[static-invocation-rule]

$\{A = l \wedge B = null \wedge \$ = S\}$

```
b = Operator.notequal(l,null)
```

$\{aB(b) = (A \neq B) \wedge \$ = S\}$

———————————————————————————————————————[var-rule]

$\{\$ = S\}$

```
b = Operator.notequal(l,null)
```

$\{aB(b) = (l \neq null) \wedge \$ = S\}$

Proof:

$\{this \neq null \wedge T = this \wedge \$ = S\}$

$\Longrightarrow$ (NodeL8_1)

$$\left\{\begin{array}{l} this \neq null \wedge \$ = S \quad \wedge \\ (\exists N : succn(T, S, N) = this \wedge succn(T, S, N+1) = \$@@loc(this, Node?pred)) \end{array}\right\}$$

```
NodeL f; f = this;
```

$$\left\{\begin{array}{l} this \neq null \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T, S, N) = f \wedge succn(T, S, N+1) = \$@@loc(this, Node?pred)) \end{array}\right\}$$

```
Node n; n = this.succ;
```

$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T, S, N) = f \wedge succn(T, S, N+1) = n) \end{array}\right\}$$

$\Longrightarrow$

$$\left\{\begin{array}{l} ct(NodeL) \leq typeof(n) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T, S, N) = f \wedge succn(T, S, N+1) = n) \end{array}\right\}$$

```
NodeL l; l = (NodeL) n;
```
$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \end{array}\right\}$$
——————————————————————————↓–[var-rule]
$$\left\{\begin{array}{l} \$ = S \wedge F \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = F \wedge succn(T,S,N+1) = L) \end{array}\right\}$$
——————————————————————————↓–[inv-rule]
$\{\$ = S\}$
```
boolean b; b = Operator.notequal(l,null); // lemma 1
```
$\{aB(b) = (l \neq null) \wedge \$ = S\}$
——————————————————————————↑–[inv-rule]
$$\left\{\begin{array}{l} aB(b) = (l \neq null) \wedge \$ = S \wedge F \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = F \wedge succn(T,S,N+1) = L) \end{array}\right\}$$
——————————————————————————↑–[var-rule]
$$\left\{\begin{array}{l} aB(b) = (l \neq null) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \end{array}\right\}$$
```
while(b) {
```
——————————————————————————↓–[while-rule]
$$\left\{\begin{array}{l} aB(b) \wedge aB(b) = (l \neq null) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \end{array}\right\}$$
$\Longrightarrow$ (NodeL8_2)
$$\left\{\begin{array}{l} \$ = S \wedge l \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = l \wedge succn(T,S,N+1) = \$@@loc(l, Node?succ)) \end{array}\right\}$$
```
f = l;
```
$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = \$@@loc(f, Node?succ)) \end{array}\right\}$$
```
n = f.succ;
```
$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = n) \end{array}\right\}$$
$\Longrightarrow$
$$\left\{\begin{array}{l} ct(NodeL) \leq typeof(n) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = n) \end{array}\right\}$$
```
l = (NodeL) n;
```
$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \end{array}\right\}$$
——————————————————————————↓–[var-rule]
$$\left\{\begin{array}{l} \$ = S \wedge F \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = F \wedge succn(T,S,N+1) = L) \end{array}\right\}$$
——————————————————————————↓–[inv-rule]
$\{\$ = S\}$
```
b = Operator.notequal(l,null);  // lemma 1
```
$\{aB(b) = (l \neq null) \wedge \$ = S\}$
——————————————————————————↑–[inv-rule]
$$\left\{\begin{array}{l} aB(b) = (l \neq null) \wedge \$ = S \wedge F \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = F \wedge succn(T,S,N+1) = L) \end{array}\right\}$$
——————————————————————————↑–[var-rule]
$$\left\{\begin{array}{l} aB(b) = (l \neq null) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \end{array}\right\}$$
——————————————————————————↑–[while-rule]
$$\left\{\begin{array}{l} aB(b) = (l \neq null) \wedge \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = l) \wedge \neg aB(b) \end{array}\right\}$$
$\Longrightarrow$ (NodeL8_3)
$$\left\{\begin{array}{l} \$ = S \wedge f \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = f \wedge succn(T,S,N+1) = null) \end{array}\right\}$$
```
return f;
```

$$\left\{ \begin{array}{l} \$ = S \wedge result \neq null \quad \wedge \\ (\exists N : succn\,(T, S, N) = result \wedge succn\,(T, S, N+1) = null) \end{array} \right\}$$

7.9 Obligation 9

Proof:

$\{inv\,(\$)\}$
$\Longrightarrow$ (NodeL9_1)
$\{\exists S, T : this = T \wedge \$ = S \wedge inv\,(\$)\}$
———————————————————————————————↓–[ex-rule]
$\{T = this \wedge \$ = S \wedge inv\,(\$)\}$
$\Longrightarrow$ (NodeL9_2)
$\{T = this \wedge \$ = S \wedge inv\,(S)\}$
———————————————————————————————↓–[inv-rule]
$\{T = this \wedge \$ = S\}$
`NodeL@getLast()`
$\left\{ \begin{array}{l} \$ = S \wedge result \neq null \quad \wedge \\ (\exists N : succn\,(T, S, N) = result \wedge succn\,(T, S, N+1) = null) \end{array} \right\}$
$\Longrightarrow$ (NodeL9_3)
$\{\$ = S\}$
———————————————————————————————↑–[inv-rule]
$\{\$ = S \wedge inv\,(S)\}$
$\Longrightarrow$ (NodeL9_4)
$\{inv\,(\$)\}$
———————————————————————————————↑–[ex-rule]
$\{inv\,(\$)\}$

7.10 Obligation 10

Let P resp. Q be the pre– resp. postcondition of Node's obligation 3.
Proof :
$\{typeof\,(this) = ct\,(NodeL) \wedge P\}$ Node@getElem() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof\,(this) < ct\,(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof\,(this) \leq ct\,(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$

Lemma 1:
$(typeof\,(this) = ct\,(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getElem() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof\,(this) = ct\,(NodeL) \wedge P\}$ Node@getElem() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof\,(this) < ct\,(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getElem() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof\,(this) < ct\,(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$

7.11 Obligation 11

Let P resp. Q be the pre– resp. postcondition of Node's obligation 4.

Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getElem() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getElem() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getElem() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getElem() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getElem() $\{Q\}$

7.12 Obligation 12

Let P resp. Q be the pre– resp. postcondition of Node's obligation 6.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getPred() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getPred() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getPred() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getPred() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$

7.13 Obligation 13

Let P resp. Q be the pre– resp. postcondition of Node's obligation 7.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getPred() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getPred() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getPred() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getPred() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getPred() $\{Q\}$

7.14 Obligation 14

Let P resp. Q be the pre– resp. postcondition of Node's obligation 9.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getSucc() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

7.15 Obligation 15

Let P resp. Q be the pre– resp. postcondition of Node's obligation 10.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getSucc() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

7.16 Obligation 16

Let P resp. Q be the pre– resp. postcondition of Node's obligation 11.

Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$ (Lemma 2)
————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ Node@getSucc() $\{Q\}$
————————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ Node@getSucc() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getSucc() { FALSE }
FALSE $\Rightarrow \{Q\}$
————————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getSucc() $\{Q\}$

7.17 Obligation 17

Let P resp. Q be the pre– resp. postcondition of NodeL's obligation 5.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@appback(NodeL n) $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$ (Lemma 2)
————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ NodeL@appback(NodeL n) $\{Q\}$
————————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@appback(NodeL n) $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:appback(NodeL n) { FALSE }
FALSE $\Rightarrow \{Q\}$
————————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$

7.18 Obligation 18

Let P resp. Q be the pre– resp. postcondition of NodeL's obligation 6.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@appback(NodeL n) $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$ (Lemma 2)
————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ NodeL@appback(NodeL n) $\{Q\}$
————————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@appback(NodeL n) $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:appback(NodeL n) { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:appback(NodeL n) $\{Q\}$

7.19 Obligation 19

Let P resp. Q be the pre– resp. postcondition of NodeL's obligation 8.
Proof :
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@getLast() $\{Q\}$ (Lemma 1)
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getLast() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(NodeL) \wedge P\}$ NodeL:getLast() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(NodeL) \wedge P) \Rightarrow P$
$\{P\}$ NodeL@getLast() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(NodeL) \wedge P\}$ NodeL@getLast() $\{Q\}$

Lemma 2:
$(this \neq null \wedge typeof(this) < ct(NodeL) \wedge P) \Rightarrow$ (NodeLVirt1) FALSE
{ FALSE } NodeL:getLast() { FALSE }
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{this \neq null \wedge typeof(this) < ct(NodeL) \wedge P\}$ NodeL:getLast() $\{Q\}$

8 Verification of Class DList

8.1 Obligation 1

Abreviations:
$S_1 := update(\$, loc(this, DList@firstNode), null)$

Proof:
$\{this \neq null \wedge alive(this, \$) \wedge T = this\}$
$\Longrightarrow$ (DList1_1)
$\{this \neq null \wedge ADList(T, S_1) = null\}$
`this.firstNode = null;`
$\{ADList(T, \$) = null\}$
`return 0;`
$\{ADList(T, \$) = null\}$

8.2 Obligation 2

Abreviations: Same as in obligation 1.

Proof:
$\{this \neq null \wedge alive(this, \$) \wedge inv(\$)\}$
$\Longrightarrow$ (DList2_1)
$\{inv(S_1)\}$
`this.firstNode = null;`
$\{inv(\$)\}$
`return 0;`
$\{inv(\$)\}$

8.3 Obligation 3

Lemmas:
1.
$\{\$ = S\}$
`DList@newDList();`
$\{\$ = S\#\#DList \wedge result = new(S, DList)\}$
———————————————————————————————[static-invocation-rule]
$\{\$ = S\}$
`DList d; d = DList.newDList();`
$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$

Proof:
$\{true\}$
$\Longrightarrow$ (DList3_1)
$\{\exists S : \$ = S\}$
———————————————————————————————↓–[ex-rule]
$\{\$ = S\}$
`DList d; d = DList.newDList(); // lemma`
$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$
$\Longrightarrow$ (DList3_2)
$\{ADList(d, \$) = null\}$
`return d;`
$\{ADList(result, \$) = null\}$
———————————————————————————————↑–[ex-rule]
$\{ADList(result, \$) = null\}$

8.4 Obligation 4

Lemmas: Same as in obligation 3.

Proof:

$\{\$ = S \wedge alive(X, S)\}$

———————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
DList d; d = DList.newDList(); // lemma
```

$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#DList \wedge d = new(S, DList) \wedge alive(X, S)\}$

$\Longrightarrow$ (DList4_2)

$\{(\$ == S)(X)\}$

```
return d;
```

$\{(\$ == S)(X)\}$

8.5 Obligation 5

Lemmas: Same as in obligation 3.

Proof:

$\{inv(\$)\}$

$\Longrightarrow$ (DList5_1)

$\{\exists S : \$ = S \wedge inv(\$)\}$

———————————————————————————————————↓–[ex-rule]

$\{\$ = S \wedge inv(\$)\}$

$\Longrightarrow$ (DList5_2)

$\{\$ = S \wedge inv(S)\}$

———————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
DList d; d = DList.newDList(); // lemma
```

$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#DList \wedge d = new(S, DList) \wedge inv(S)\}$

$\Longrightarrow$ (DList5_3))

$\{inv(\$)\}$

———————————————————————————————————↑–[ex-rule]

$\{inv(\$)\}$

```
return d;
```

$\{inv(\$)\}$

8.6 Obligation 6

Lemmas:

1.

$\{\$ = S \wedge T = this\}$

```
Node:getElem()
```

$\{aI(result) = ANode(T, S)\}$

———————————————————————————————————–[invocation-rule]

$\{f \neq null \wedge \$ = S \wedge T = f\}$

```
int k; k = f.getElem();
```

$\{aI(k) = ANode(T, S)\}$

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList(this, \$) \neq null \quad \wedge \\ ADList(this, \$) = L \end{array}\right\}$$

$\Longrightarrow$ (DList6_1)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ ADList(this, \$) = L \end{array}\right\}$$

$\Longrightarrow$ (DList6_2)

$$\left\{\begin{array}{l} \exists S, T : \\ S = \$ \quad \wedge \\ T = \$@@loc(this, DList?firstNode) \quad \wedge \\ this \neq null \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ ADList(this, S) = L \quad \wedge \\ T \neq null \end{array}\right\}$$

——————————————————————————————————↓–[ex-rule]

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = \$@@loc(this, DList?firstNode) \quad \wedge \\ ADList(this, S) = L \quad \wedge \\ T \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList6_3)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = \$@@loc(this, DList?firstNode) \quad \wedge \\ ADList(this, S) = L \quad \wedge \\ T = S@@loc(this, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

```
Node f; f = this.firstNode;
```

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, S) = L \quad \wedge \\ T = S@@loc(this, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

——————————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ T_1 \neq null \quad \wedge \\ ADList(T_1, S) = L \quad \wedge \\ T = S@@loc(T_1, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

——————————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \wedge T = f \end{array}\right\}$$

```
int k; k = f.getElem();  // lemma
```

$$\{\ aI(k) = ANode(T, S)\ \}$$

———————————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} aI(k) = ANode(T,S) \quad \wedge \\ T_1 \neq null \quad \wedge \\ ADList(T_1,S) = L \quad \wedge \\ T = S@@loc(T_1, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

———————————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} aI(k) = ANode(T,S) \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this,S) = L \quad \wedge \\ T = S@@loc(this, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList6_4)

$\{\ aI(k) = car(L)\ \}$

```
return k;
```

$\{\ aI(result) = car(L)\ \}$

———————————————————————————————————↑–[ex-rule]

$\{\ aI(result) = car(L)\ \}$

8.7 Obligation 7

Lemmas:

1.

$\{S = \$\}$

```
Node:getElem()
```

$\{\$ = S\}$

———————————————————————————————————–[invocation-rule]

$\{S = \$ \wedge f \neq null\}$

```
int k; k = f.getElem();
```

$\{\$ = S\}$

Proof:

$\{this \neq null \wedge ADList(this,\$) \neq null \wedge S = \$\}$

$\Longrightarrow$ (DList7_1)

$\{this \neq null \wedge S = \$ \wedge \$@@loc(this, DList?firstNode) \neq null\}$

```
Node f; f = this.firstNode;
```

$\{S = \$ \wedge f \neq null\}$

```
int k; k = f.getElem();  // lemma
```

$\{\$ = S\}$

```
return k;
```

$\{\$ = S\}$

8.8 Obligation 8

Proof: (it is used, that the last specification is already proved)

$\{this \neq null \wedge ADList(this,\$) \neq null \wedge inv(\$)\}$

$\Longrightarrow$ (DList8_1)

$\{\exists S : \$ = S \wedge this \neq null \wedge ADList(this,\$) \neq null \wedge inv(\$)\}$

———————————————————————————————————↓–[ex-rule]

$\{\$ = S \wedge this \neq null \wedge ADList(this,\$) \neq null \wedge inv(\$)\}$

$\Longrightarrow$ (DList8_2)

$\{\$ = S \wedge this \neq null \wedge ADList(this,\$) \neq null \wedge inv(S)\}$

———————————————————————————————————↓–[inv-rule]

$\{this \neq null \wedge ADList(this,\$) \neq null \wedge S = \$\}$

`DList@first()`

$\{\$ = S\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S \wedge inv\,(S)\}$

$\Longrightarrow$ (DList8_3)

$\{inv\,(\$)\}$

———————————————————————————————————↑–[ex-rule]

$\{inv\,(\$)\}$

8.9 Obligation 9

Abreviations:

$S_1 := update\,(\$, loc\,(d, DList?lastNode\,), l)$

$S_2 := update\,(update\,(\$, loc\,(d, DList?firstNode\,), f), loc\,(d, DList?lastNode\,), l)$

Lemmas:

1.

$\{\$ = S\}$

`DList@newDList();`

$\{\$ = S\#\#DList \wedge result\, = new\,(S, DList\,)\}$

———————————————————————————————————–[static-invocation-rule]

$\{\$ = S\}$

`DList d; d = DList.newDList();`

$\{\$ = S\#\#DList \wedge d = new\,(S, DList\,)\}$

Proof:

$\{wf2Nodes\,(f, l, \$) \wedge S = \$ \wedge alive\,(X, S)\}$

$\Longrightarrow$ (DList9_1)

$\{\$ = S \wedge alive\,(X, S)\}$

———————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

`DList d; d = DList.newDList();`

$\{\$ = S\#\#DList \wedge d = new\,(S, DList\,)\}$

———————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#DList \wedge d = new\,(S, DList\,) \wedge alive\,(X, S)\}$

$\Longrightarrow$ (DList9_2)

$\{d \neq null \wedge (S == S_2)(X)\}$

`d.firstNode = f;`

$\{d \neq null \wedge (S == S_1)(X)\}$

`d.lastNode = l;`

$\{(S == \$)(X)\}$

`return d;`

$\{(S == \$)(X)\}$

8.10 Obligation 10

Abreviations and lemmas: same as in obligation 9.

Proof:

$\{\ S = \$ \wedge F = f \wedge wf2Nodes\,(f, l, \$) \wedge\ alive(f, \$) \wedge invNodeL\,(\$)\ \}$

$\Longrightarrow$ (DList10_1)

$\{S = \$ \wedge F = f \wedge alive\,(F, S) \wedge invNodeL\,(S)\}$

———————————————————————————————————↓–[var-rule]

$\{S = \$ \wedge F = F' \wedge alive\,(F, S) \wedge invNodeL\,(S)\}$

———————————————————————————————————————↓–[inv-rule]

$\{S = \$\}$

```
DList d; d = DList.newDList();
```

$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$

———————————————————————————————————————↑–[inv-rule]

$\{\$ = S\#\#DList \wedge d = new(S, DList) \wedge F = F' \wedge alive(F, S) \wedge invNodeL(S)\}$

———————————————————————————————————————↑–[var-rule]

$\{\$ = S\#\#DList \wedge d = new(S, DList) \wedge F = f \wedge alive(F, S) \wedge invNodeL(S)\}$

$\Longrightarrow$ (DList10_2)

$\{d \neq null \wedge ADList(d, S_2) = ANodeL(F, S)\}$

```
d.firstNode = f
```

$\{d \neq null \wedge ADList(d, S_1) = ANodeL(F, S)\}$

```
d.lastNode = l;
```

$\{ADList(d, \$) = ANodeL(F, S)\}$

```
return d;
```

$\{ADList(result, \$) = ANodeL(F, S)\}$

8.11 Obligation 11

Abreviations and lemmas: same as in obligation 9.

Proof:

$\{\ wf2Nodes(f, l, \$) \wedge alive(f, \$) \wedge alive(l, \$) \wedge inv(\$)\ \}$

$\Longrightarrow$ (DList11_1)

$\left\{ \begin{array}{l} \exists S : \$ = S \quad \wedge \\ wf2Nodes(f, l, \$) \wedge alive(f, \$) \wedge alive(l, \$) \wedge inv(\$) \end{array} \right\}$

———————————————————————————————————————↓–[ex-rule]

$\{\ wf2Nodes(f, l, \$) \wedge \$ = S \wedge alive(f, \$) \wedge alive(l, \$) \wedge inv(\$)\ \}$

$\Longrightarrow$ (DList11_2)

$\{\ wf2Nodes(f, l, S) \wedge \$ = S \wedge alive(f, S) \wedge alive(l, S) \wedge inv(S)\ \}$

———————————————————————————————————————↓–[var-rule]

$\{\ wf2Nodes(F, L, S) \wedge \$ = S \wedge alive(F, S) \wedge alive(L, S) \wedge inv(S)\ \}$

———————————————————————————————————————↓–[inv-rule]

$\{\$ = S\}$

```
DList d; d = DList.newDList();
```

$\{\$ = S\#\#DList \wedge d = new(S, DList)\}$

———————————————————————————————————————↑–[inv-rule]

$\left\{ \begin{array}{l} \$ = S\#\#DList \wedge d = new(S, DList) \wedge wf2Nodes(F, L, S) \quad \wedge \\ alive(F, S) \wedge alive(L, S) \wedge inv(S) \end{array} \right\}$

———————————————————————————————————————↑–[var-rule]

$\left\{ \begin{array}{l} \$ = S\#\#DList \wedge d = new(S, DList) \wedge wf2Nodes(f, l, S) \quad \wedge \\ alive(f, S) \wedge alive(l, S) \wedge inv(S) \end{array} \right\}$

$\Longrightarrow$ (DList11_3)

$\{d \neq null \wedge inv(S_2)\}$

———————————————————————————————————————↑–[ex-rule]

$\{d \neq null \wedge inv(S_2)\}$

```
d.firstNode = f;
```

$\{d \neq null \wedge inv(S_1)\}$

```
d.lastNode = l;
```

$\{inv(\$)\}$

```
return d;
```

$\{inv(\$)\}$

8.12 Obligation 12

Lemmas:

1.

$$\frac{\begin{array}{ll}\{\$ = S \wedge T = this\} & \texttt{Node:getSucc()}\ \{result = S@@loc(T, Node?succ)\}\\ \{\$ = S\} & \texttt{Node:getSucc()}\ \{\$ = S\}\\ \{ANodeL(this, \$) = L\} & \texttt{Node:getSucc()}\ \{ANodeL(result, \$) = cdr(L)\}\end{array}}{\begin{array}{l}\{\$ = S \wedge T = this \wedge ANodeL(this, \$) = L\}\\ \texttt{Node@getSucc()}\\ \{\$ = S \wedge result = S@@loc(T, Node?succ) \wedge ANodeL(result, \$) = cdr(L)\}\end{array}}\text{[conjunct-rule]}$$

$$\frac{\begin{array}{l}\{\$ = S \wedge T = this \wedge ANodeL(this, \$) = L\}\\ \texttt{Node@getSucc()}\\ \{\$ = S \wedge result = S@@loc(T, Node?succ) \wedge ANodeL(result, \$) = cdr(L)\}\end{array}}{\begin{array}{l}\{f \neq null \wedge \$ = S \wedge T = f \wedge ANodeL(f, \$) = L \wedge\}\\ \texttt{Node s; s = f.getSucc();}\\ \{\$ = S \wedge s = S@@loc(T, Node?succ) \wedge ANodeL(s, \$) = cdr(L)\}\end{array}}\text{[invocation-rule]}$$

2.

$$\frac{\begin{array}{l}\{T = this \wedge \$ = S\}\\ \texttt{Node:getLast()}\\ \{\$ = S \wedge result \neq null \wedge (\exists N : succn(T, S, N) = result \wedge succn(T, S, N+1) = null)\}\end{array}}{\begin{array}{l}\{f \neq null \wedge T = f \wedge \$ = S\}\\ \texttt{NodeL l; l = f.getLast();}\\ \{\$ = S \wedge l \neq null \wedge (\exists N : succn(T, S, N) = l \wedge succn(T, S, N+1) = null)\}\end{array}}\text{[invocation-rule]}$$

3.

$$\frac{\begin{array}{l}\{wf2Nodes(f, l, \$) \wedge \$ = S \wedge F = f\}\\ \texttt{DList@create(NodeL f, NodeL l)}\\ \{ADList(result, \$) = ANodeL(F, S)\}\end{array}}{\begin{array}{l}\{wf2Nodes(s, l, \$) \wedge \$ = S \wedge F = s\}\\ \texttt{DList d; d = DList.create(s,l);}\\ \{ADList(d, \$) = ANodeL(F, S)\}\end{array}}\text{[static-invocation-rule]}$$

Proof:

$$\frac{\left\{\begin{array}{ll} this \neq null & \wedge\\ ADList(this, \$) \neq null & \wedge\\ ADList(this, \$) = L & \end{array}\right\}}{\begin{array}{l}\Longrightarrow \text{(DList12\_1)}\\ \left\{\begin{array}{ll} \exists S : \$ = S & \wedge\\ this \neq null & \wedge\\ \$@@loc(this, DList?firstNode) \neq null & \wedge\\ ANodeL(\$@@loc(this, DList?firstNode), \$) = L & \wedge\\ L \neq null & \end{array}\right\}\end{array}}\downarrow\text{-[ex-rule]}$$

$$\left\{\begin{array}{ll} this \neq null & \wedge\\ \$@@loc(this, DList?firstNode) \neq null & \wedge\\ \$ = S & \wedge\\ ANodeL(\$@@loc(this, DList?firstNode), \$) = L & \wedge\\ L \neq null & \end{array}\right\}$$

```
NodeL f; f = this.firstNode;
```

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ ANodeL(f, \$) = L \quad \wedge \\ L \neq null \end{array}\right\}$$

$\downarrow$–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ ANodeL(f, \$) = L \quad \wedge \\ T \neq null \quad \wedge \\ L \neq null \end{array}\right\}$$

$\downarrow$–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ ANodeL(f, \$) = L \end{array}\right\}$$

```
Node n; n = f.getSucc(); // lemma 1
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \quad \wedge \\ ANodeL(n, \$) = cdr(L) \end{array}\right\}$$

$\Longrightarrow$

$$\left\{\begin{array}{l} ct(NodeL) \leq typeof(n) \quad \wedge \\ \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \quad \wedge \\ ANodeL(n, \$) = cdr(L) \end{array}\right\}$$

```
NodeL s; s = (NodeL) n;
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \quad \wedge \\ ANodeL(s, \$) = cdr(L) \end{array}\right\}$$

$\uparrow$–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \quad \wedge \\ ANodeL(s, \$) = cdr(L) \quad \wedge \\ T \neq null \quad \wedge \\ L \neq null \end{array}\right\}$$

$\uparrow$–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ ANodeL(s, \$) = cdr(L) \quad \wedge \\ f \neq null \quad \wedge \\ L \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList12_2)

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ ANodeL(s, S) = cdr(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc\,(T, Node?succ\,) \quad \wedge \\ ANodeL\,(T_2, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

——————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \end{array}\right\}$$

```
NodeL l; l = f.getLast(); // lemma 2
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn\,(T, S, N) = l \wedge succn\,(T, S, N+1) = null \end{array}\right\}$$

——————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn\,(T, S, N) = l \wedge succn\,(T, S, N+1) = null \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc\,(T, Node?succ\,) \quad \wedge \\ ANodeL\,(T_2, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

——————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn\,(f, S, N) = l \wedge succn\,(f, S, N+1) = null \quad \wedge \\ f \neq null \quad \wedge \\ s = S@@loc\,(f, Node?succ\,) \quad \wedge \\ ANodeL\,(s, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList12_3)

$$\left\{\begin{array}{l} wf2Nodes\,(s, l, \$) \quad \wedge \\ \$ = S \quad \wedge \\ ANodeL\,(s, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

——————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} wf2Nodes\,(s, l, \$) \quad \wedge \\ \$ = S \quad \wedge \\ F = s \quad \wedge \\ ANodeL\,(F, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

——————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} wf2Nodes\,(s, l, \$) \quad \wedge \\ \$ = S \quad \wedge \\ F = s \end{array}\right\}$$

```
DList d; d = DList.create(s,l);  // lemma 3
```

$\{\ ADList\,(d, \$) = ANodeL\,(F, S)\ \}$

——————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} ADList\,(d, \$) = ANodeL\,(F, S) \quad \wedge \\ ANodeL\,(F, S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\}$$

$$\frac{\left\{\begin{array}{l} ADList\,(d,\$) = ANodeL\,(s,S) \quad \wedge \\ ANodeL\,(s,S) = cdr\,(L) \quad \wedge \\ L \neq null \end{array}\right\} \Longrightarrow \text{(DList12\_4)} \quad \{\ ADList\,(d,\$) = cdr\,(L)\ \}}{\{\ ADList\,(d,\$) = cdr\,(L)\ \}}\uparrow\text{–[var-rule]}$$

```
return d;
```

$$\{\ ADList\,(result,\$) = cdr\,(L)\ \}$$

$$\frac{\{\ ADList\,(result,\$) = cdr\,(L)\ \}}{\{\ ADList\,(result,\$) = cdr\,(L)\ \}}\uparrow\text{–[ex-rule]}$$

8.13 Obligation 13

Lemmas:

1.

$$\frac{\begin{array}{ll} \{\$ = S \wedge T = this\} & \text{Node:getSucc() } \{result = S@@loc\,(T, Node?succ)\} \\ \{\$ = S\} & \text{Node:getSucc() } \{\$ = S\} \end{array}}{\{\$ = S \wedge T = this\}}\text{[conjunct-rule]}$$

```
Node:getSucc()
```

$$\frac{\{\$ = S \wedge result = S@@loc\,(T, Node?succ)\}}{\{f \neq null \wedge \$ = S \wedge T = f\}}\text{[invocation-rule]}$$

```
NodeL s; s = f.getSucc();
```

$$\{\$ = S \wedge s = S@@loc\,(T, Node?succ)\}$$

2.

$$\{T = this \wedge \$ = S\}$$

```
Node:getLast()
```

$$\frac{\{\$ = S \wedge result \neq null \wedge (\exists N : succn\,(T,S,N) = result \wedge succn\,(T,S,N+1) = null)\}}{\{f \neq null \wedge T = f \wedge \$ = S\}}\text{[invocation-rule]}$$

```
NodeL l; l = f.getLast();
```

$$\{\$ = S \wedge l \neq null \wedge (\exists N : succn\,(T,S,N) = l \wedge succn\,(T,S,N+1) = null)\}$$

3.

$$\{wf2Nodes\,(f,l,\$) \wedge alive\,(X,\$) \wedge S = \$\}$$

```
DList@create(NodeL f, NodeL l)
```

$$\frac{\{(S == \$)(X)\}}{\{wf2Nodes\,(s,l,\$) \wedge alive\,(X,\$) \wedge S = \$\}}\text{[static-invocation-rule]}$$

```
DList d; d = create(s,l);
```

$$\{(S == \$)(X)\}$$

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList\,(this\,,\$) \neq null \quad \wedge \\ alive\,(X,\$) \quad \wedge \\ \$ = S \end{array}\right\} \Longrightarrow \text{(DList13\_1)}$$

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ \$ = S \quad \wedge \\ alive(X, S) \end{array}\right\}$$

```
NodeL f; f = this.firstNode;
```

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ alive(X, S) \end{array}\right\}$$

———————————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ T \neq null \quad \wedge \\ alive(X, S) \end{array}\right\}$$

———————————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \end{array}\right\}$$

```
Node n; n = f.getSucc();  // lemma 1
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \end{array}\right\}$$

$$\Longrightarrow$$

$$\left\{\begin{array}{l} ct(NodeL) \leq typeof(n) \quad \wedge \\ \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \end{array}\right\}$$

```
NodeL s; s = (NodeL) n;
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \end{array}\right\}$$

———————————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \quad \wedge \\ T \neq null \quad \wedge \\ alive(X, S) \end{array}\right\}$$

———————————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ f \neq null \quad \wedge \\ alive(X, S) \end{array}\right\}$$

$\Longrightarrow$ (DList13_2)

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ alive(X, S) \end{array}\right\}$$

———————————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc(T, Node?succ) \quad \wedge \\ alive(X, S) \end{array}\right\}$$

———————————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \end{array}\right\}$$

```
NodeL l; l = f.getLast();  // lemma 2
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(T,S,N) = l \wedge succn(T,S,N+1) = null \end{array}\right\}$$

———————————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(T,S,N) = l \wedge succn(T,S,N+1) = null \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc(T, Node?succ) \quad \wedge \\ alive(X,S) \end{array}\right\}$$

———————————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(f,S,N) = l \wedge succn(f,S,N+1) = null \quad \wedge \\ f \neq null \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ alive(X,S) \end{array}\right\}$$

$\Longrightarrow$ (DList13_3)

$$\left\{\begin{array}{l} wf2Nodes(s,l,\$) \quad \wedge \\ \$ = S \quad \wedge \\ alive(X,S) \end{array}\right\}$$

```
DList d; d = DList.create(s, l);  // lemma 3
```

$\{\ (S == \$)(X)\ \}$

$\Longrightarrow$ (DList13_4)

$\{\ (\$ == S)(X)\ \}$

8.14 Obligation 14

Lemmas:

1.

$\{\$ = S \wedge T = this\}$ Node:getSucc() $\{result = S@@loc(T, Node?succ)\}$

$\{\$ = S\}$ Node:getSucc() $\{\$ = S\}$

$\{inv(\$)\}$ Node:getSucc() $\{inv(\$)\}$

———————————————————————————————————[conjunct-rule]

$\{\$ = S \wedge T = this \wedge inv(\$)\}$

```
Node:getSucc()
```

$\{\$ = S \wedge result = S@@loc(T, Node?succ) \wedge inv(\$)\}$

———————————————————————————————————[invocation-rule]

$\{f \neq null \wedge \$ = S \wedge T = f \wedge inv(\$)\}$

```
Node s; s = f.getSucc();
```

$\{\$ = S \wedge s = S@@loc(T, Node?succ) \wedge inv(\$)\}$

2.

$\{T = this \wedge \$ = S\}$

```
Node:getLast()
```

$$\left\{\begin{array}{l} \$ = S \wedge result \neq null \quad \wedge \\ (\exists N : succn(T,S,N) = result \wedge succn(T,S,N+1) = null) \end{array}\right\}$$

$\{inv(\$)\}$

```
Node:getLast()
```

$\{inv(\$)\}$

———————————————————————————————————[conjunct-rule]

$\{\ T = this \wedge \$ = S \wedge inv(\$)\ \}$

`Node:getLast()`

$$\left\{\begin{array}{l} inv(\$) \wedge \$ = S \wedge result \neq null \quad \wedge \\ (\exists N : succn(T, S, N) = result \wedge succn(T, S, N+1) = null) \end{array}\right\}$$

———————————————————————————————[invocation-rule]

$\{f \neq null \wedge T = f \wedge \$ = S \wedge inv(\$)\}$

`NodeL l; l = f.getLast();`

$$\left\{\begin{array}{l} inv(\$) \wedge \$ = S \wedge l \neq null \quad \wedge \\ (\exists N : succn(T, S, N) = l \wedge succn(T, S, N+1) = null) \end{array}\right\}$$

3.

$\{wf2Nodes(f, l, \$) \wedge inv(\$)\}$

`DList@create(NodeL f, NodeL l)`

$\{inv(\$)\}$

———————————————————————————————[static-invocation-rule]

$\{wf2Nodes(s, l, \$) \wedge inv(\$)\}$

`DList d; d = DList.create(s, l);`

$\{inv(\$)\}$

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList(this, \$) \neq null \quad \wedge \\ inv(\$) \end{array}\right\}$$

$\Longrightarrow$ (DList14_1)

$$\left\{\begin{array}{l} \exists S : \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ inv(\$) \end{array}\right\}$$

———————————————————————————————↓–[ex-rule]

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \end{array}\right\}$$

`NodeL f; f = this.firstNode;`

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \end{array}\right\}$$

———————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ inv(\$) \quad \wedge \\ T \neq null \end{array}\right\}$$

———————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ T = f \quad \wedge \\ inv(\$) \end{array}\right\}$$

`Node n; n = f.getSucc();  // lemma 1`

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \quad \wedge \\ inv(\$) \end{array}\right\}$$

$\Longrightarrow$

$$\left\{\begin{array}{l} ct(NodeL) \leq typeof(n) \quad \wedge \\ \$ = S \quad \wedge \\ n = S@@loc(T, Node?succ) \quad \wedge \\ inv(\$) \end{array}\right\}$$

```
NodeL s; s = (NodeL) n;
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \quad \wedge \\ inv(\$) \end{array}\right\}$$

---------- ↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(T, Node?succ) \quad \wedge \\ inv(\$) \quad \wedge \\ T \neq null \end{array}\right\}$$

---------- ↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ s = S@@loc(f, Node?succ) \quad \wedge \\ inv(\$) \quad \wedge \\ f \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList14_2)

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ s = S@@loc(f, Node?succ) \end{array}\right\}$$

---------- ↓–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc(T, Node?succ) \end{array}\right\}$$

---------- ↓–[inv-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T = f \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \end{array}\right\}$$

```
NodeL l; l = f.getLast();  // lemma 2
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(T, S, N) = l \wedge succn(T, S, N+1) = null \quad \wedge \\ inv(\$) \end{array}\right\}$$

---------- ↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(T, S, N) = l \wedge succn(T, S, N+1) = null \quad \wedge \\ inv(\$) \quad \wedge \\ T \neq null \quad \wedge \\ T_2 = S@@loc(T, Node?succ) \end{array}\right\}$$

---------- ↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ l \neq null \quad \wedge \\ \exists N : succn(f, S, N) = l \wedge succn(f, S, N+1) = null \quad \wedge \\ inv(\$) \quad \wedge \\ f \neq null \quad \wedge \\ s = S@@loc(f, Node?succ) \end{array}\right\}$$

$\Longrightarrow$ (DList14_3)

$\left\{ \begin{array}{ll} wf2Nodes\,(s,l,\$) & \wedge \\ inv\,(\$) & \end{array} \right\}$

```
DList d; d = DList.create(s, l);  // lemma 3
```

$\{\ inv\,(\$)\ \}$

```
return d;
```

$\{\ inv\,(\$)\ \}$

———↑–[ex-rule]

$\{\ inv\,(\$)\ \}$

8.15 Obligation 15

Lemmas:

1.

$\{A = a \wedge B = b \wedge S = \$\}$

```
Operator@equal(Object a, Object b)
```

$\{result\ = boolV\,(A = B) \wedge \$ = S\}$

———[static-invocation-rule]

$\{A = f \wedge B = null \wedge S = \$\}$

```
boolean b; b = Operator.equal(f,null);
```

$\{b = boolV\,(A = B) \wedge \$ = S\}$

Proof:

$\{this\ \neq null \wedge L = ADList\,(this\,,\$)\}$

$\Longrightarrow$ (DList15_1)

$\{\exists S : \$ = S \wedge this\ \neq null \wedge L = ADList\,(this\,,\$)\}$

———↓–[ex-rule]

$\{\ S = \$ \wedge this\ \neq null \wedge L = ADList\,(this\,,\$)\ \}$

$\Longrightarrow$ (DList15_2)

$\left\{ \begin{array}{l} S = \$ \wedge this\ \neq null \wedge L = ADList\,(this\,,S) \quad \wedge \\ \$@@loc\,(this\,,DList?firstNode) = S@@loc\,(this\,,DList?firstNode) \end{array} \right\}$

```
NodeL f; f = this.firstNode;
```

$\{S = \$ \wedge L = ADList\,(this\,,S) \wedge f = S@@loc\,(this\,,DList?firstNode)\}$

———↓–[var-rule]

$\left\{ \begin{array}{ll} A = f \wedge B = null \wedge S = \$ \wedge L = ADList\,(T,S) & \wedge \\ A = S@@loc\,(T,DList?firstNode) & \end{array} \right\}$

———↓–[inv-rule]

$\{A = f \wedge B = null \wedge S = \$\}$

```
boolean b; b = Operator.equal(f,null);  // lemma
```

$\{b = boolV\,(A = B) \wedge \$ = S\}$

———↑–[inv-rule]

$\left\{ \begin{array}{ll} b = boolV\,(A = B) \wedge \$ = S \wedge L = ADList\,(T,S) & \wedge \\ A = S@@loc\,(T,DList?firstNode) & \end{array} \right\}$

———↑–[var-rule]

$\left\{ \begin{array}{ll} b = boolV\,(f = null\,) \wedge \$ = S \wedge L = ADList\,(this\,,S) & \wedge \\ f = S@@loc\,(this\,,DList?firstNode) & \end{array} \right\}$

$\Longrightarrow$ (DList15_3)

$\{aB\,(b) = null?\,(L)\}$

```
return b;
```

$\{aB\,(result) = null?\,(L)\}$

———↑–[ex-rule]

$\{aB\,(result) = null?\,(L)\}$

8.16 Obligation 16

Lemmas: same as in the last obligation.
Proof:
$\{this \neq null \wedge S = \$\}$
`NodeL f; f = this.firstNode;`
$\{S = \$\}$
———————————————————————————————————↓–[var-rule]
$\{A = f \wedge B = null \wedge S = \$\}$
`boolean b; b = Operator.equal(f,null);`
$\{b = boolV(A = B) \wedge \$ = S\}$
———————————————————————————————————↑–[var-rule]
$\{b = boolV(f = null) \wedge \$ = S\}$
$\Longrightarrow$ (DList16_1)
$\{\$ = S\}$
`return b;`
$\{\$ = S\}$

8.17 Obligation 17

Lemmas: same as in the last obligation.
Proof:
$\{this \neq null \wedge inv(\$)\}$
$\Longrightarrow$ (DList17_1)
$\{\exists S : \$ = S \wedge this \neq null \wedge inv(\$)\}$
———————————————————————————————————↓–[ex-rule]
$\{\$ = S \wedge this \neq null \wedge inv(\$)\}$
$\Longrightarrow$ (DList17_2)
$\{\$ = S \wedge this \neq null \wedge inv(S)\}$
———————————————————————————————————↓–[inv-rule]
$\{this \neq null \wedge \$ = S\}$
`DList@isempty()`
$\{\$ = S\}$
———————————————————————————————————↑–[inv-rule]
$\{\$ = S \wedge inv(S)\}$
$\Longrightarrow$ (DList17_3)
$\{inv(\$)\}$
———————————————————————————————————↑–[ex-rule]
$\{inv(\$)\}$

8.18 Obligation 18

Lemmas:
1.
$\{\$ = S\}$
`NodeL@initNodeL(int i)`
$\{\ \$ = update(S\#\#NodeL, loc(result, Node?elem), i) \wedge result = new(S, NodeL)\ \}$
$\{inv(\$)\}$
`NodeL@initNodeL(int i)`
$\{inv(\$)\}$
———————————————————————————————————[conjunct-rule]
$\{\$ = S \wedge inv(\$)\}$

`NodeL@initNodeL(int i)`
$\{\$ = update(S\#\#NodeL, loc(l, Node?elem), i) \wedge l = new(S, NodeL) \wedge inv(\$)\}$

[static-invocation-rule]

$\{\$ = S \wedge inv(\$)\}$
`NodeL l; l = NodeL.initNodeL(i);`
$\{\$ = update(S\#\#NodeL, loc(l, Node?elem), i) \wedge l = new(S, NodeL) \wedge inv(\$)\}$

2.

$\{\$ = S\}$ DList@isempty() $\{\$ = S\}$
$\{ADList(this, \$) = L\}$ DList@isempty() $\{aB(result) = null?(L)\}$

[conjunct-rule]

$\{\$ = S \wedge ADList(this, \$) = L\}$
`DList@isempty()`
$\{\$ = S \wedge aB(result) = null?(L)\}$

[invocation-rule]

$\{this \neq null \wedge \$ = S \wedge ADList(this, \$) = L\}$
`boolean b; b = this.isempty();`
$\{\$ = S \wedge aB(b) = null?(L)\}$

3.

$\{T = this \wedge \$ = S\}$
`NodeL@getLast()`
$\{\$ = S \wedge result \neq null \wedge (\exists N : succn(T, S, N) = result \wedge succn(T, S, N+1) = null)\}$

[invocation-rule]

$\{f \neq null \wedge T = f \wedge \$ = S\}$
`NodeL h; h = f.getLast();`
$\{\$ = S \wedge h \neq null \wedge (\exists N : succn(T, S, N) = h \wedge succn(T, S, N+1) = null)\}$

[subst-rule]

$\{f \neq null \wedge T_1 = f \wedge \$ = S\}$
`NodeL h; h = f.getLast();`
$\{\$ = S \wedge h \neq null \wedge (\exists N : succn(T_1, S, N) = h \wedge succn(T_1, S, N+1) = null)\}$

4.

$\left\{ \begin{array}{l} n \neq null \wedge n \neq this \wedge fstNode?(n, \$) \wedge lstNode?(n, \$) \wedge lstNode?(this, \$) \quad \wedge \\ succn(X, \$, k) = this \wedge ANodeL(X, \$) = L \wedge ANodeL(n, \$) = M \end{array} \right\}$
`NodeL@appback(NodeL n)`
$\{\ ANodeL(X, \$) = append(L, M)\ \}$

$\left\{ \begin{array}{l} n \neq null \wedge n \neq this \wedge fstNode?(n, \$) \wedge lstNode?(n, \$) \wedge lstNode?(this, \$) \quad \wedge \\ T = this \wedge N = n \wedge \$ = S \end{array} \right\}$
`NodeL@appback(NodeL n)`
$\{\ \$ = update(update(S, loc(T, Node?succ), N), loc(N, Node?pred), T)\ \}$

$\left\{ \begin{array}{l} n \neq null \wedge n \neq this \wedge fstNode?(n, \$) \wedge lstNode?(n, \$) \wedge lstNode?(this, \$) \quad \wedge \\ inv(\$) \end{array} \right\}$
`NodeL@appback(NodeL n)`
$\{\ inv(\$)\ \}$

[conjunct-rule]

$\left\{ \begin{array}{l} n \neq null \wedge n \neq this \wedge fstNode?(n, \$) \wedge lstNode?(n, \$) \wedge lstNode?(this, \$) \quad \wedge \\ succn(X, \$, k) = this \wedge ANodeL(X, \$) = L \wedge ANodeL(n, \$) = M \quad \wedge \\ T = this \wedge N = n \wedge \$ = S \wedge inv(\$) \end{array} \right\}$
`NodeL@appback(NodeL n)`
$\left\{ \begin{array}{l} ANodeL(X, \$) = append(L, M) \quad \wedge \\ \$ = update(update(S, loc(T, Node?succ), N), loc(N, Node?pred), T) \quad \wedge \\ inv(\$) \end{array} \right\}$

$$\left\{\begin{array}{l} h \neq null \wedge l \neq null \wedge l \neq h \wedge fstNode?(l,\$) \wedge lstNode?(l,\$) \wedge lstNode?(h,\$) \quad \wedge \\ succn(X,\$,k) = h \wedge ANodeL(X,\$) = L \wedge ANodeL(l,\$) = M \quad \wedge \\ T = h \wedge N = l \wedge \$ = S \wedge inv(\$) \end{array}\right\}$$

```
int k; k = h.appback(l);
```

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(L,M) \quad \wedge \\ \$ = update(update(S, loc(T, Node?succ), N), loc(N, Node?pred), T) \quad \wedge \\ inv(\$) \end{array}\right\}$$

---[subst-rule]

$$\left\{\begin{array}{l} h \neq null \wedge l \neq null \wedge l \neq h \wedge fstNode?(l,\$) \wedge lstNode?(l,\$) \wedge lstNode?(h,\$) \quad \wedge \\ succn(X,\$,k) = h \wedge ANodeL(X,\$) = L_1 \wedge ANodeL(l,\$) = M \quad \wedge \\ H = h \wedge L_2 = l \wedge \$ = S \wedge inv(\$) \end{array}\right\}$$

```
int k; k = h.appback(l);
```

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(L_1,M) \quad \wedge \\ \$ = update(update(S, loc(H, Node?succ), L_2), loc(L_2, Node?pred), H) \quad \wedge \\ inv(\$) \end{array}\right\}$$

Proof:

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ alive(this,\$) \quad \wedge \\ ADList(this,\$) = L \quad \wedge \\ I = aI(i) \quad \wedge \\ T = this \quad \wedge \\ inv(\$) \end{array}\right\}$$

$\Longrightarrow$ (DList18_1)

$$\left\{\begin{array}{l} \exists S1 : \$ = S1 \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ T = this \quad \wedge \\ I = aI(i) \quad \wedge \\ L = ADList(this,\$) \quad \wedge \\ alive(this,S1) \end{array}\right\}$$

---↓-[ex-rule]

$$\left\{\begin{array}{l} \$ = S1 \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ T = this \quad \wedge \\ I = aI(i) \quad \wedge \\ L = ADList(this,\$) \quad \wedge \\ alive(this,S1) \end{array}\right\}$$

$\Longrightarrow$ (DList18_2)

$$\left\{\begin{array}{l} \$ = S1 \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ T = this \quad \wedge \\ I = aI(i) \quad \wedge \\ L = ADList(this,S1) \quad \wedge \\ alive(this,S1) \quad \wedge \\ inv(S1) \end{array}\right\}$$

$$\left\{\begin{array}{l} \$ = S1 \quad \wedge \\ inv(\$) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T = T_1 \quad \wedge \\ I = aI(I_1) \quad \wedge \\ L = ADList(T_1, S1) \quad \wedge \\ alive(T_1, S1) \quad \wedge \\ inv(S1) \end{array}\right\}$$

——————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} \$ = S1 \quad \wedge \\ inv(\$) \end{array}\right\}$$

——————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \end{array}\right\}$$

——————————————————————————↓–[subst-rule]

```
NodeL l; l = NodeL.initNodeL(i);  // lemma 1
```

$$\left\{\begin{array}{l} \$ = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S, NodeL) \quad \wedge \\ inv(\$) \end{array}\right\}$$

——————————————————————————↑–[subst-rule]

$$\left\{\begin{array}{l} \$ = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ inv(\$) \end{array}\right\}$$

——————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ inv(\$) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T = T_1 \quad \wedge \\ I = aI(I_1) \quad \wedge \\ L = ADList(T_1, S1) \quad \wedge \\ alive(T_1, S1) \quad \wedge \\ inv(S1) \end{array}\right\}$$

——————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ T = this \quad \wedge \\ I = aI(i) \quad \wedge \\ L = ADList(this, S1) \quad \wedge \\ alive(this, S1) \quad \wedge \\ inv(S1) \end{array}\right\}$$

$\Longrightarrow$ (DList18_3)

$$\left\{\begin{array}{l} \exists S : \$ = S \quad \wedge \\ S = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ T = this \quad \wedge \\ I = aI(i) \quad \wedge \\ inv(S) \quad \wedge \\ L = ADList(this, S) \quad \wedge \\ alive(this, S) \end{array}\right\}$$

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
S = update\,(S1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\
this \neq null \quad \wedge \\
ADList\,(this\,, \$) = L \quad \wedge \\
l = new\,(S1, NodeL\,) \quad \wedge \\
T = this \quad \wedge \\
I = aI\,(i) \quad \wedge \\
inv\,(S) \quad \wedge \\
L = ADList\,(this\,, S) \quad \wedge \\
alive\,(this\,, S)
\end{array}
\right\}
\quad \downarrow\text{-[ex-rule]}
$$

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
S = update\,(S1\#\#NodeL\,, loc\,(L_1, Node?elem\,), I_1) \quad \wedge \\
this \neq null \quad \wedge \\
ADList\,(this\,, \$) = L \quad \wedge \\
L_1 = new\,(S1, NodeL\,) \quad \wedge \\
T = T_1 \quad \wedge \\
T_1 \neq null \quad \wedge \\
I = aI\,(I_1) \quad \wedge \\
inv\,(S) \quad \wedge \\
L = ADList\,(T_1, S) \quad \wedge \\
alive\,(T_1, S)
\end{array}
\right\}
\quad \downarrow\text{-[var-rule]}
$$

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
this \neq null \quad \wedge \\
ADList\,(this\,, \$) = L
\end{array}
\right\}
\quad \downarrow\text{-[inv-rule]}
$$

```
boolean b; b = this.isempty();  // lemma 2
```

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
aB\,(b) = null?\,(L)
\end{array}
\right\}
\quad \uparrow\text{-[inv-rule]}
$$

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
S = update\,(S1\#\#NodeL\,, loc\,(L_1, Node?elem\,), I_1) \quad \wedge \\
L_1 = new\,(S1, NodeL\,) \quad \wedge \\
T = T_1 \quad \wedge \\
T_1 \neq null \quad \wedge \\
I = aI\,(I_1) \quad \wedge \\
inv\,(S) \quad \wedge \\
L = ADList\,(T_1, S) \quad \wedge \\
aB\,(b) = null?\,(L) \quad \wedge \\
alive\,(T_1, S)
\end{array}
\right\}
\quad \uparrow\text{-[var-rule]}
$$

$$
\left\{
\begin{array}{l}
\$ = S \quad \wedge \\
S = update\,(S1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\
l = new\,(S1, NodeL\,) \quad \wedge \\
T = this \quad \wedge \\
this \neq null \quad \wedge \\
I = aI\,(i) \quad \wedge \\
inv\,(S) \quad \wedge \\
L = ADList\,(this\,, S) \quad \wedge \\
aB\,(b) = null?\,(L) \quad \wedge \\
alive\,(this\,, S)
\end{array}
\right\}
$$

$\Longrightarrow$ (DList18_4)

$$\left\{\begin{array}{l} \exists X : X = S@@loc(this, DList?firstNode) \quad \wedge \\ \$ = S \quad \wedge \\ S = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ I = aI(i) \quad \wedge \\ inv(S) \quad \wedge \\ L = ADList(this, \$) \quad \wedge \\ aB(b) = null?(L) \quad \wedge \\ alive(this, \$) \end{array}\right\}$$

$\downarrow$–[ex-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ S = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ X = S@@loc(this, DList?firstNode) \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ I = aI(i) \quad \wedge \\ inv(S) \quad \wedge \\ L = ADList(this, \$) \quad \wedge \\ aB(b) = null?(L) \quad \wedge \\ alive(this, \$) \end{array}\right\}$$

$\downarrow$–[if-rule]

```
if(b) {  // begin of if-part
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ S = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ X = S@@loc(this, DList?firstNode) \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ I = aI(i) \quad \wedge \\ inv(S) \quad \wedge \\ L = ADList(this, \$) \quad \wedge \\ aB(b) = null?(L) \quad \wedge \\ aB(b) \quad \wedge \\ alive(this, \$) \end{array}\right\}$$

$\Longrightarrow$ (DList18_5)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList(T, update(update(\$, loc(this, DList?lastNode), l), loc(this, DList?firstNode), l)) = \\ \quad append(L, cons(I, null)) \end{array}\right\}$$

```
this.lastNode:=l;
```

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList(T, update(\$, loc(this, DList?firstNode), l)) = append(L, cons(I, null)) \end{array}\right\}$$

```
this.firstNode = l;
```

$$\{\ ADList(T, \$) = append(L, cons(I, null))\ \}$$

```
}  // end of if-aprt
```

```
else {  // begin of else-part
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ S = update\,(S1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S1, NodeL\,) \quad \wedge \\ X = S@@loc\,(this\,, DList?firstNode\,) \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ I = aI\,(i) \quad \wedge \\ inv\,(S) \quad \wedge \\ L = ADList\,(this\,, \$) \quad \wedge \\ aB\,(b) = null?\,(L) \quad \wedge \\ \neg\, aB\,(b) \quad \wedge \\ alive\,(this\,, \$) \end{array}\right\}$$

$\Longrightarrow$ (DList18_6)

$$\left\{\begin{array}{l} \$@@loc\,(this\,, DList?firstNode\,) \neq null \quad \wedge \\ \$ = S \quad \wedge \\ S = update\,(S1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S1, NodeL\,) \quad \wedge \\ \$@@loc\,(this\,, DList?firstNode\,) = S@@loc\,(this\,, DList?firstNode\,) \quad \wedge \\ X = \$@@loc\,(this\,, DList?firstNode\,) \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ ANodeL\,(X, S) = L \quad \wedge \\ I = aI\,(i) \quad \wedge \\ inv\,(S) \end{array}\right\}$$

```
NodeL f; f = this.firstNode;
```

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ \$ = S \quad \wedge \\ S = update\,(S1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S1, NodeL\,) \quad \wedge \\ f = S@@loc\,(this\,, DList?firstNode\,) \quad \wedge \\ X = f \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ ANodeL\,(X, S) = L \quad \wedge \\ I = aI\,(i) \quad \wedge \\ inv\,(S) \end{array}\right\}$$

$\downarrow$–[var-rule]

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T_1 = f \quad \wedge \\ \$ = S \quad \wedge \\ S = update\,(S1\#\#NodeL\,, loc\,(L_1, Node?elem\,), I_1) \quad \wedge \\ L_1 = new\,(S1, NodeL\,) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T_1 = S@@loc\,(T_2, DList?firstNode\,) \quad \wedge \\ X = T_1 \quad \wedge \\ T = T_2 \quad \wedge \\ T_2 \neq null \quad \wedge \\ ANodeL\,(X, S) = L \quad \wedge \\ I = aI\,(I_1) \quad \wedge \\ inv\,(S) \end{array}\right\}$$

$$\left\{\begin{array}{l} f \neq null \quad \wedge \\ T_1 = f \quad \wedge \\ \$ = S \end{array}\right\}$$

```
NodeL h; h = f.getLast();  // lemma 3
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : succn(T_1, S, N) = h \wedge succn(T_1, S, N+1) = null) \end{array}\right\}$$

——————————————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : succn(T_1, S, N) = h \wedge succn(T_1, S, N+1) = null) \quad \wedge \\ S = update(S1\#\#NodeL, loc(L_1, Node?elem), I_1) \quad \wedge \\ L_1 = new(S1, NodeL) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T_1 = S@@loc(T_2, DList?firstNode) \quad \wedge \\ X = T_1 \quad \wedge \\ T = T_2 \quad \wedge \\ T_2 \neq null \quad \wedge \\ ANodeL(X, S) = L \quad \wedge \\ I = aI(I_1) \quad \wedge \\ inv(S) \end{array}\right\}$$

——————————————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : succn(f, S, N) = h \wedge succn(f, S, N+1) = null) \quad \wedge \\ S = update(S1\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S1, NodeL) \quad \wedge \\ f \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \quad \wedge \\ X = f \quad \wedge \\ T = this \quad \wedge \\ this \neq null \quad \wedge \\ ANodeL(X, S) = L \quad \wedge \\ I = aI(i) \quad \wedge \\ inv(S) \end{array}\right\}$$

$\Longrightarrow$ (DList18_7)

$$\left\{\begin{array}{l} \exists k : l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l, \$) \quad \wedge \\ lastNode(l, \$) \quad \wedge \\ h \neq null \quad \wedge \\ lastNode(h, \$) \quad \wedge \\ succn(X, \$, k) = h \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ X = update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h)@@ \\ \qquad loc(T, DList?firstNode) \quad \wedge \\ ANodeL(X, S) = L \quad \wedge \\ ANodeL(l, S) = cons(I, null) \quad \wedge \\ this \neq null \end{array}\right\}$$

$$
\begin{array}{l}
\left\{\begin{array}{l}
l \neq null \quad \wedge \\
l \neq h \quad \wedge \\
fstNode?\,(l, \$) \quad \wedge \\
lastNode\,(l, \$) \quad \wedge \\
h \neq null \quad \wedge \\
lastNode\,(h, \$) \quad \wedge \\
succn\,(X, \$, k) = h \quad \wedge \\
\$ = S \quad \wedge \\
inv\,(\$) \quad \wedge \\
X = update\,(update\,(S, loc\,(h, Node?succ), l), loc\,(l, Node?pred), h)@@ \\
\qquad loc\,(T, DList?firstNode) \quad \wedge \\
ANodeL\,(X, S) = L \quad \wedge \\
ANodeL\,(l, S) = cons\,(I, null) \quad \wedge \\
this \neq null
\end{array}\right\} \\
\Longrightarrow \text{(DList18\_8)} \\
\left\{\begin{array}{l}
l \neq null \quad \wedge \\
l \neq h \quad \wedge \\
fstNode?\,(l, \$) \quad \wedge \\
lastNode\,(l, \$) \quad \wedge \\
h \neq null \quad \wedge \\
lastNode\,(h, \$) \quad \wedge \\
succn\,(X, \$, k) = h \quad \wedge \\
ANodeL\,(X, \$) = ANodeL\,(X, S) \quad \wedge \\
ANodeL\,(l, \$) = ANodeL\,(l, S) \quad \wedge \\
\$ = S \quad \wedge \\
inv\,(\$) \quad \wedge \\
X = update\,(update\,(S, loc\,(h, Node?succ), l), loc\,(l, Node?pred), h)@@ \\
\qquad loc\,(T, DList?firstNode) \quad \wedge \\
ANodeL\,(X, S) = L \quad \wedge \\
ANodeL\,(l, S) = cons\,(I, null) \quad \wedge \\
this \neq null
\end{array}\right\}
\end{array}
\quad \text{—↓—[ex-rule]}
$$

$$
\left\{\begin{array}{l}
l \neq null \quad \wedge \\
l \neq h \quad \wedge \\
fstNode?\,(l, \$) \quad \wedge \\
lastNode\,(l, \$) \quad \wedge \\
h \neq null \quad \wedge \\
lastNode\,(h, \$) \quad \wedge \\
succn\,(X, \$, k) = h \quad \wedge \\
ANodeL\,(X, \$) = L_1 \quad \wedge \\
ANodeL\,(l, \$) = M \quad \wedge \\
H = h \quad \wedge \\
L_2 = l \quad \wedge \\
\$ = S \quad \wedge \\
inv\,(\$) \quad \wedge \\
X = update\,(update\,(S, loc\,(H, Node?succ), L_2), loc\,(L_2, Node?pred), H)@@ \\
\qquad loc\,(T, DList?firstNode) \quad \wedge \\
L_1 = L \quad \wedge \\
M = cons\,(I, null) \quad \wedge \\
L_1 = ANodeL\,(X, S) \quad \wedge \\
T_1 \neq null
\end{array}\right\}
\quad \text{—↓—[var-rule]}
$$

$$\left\{\begin{array}{l} l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l,\$) \quad \wedge \\ lastNode(l,\$) \quad \wedge \\ h \neq null \quad \wedge \\ lastNode(h,\$) \quad \wedge \\ succn(X,\$,k) = h \quad \wedge \\ ANodeL(X,\$) = L_1 \quad \wedge \\ ANodeL(l,\$) = M \quad \wedge \\ H = h \quad \wedge \\ L_2 = l \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \end{array}\right\}$$

```
int k; k = h.appback(l);  // lemma 4
```

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(L_1, M) \quad \wedge \\ \$ = update(update(S, loc(H, Node?succ), L_2), loc(L_2, Node?pred), H) \quad \wedge \\ inv(\$) \end{array}\right\}$$

↑–[inv-rule]

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(L_1, M) \quad \wedge \\ \$ = update(update(S, loc(H, Node?succ), L_2), loc(L_2, Node?pred), H) \quad \wedge \\ inv(\$) \quad \wedge \\ X = update(update(S, loc(H, Node?succ), L_2), loc(L_2, Node?pred), H)@@ \\ \qquad loc(T, DList?firstNode) \quad \wedge \\ L_1 = L \quad \wedge \\ M = cons(I, null) \quad \wedge \\ L_1 = ANodeL(X,S) \quad \wedge \\ T_1 \neq null \end{array}\right\}$$

↑–[var-rule]

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(ANodeL(X,S), ANodeL(l,S)) \quad \wedge \\ \$ = update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h) \quad \wedge \\ inv(\$) \quad \wedge \\ X = update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h)@@ \\ \qquad loc(T, DList?firstNode) \quad \wedge \\ ANodeL(X,S) = L \quad \wedge \\ ANodeL(l,S) = cons(I, null) \quad \wedge \\ this \neq null \end{array}\right\}$$

↑–[ex-rule]

$$\left\{\begin{array}{l} ANodeL(X,\$) = append(ANodeL(X,S), ANodeL(l,S)) \quad \wedge \\ \$ = update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h) \quad \wedge \\ inv(\$) \quad \wedge \\ X = update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h)@@ \\ \qquad loc(T, DList?firstNode) \quad \wedge \\ ANodeL(X,S) = L \quad \wedge \\ ANodeL(l,S) = cons(I, null) \quad \wedge \\ this \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList18_9)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ ADList(T, update(\$, loc(this, DList?lastNode), l)) = append(L, cons(I, null)) \end{array}\right\}$$

```
this.lastNode = l;
```

$$\{\ ADList(T,\$) = append(L, cons(I, null))\ \}$$

```
}  // end of else-part
```

↑–[if-rule]

$$\{\ ADList(T,\$) = append(L, cons(I, null))\ \}$$

```
return 0;
```
$\{\ ADList(T,\$) = append(L, cons(I, null))\ \}$

$\{\ ADList(T,\$) = append(L, cons(I, null))\ \}$ ↑–[ex-rule]

8.19 Obligation 19

Lemmas:

1.

$\{\ \$ = S\ \}$

```
NodeL@initNodeL(int i)
```

$\left\{ \begin{array}{l} \$ = update(S\#\#NodeL, loc(result, Node?elem), i) \quad \wedge \\ result = new(S, NodeL) \end{array} \right\}$

$\{\ inv(\$)\ \}$

```
NodeL@initNodeL(int i)
```

$\{\ inv(\$)\ \}$

[conjunct-rule]

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \end{array} \right\}$

```
NodeL@initNodeL(int i)
```

$\left\{ \begin{array}{l} \$ = update(S\#\#NodeL, loc(result, Node?elem), i) \quad \wedge \\ result = new(S, NodeL) \quad \wedge \\ inv(\$) \end{array} \right\}$

[static-invocation-rule]

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \end{array} \right\}$

```
NodeL l; l = NodeL.initNodeL(i);
```

$\left\{ \begin{array}{l} \$ = update(S\#\#NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S, NodeL) \quad \wedge \\ inv(\$) \end{array} \right\}$

2.

$\{\ \$ = S\ \}$

```
DList@isempty()
```

$\{\ \$ = S\ \}$

$\{\ ADList(this, \$) = L\ \}$

```
DList@isempty()
```

$\{\ result = null?(L)\ \}$

[conjunct-rule]

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ ADList(this, \$) = L \end{array} \right\}$

```
DList@isempty()
```

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ result = null?(L) \end{array} \right\}$

[invocation-rule]

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \end{array} \right\}$

```
boolean b; b = this.isempty();
```

$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ b = null?(L) \end{array} \right\}$

3.

$$\left\{ \begin{array}{l} T = this \quad \wedge \\ \$ = S \end{array} \right\}$$

`NodeL@getLast()`

$$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ result \neq null \quad \wedge \\ (\exists N : \\ succn\,(T, S, N) = result \quad \wedge \\ succn\,(T, S, N+1) = null\,) \end{array} \right\}$$

$\{\ inv\,(\$)\ \}$

`NodeL@getLast()`

$\{\ inv\,(\$)\ \}$

———————————————————————————————————[conjunct-rule]

$$\left\{ \begin{array}{l} T = this \quad \wedge \\ \$ = S \quad \wedge \\ inv\,(\$) \end{array} \right\}$$

`NodeL@getLast()`

$$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ result \neq null \quad \wedge \\ (\exists N : \\ succn\,(T, S, N) = result \quad \wedge \\ succn\,(T, S, N+1) = null\,) \quad \wedge \\ inv\,(\$) \end{array} \right\}$$

———————————————————————————————————[invocation-rule]

$$\left\{ \begin{array}{l} T = f \quad \wedge \\ \$ = S \quad \wedge \\ inv\,(\$) \quad \wedge \\ f \neq null \end{array} \right\}$$

`NodeL h; h = f.getLast();`

$$\left\{ \begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : \\ succn\,(T, S, N) = h \quad \wedge \\ succn\,(T, S, N+1) = null\,) \quad \wedge \\ inv\,(\$) \end{array} \right\}$$

4.

$$\left\{ \begin{array}{l} n \neq null \quad \wedge \\ n \neq this \quad \wedge \\ fstNode?\,(n, \$) \quad \wedge \\ lstNode?\,(n, \$) \quad \wedge \\ lstNode?\,(this\,, \$) \quad \wedge \\ inv\,(\$) \end{array} \right\}$$

`NodeL@appback(NodeL n)`

$\{\ inv\,(\$)\ \}$

$$\left\{ \begin{array}{l} n \neq null \quad \wedge \\ n \neq this \quad \wedge \\ fstNode?\,(n, \$) \quad \wedge \\ lstNode?\,(n, \$) \quad \wedge \\ lstNode?\,(this\,, \$) \quad \wedge \\ T = this \quad \wedge \\ N = n \quad \wedge \\ \$ = S \end{array} \right\}$$

`NodeL@appback(NodeL n)`

$\{\ \$ = update(update(S, loc(T, Node?succ), N), loc(T, Node?pred), T)\ \}$

---[conjunct-rule]

$$\left\{\begin{array}{l} n \neq null \quad \wedge \\ n \neq this \quad \wedge \\ fstNode?(n, \$) \quad \wedge \\ lstNode?(n, \$) \quad \wedge \\ lstNode?(this, \$) \quad \wedge \\ inv(\$) \quad \wedge \\ T = this \quad \wedge \\ N = n \quad \wedge \\ \$ = S \end{array}\right\}$$

`NodeL@appback(NodeL n)`

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ \$ = \$ = update(update(S, loc(T, Node?succ), N), loc(T, Node?pred), T) \end{array}\right\}$$

---[invocation-rule]

$$\left\{\begin{array}{l} h \neq null \quad \wedge \\ l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l, \$) \quad \wedge \\ lstNode?(l, \$) \quad \wedge \\ lstNode?(h, \$) \quad \wedge \\ inv(\$) \quad \wedge \\ T = h \quad \wedge \\ N = l \quad \wedge \\ \$ = S \end{array}\right\}$$

`int k; k = h.appback(l);`

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ \$ = \$ = update(update(S, loc(T, Node?succ), N), loc(T, Node?pred), T) \end{array}\right\}$$

Proof:

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ alive(this, \$) \end{array}\right\}$$

$\Longrightarrow$ (DList19_2)

$$\left\{\begin{array}{l} \exists S_1 : \\ (\exists L : \\ \$ = S_1 \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, S_1) = L \quad \wedge \\ alive(this, S_1)) \end{array}\right\}$$

---$\downarrow$–[ex-rule]

$$\left\{\begin{array}{l} \exists L : \\ \$ = S_1 \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, S_1) = L \quad \wedge \\ alive(this, S_1) \end{array}\right\}$$

$$\left\{\begin{array}{l} \$ = S_1 \quad \wedge \\ inv\,(\$) \quad \wedge \\ this \neq null \quad \wedge \\ ADList\,(this\,, S_1) = L \quad \wedge \\ alive\,(this\,, S_1) \end{array}\right\}$$

——————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} \$ = S_1 \quad \wedge \\ inv\,(\$) \quad \wedge \\ T \neq null \quad \wedge \\ ADList\,(T, S_1) = L \quad \wedge \\ alive\,(T, S_1) \end{array}\right\}$$

——————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} \$ = S_1 \quad \wedge \\ inv\,(\$) \end{array}\right\}$$

——————————————————————————————↓–[subst-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv\,(\$) \end{array}\right\}$$

```
NodeL l; l = NodeL.initNodeL(i);  // lemma 1
```

$$\left\{\begin{array}{l} \$ = update\,(S\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S, NodeL\,) \quad \wedge \\ inv\,(\$) \end{array}\right\}$$

——————————————————————————————↑–[subst-rule]

$$\left\{\begin{array}{l} \$ = update\,(S_1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S_1, NodeL\,) \quad \wedge \\ inv\,(\$) \end{array}\right\}$$

——————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = update\,(S_1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S_1, NodeL\,) \quad \wedge \\ inv\,(\$) \quad \wedge \\ T \neq null \quad \wedge \\ ADList\,(T, S_1) = L \quad \wedge \\ alive\,(T, S_1) \end{array}\right\}$$

——————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = update\,(S_1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \quad \wedge \\ l = new\,(S_1, NodeL\,) \quad \wedge \\ inv\,(\$) \quad \wedge \\ this \neq null \quad \wedge \\ ADList\,(this\,, S_1) = L \quad \wedge \\ alive\,(this\,, S_1) \quad \wedge \\ inv\,(S_1) \end{array}\right\}$$

$\Longrightarrow$ (DList19_3)

$$\left\{\begin{array}{l} \exists S: \\ \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList\,(this\,, \$) = L \quad \wedge \\ inv\,(\$) \quad \wedge \\ alive\,(this\,, \$) \quad \wedge \\ l = new\,(S_1, NodeL\,) \quad \wedge \\ S = update\,(S_1\#\#NodeL\,, loc\,(l, Node?elem\,), i) \end{array}\right\}$$

——————————————————————————————↓–[ex-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ inv(\$) \quad \wedge \\ alive(this, \$) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \end{array}\right\}$$

$\Longrightarrow$ (DList19_4)

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ inv(S) \quad \wedge \\ alive(this, S) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ ADList(this, S) = L \end{array}\right\}$$

——————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ inv(S) \quad \wedge \\ T \neq null \quad \wedge \\ alive(T, S) \quad \wedge \\ L_1 = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(L_1, Node?elem), I) \quad \wedge \\ ADList(T, S) = L \end{array}\right\}$$

——————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ ADList(this, \$) = L \end{array}\right\}$$

```
boolean b; b = this.isempty();  // lemma 2
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ b = null?(L) \end{array}\right\}$$

——————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ b = null?(L) \quad \wedge \\ inv(S) \quad \wedge \\ T \neq null \quad \wedge \\ alive(T, S) \quad \wedge \\ L_1 = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(L_1, Node?elem), I) \quad \wedge \\ ADList(T, S) = L \end{array}\right\}$$

——————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ b = null?(L) \quad \wedge \\ inv(S) \quad \wedge \\ this \neq null \quad \wedge \\ alive(this, S) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ ADList(this, S) = L \end{array}\right\}$$

$\Longrightarrow$ (DList19_5)

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ alive(this, \$) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ b = null?(L) \end{array}\right\}$$

$\downarrow$–[if-rule]

```
if(b) {  // begin of if-part
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ alive(this, \$) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ b = null?(L) \quad \wedge \\ b \end{array}\right\}$$

$\Longrightarrow$ (DList19_6)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ inv(update(update(\$, loc(this, DList?lastNode), l), loc(this, DList?firstNode), l)) \end{array}\right\}$$

```
this.lastNode:=l;
```

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ inv(update(\$, loc(this, DList?firstNode), l)) \end{array}\right\}$$

```
this.firstNode = l;
```

$\{\ inv(\$)\ \}$

```
}  // end of if-aprt
else {  // begin of else-part
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ alive(this, \$) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ ADList(this, \$) = L \quad \wedge \\ b = null?(L) \quad \wedge \\ NOTb \end{array}\right\}$$

$\Longrightarrow$ (DList19_7)

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ this \neq null \quad \wedge \\ \$@@loc(this, DList?firstNode) \neq null \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ \$@@loc(this, DList?firstNode) = S@@loc(this, DList?firstNode) \end{array}\right\}$$

```
NodeL f; f = this.firstNode;
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ f \neq null \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ this \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \end{array}\right\}$$

$$\left\{\begin{array}{l} T = f \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ f \neq null \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(L, Node?elem), I_1) \quad \wedge \\ L = new(S_1, NodeL) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T = S@@(T_1, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

———————————————————————————————————————↓–[var-rule]

$$\left\{\begin{array}{l} T = f \quad \wedge \\ \$ = S \quad \wedge \\ inv(\$) \quad \wedge \\ f \neq null \end{array}\right\}$$

```
NodeL h; h = f.getLast();  lemma 3
```

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : \\ succn(T, S, N) = h \quad \wedge \\ succn(T, S, N+1) = null) \quad \wedge \\ inv(\$) \end{array}\right\}$$

———————————————————————————————————————↓–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : \\ succn(T, S, N) = h \quad \wedge \\ succn(T, S, N+1) = null) \quad \wedge \\ inv(\$) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(L, Node?elem), I_1) \quad \wedge \\ L = new(S_1, NodeL) \quad \wedge \\ T_1 \neq null \quad \wedge \\ T = S@@loc(T_1, DList?firstNode) \quad \wedge \\ T \neq null \end{array}\right\}$$

———————————————————————————————————————↑–[inv-rule]

$$\left\{\begin{array}{l} \$ = S \quad \wedge \\ h \neq null \quad \wedge \\ (\exists N : \\ succn(f, S, N) = h \quad \wedge \\ succn(f, S, N+1) = null) \quad \wedge \\ inv(\$) \quad \wedge \\ S = update(S_1 \#\# NodeL, loc(l, Node?elem), i) \quad \wedge \\ l = new(S_1, NodeL) \quad \wedge \\ this \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \quad \wedge \\ f \neq null \end{array}\right\}$$

$\Longrightarrow$ (DList19_8)

———————————————————————————————————————↑–[var-rule]

$$\left\{\begin{array}{l} \exists k : \\ h \neq null \quad \wedge \\ l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l, \$) \quad \wedge \\ lstNode?(l, \$) \quad \wedge \\ lstNode?(h, \$) \quad \wedge \\ inv(\$) \quad \wedge \\ \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ f \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \quad \wedge \\ succn(f, S, k) = h \quad \wedge \\ lstNode?(h, S) \quad \wedge \\ alive(l, S) \quad \wedge \\ alive(h, S) \end{array}\right\}$$

$\downarrow$–[ex-rule]

$$\left\{\begin{array}{l} h \neq null \quad \wedge \\ l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l, \$) \quad \wedge \\ lstNode?(l, \$) \quad \wedge \\ lstNode?(h, \$) \quad \wedge \\ inv(\$) \quad \wedge \\ \$ = S \quad \wedge \\ this \neq null \quad \wedge \\ f \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \quad \wedge \\ succn(f, S, k) = h \quad \wedge \\ lstNode?(h, S) \quad \wedge \\ alive(l, S) \quad \wedge \\ alive(h, S) \end{array}\right\}$$

$\downarrow$–[var-rule]

$$\left\{\begin{array}{l} h \neq null \quad \wedge \\ l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l, \$) \quad \wedge \\ lstNode?(l, \$) \quad \wedge \\ lstNode?(h, \$) \quad \wedge \\ inv(\$) \quad \wedge \\ T = h \quad \wedge \\ N = l \quad \wedge \\ \$ = S \quad \wedge \\ A \neq null \quad \wedge \\ T \neq null \quad \wedge \\ F \neq null \quad \wedge \\ N \neq null \quad \wedge \\ F = S@@loc(A, DList?firstNode) \quad \wedge \\ succn(F, S, k) = T \quad \wedge \\ lstNode?(T, S) \quad \wedge \\ alive(N, S) \quad \wedge \\ alive(T, S) \end{array}\right\}$$

$$\left\{\begin{array}{l} h \neq null \quad \wedge \\ l \neq null \quad \wedge \\ l \neq h \quad \wedge \\ fstNode?(l,\$) \quad \wedge \\ lstNode?(l,\$) \quad \wedge \\ lstNode?(h,\$) \quad \wedge \\ inv(\$) \quad \wedge \\ T = h \quad \wedge \\ N = l \quad \wedge \\ \$ = S \end{array}\right\}$$

```
int k; k = h.appback(l);  //lemma 4
```

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ \$ = update(update(S, loc(T, Node?succ), N)loc(N, Node?pred), T) \end{array}\right\}$$

---------- ↑–[inv-rule]

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ \$ = update(update(S, loc(T, Node?succ), N)loc(N, Node?pred), T) \quad \wedge \\ A \neq null \quad \wedge \\ T \neq null \quad \wedge \\ F \neq null \quad \wedge \\ N \neq null \quad \wedge \\ F = S@@loc(A, DList?firstNode) \quad \wedge \\ succn(F, S, k) = T \quad \wedge \\ lstNode?(T, S) \quad \wedge \\ alive(N, S) \quad \wedge \\ alive(T, S) \end{array}\right\}$$

---------- ↑–[var-rule]

$$\left\{\begin{array}{l} inv(\$) \quad \wedge \\ \$ = update(update(S, loc(h, Node?succ), l)loc(l, Node?pred), h) \quad \wedge \\ this \neq null \quad \wedge \\ h \neq null \quad \wedge \\ f \neq null \quad \wedge \\ l \neq null \quad \wedge \\ f = S@@loc(this, DList?firstNode) \quad \wedge \\ succn(f, S, k) = h \quad \wedge \\ lstNode?(h, S) \quad \wedge \\ alive(l, S) \quad \wedge \\ alive(h, S) \end{array}\right\}$$

$\Longrightarrow$ (DList19_9)

$$\left\{\begin{array}{l} this \neq null \quad \wedge \\ inv(update(\$, loc(this, DList?lastNode), l)) \end{array}\right\}$$

```
this.lastNode = l;
```

$\{\ inv(\$)\ \}$

---------- ↑–[ex-rule]

$\{\ inv(\$)\ \}$

```
}  // end of else-part
```

---------- ↑–[if-rule]

$\{\ inv(\$)\ \}$

```
return 0;
```

$\{\ inv(\$)\ \}$

---------- ↑–[ex-rule]

$\{\ inv(\$)\ \}$

8.20 Obligation 20

Let P resp. Q be the pre- resp. postcondition of DList's obligation 1.

Proof :

$\{typeof(this) = ct(DList) \wedge P\}$ DList@init() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:init() $\{Q\}$ (Lemma 2)
———————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:init() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@init() $\{Q\}$
———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@init() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:init() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:init() $\{Q\}$

8.21 Obligation 21

Let P resp. Q be the pre- resp. postcondition of DList's obligation 6.

Proof :

$\{typeof(this) = ct(DList) \wedge P\}$ DList@first() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:first() $\{Q\}$ (Lemma 2)
———————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:first() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@first() $\{Q\}$
———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@first() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:first() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:first() $\{Q\}$

8.22 Obligation 22

Let P resp. Q be the pre- resp. postcondition of DList's obligation 7.

Proof :

$\{typeof(this) = ct(DList) \wedge P\}$ DList@first() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:first() $\{Q\}$ (Lemma 2)
———————————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:first() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@first() $\{Q\}$

———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@first() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:first() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:first() $\{Q\}$

8.23 Obligation 23

Let P resp. Q be the pre- resp. postcondition of DList's obligation 12.

Proof :

$\{typeof(this) = ct(DList) \wedge P\}$	DList@rest()	$\{Q\}$	(Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$	DList:rest()	$\{Q\}$	(Lemma 2)
——————————————————————————————————[class-rule]			
$\{typeof(this) \leq ct(DList) \wedge P\}$	DList:rest()	$\{Q\}$	

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@rest() $\{Q\}$
———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@rest() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:rest() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:rest() $\{Q\}$

8.24 Obligation 24

Let P resp. Q be the pre- resp. postcondition of DList's obligation 13.

Proof :

$\{typeof(this) = ct(DList) \wedge P\}$	DList@rest()	$\{Q\}$	(Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$	DList:rest()	$\{Q\}$	(Lemma 2)
——————————————————————————————————[class-rule]			
$\{typeof(this) \leq ct(DList) \wedge P\}$	DList:rest()	$\{Q\}$	

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@rest() $\{Q\}$
———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@rest() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:rest() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:rest() $\{Q\}$

8.25 Obligation 25

Let P resp. Q be the pre- resp. postcondition of DList's obligation 15.

Proof :
$\{typeof(this) = ct(DList) \wedge P\}$ DList@isempty() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:isempty() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:isempty() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@isempty() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@isempty() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:isempty() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:isempty() $\{Q\}$

8.26 Obligation 26

Let P resp. Q be the pre- resp. postcondition of DList's obligation 16.

Proof :
$\{typeof(this) = ct(DList) \wedge P\}$ DList@isempty() $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:isempty() $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:isempty() $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@isempty() $\{Q\}$
———————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@isempty() $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:isempty() { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:isempty() $\{Q\}$

8.27 Obligation 27

Let P resp. Q be the pre- resp. postcondition of DList's obligation 18.

Proof :
$\{typeof(this) = ct(DList) \wedge P\}$ DList@append(int i) $\{Q\}$ (Lemma 1)
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:append(int i) $\{Q\}$ (Lemma 2)
———————————————————————————————[class-rule]
$\{typeof(this) \leq ct(DList) \wedge P\}$ DList:append(int i) $\{Q\}$

Lemma 1:
$(typeof(this) = ct(DList) \wedge P) \Rightarrow P$
$\{P\}$ DList@append(int i) $\{Q\}$

———————————————————————————————————[strength-rule]
$\{typeof(this) = ct(DList) \wedge P\}$ DList@append(int i) $\{Q\}$

Lemma 2:
$(typeof(this) < ct(DList) \wedge this \neq null \wedge P) \Rightarrow$ (DListVirt1) FALSE
{ FALSE } DList:append(int i) { FALSE } (false-axiom)
FALSE $\Rightarrow \{Q\}$
———————————————————————————————————[strength-, weak-rule]
$\{typeof(this) < ct(DList) \wedge this \neq null \wedge P\}$ DList:append(int i) $\{Q\}$

9 Experiences and Further Work

The case study reported here revealed several interesting aspects of software verification in general, and about our specification and verification technique in particular.

The doubly linked list example consists of about 250 lines of annotated ANJA code and about 180 lines universal specification. This results in 63 proof obligations. For the proofs of these obligations, we used 98 program-independent lemmas. These numbers illustrate the complexity of verification in object-oriented programs, and make clear that tool support is indispensable for the verification of nontrivial programs (1) to check proof steps and avoid errors and sloppiness; (2) to provide user guidance and clear presentation of complex proofs and formulas; (3) to automate recurring proof steps as tactics.

The correctness of proofs relies on subtle points, in particular when assumptions are used to handle recursion. Thus, mechanical proof checking is necessary to avoid errors.

The case study illustrates the complexity of specifications and proofs. Therefore, sophisticated user interfaces are necessary to display proofs in a clear way. So far, the JIVE system provides a tree view that visualizes the proof data structures, and a text view that displays proofs embedded into program code. Both views allow one to selectively display proof information, and thus to handle complex formulas and proofs.

The proofs also reveal that much of the effort is used for almost trivial, recurring proof steps, in particular for the verification of method invocations (see e.g. the verification of `DList`'s method `append`, obligations 18 and 19). This effort can be drastically reduced by elaborate proof strategies that develop large parts of the proofs automatically. So far, we implemented a weakest-precondition strategy, and strategies to verify virtual methods and to handle/eliminate assumptions [MPH00b]. The development of other strategies, for instance for the verification of method invocations is considered further work.

In the current implementation of JIVE, we exploit subtyping in PVS specifications. Since the PVS type system is undecidable, that leads to large numbers of type check conditions that cause much of the proof effort for the program-independent lemmas. Thus, it seems reasonable to replace the current formalization of the data and state model (see App. A) by a version without subtyping. We will do that in future versions of JIVE.

In a nutshell, formal verification of nontrivial object-oriented programs is made possible by the JIVE system. However, to make verification practical, we have to improve our specification and verification technique to reduce the number and complexity of proof obligations (in particular, for type invariants), and we must further increase the automation of program proofs.

A The Predefined PVS-Theories

The section presents the formal data and state model of the ANJA programming language. This formalization is program-independent. It is part of all universal specifications and can for instance be used to formally define abstraction functions. A detailed discussion of the formal data and state models of object-oriented languages can be found in [PH97, PHM98].

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Java and PVS integers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

JavaIntegers: THEORY
BEGIN

MinInt: int = -2147483648
MaxInt: int =  2147483647

% JavaInt is a subsort of PVS int. Therefore it is not necessary to
% introduce mapping. Whenever an int is used as JavaInt, a TCC will
% guarantee that the value lies within the boundaries.

JavaInt: TYPE = { i: int | MinInt <= i AND i <= MaxInt }

END JavaIntegers

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Java types
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

JavaTypes[CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE]: THEORY
BEGIN

JavaType: DATATYPE
BEGIN
  booleanType           : booleanType?
  intType               : intType?
  nullType              : nullType?
  ct( cid: CTypeId )    : ct?
  at( aid: ATypeId )    : at?
  it( iid: ITypeId )    : it?
END JavaType

isprimtype(t: JavaType): bool =
  booleanType?(t) OR intType?(t) OR nullType?(t)

END JavaTypes

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The subtype relation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Subtype[CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE]: THEORY
BEGIN
IMPORTING JavaTypes[CTypeId, ATypeId, ITypeId]

% The subtype relation could be axiomized by enumerating the direct
% subtype relation. However, showing that a type is a subtype of another
% type would require several steps and is therefore quite cumbersome.
% Thus, we directly axiomize the transitive closure of the direct subtype
```

```
% relation.

s, t: VAR JavaType

< (s, t): bool
<=(s, t): bool


sub2: AXIOM
  relations[JavaType].transitive?(<)

sub3: AXIOM
  relations[JavaType].irreflexive?(<)

sub4: AXIOM
  NOT isprimtype(t)  <=>  nullType < t

sub5: AXIOM
  NOT booleanType < t

sub6: AXIOM
  NOT intType < t

sub7: AXIOM
  isprimtype(t) => NOT s < t

sub8: AXIOM
  s <= t  <=>  s = t OR s < t

sub9: LEMMA
  orders[JavaType].partial_order?(<=)

sub10: LEMMA
  t <= t

END Subtype

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Values[CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE]: THEORY
BEGIN

IMPORTING Subtype[CTypeId, ATypeId, ITypeId], JavaIntegers

ObjectId: TYPE+

% Note that this data type implicitly declares the standard abstraction
% functions for bool and int by naming the seletors.

Value: DATATYPE
BEGIN
  bool(aB: bool)   : bool?
  int (aI: JavaInt): int?
  null             : null?
  ref (tid: CTypeId, oid: ObjectId): ref?
END Value
```

```
% Just for notational convenience:
JavaObject: TYPE = (ref?)

typeof(v: Value): JavaType =
  CASES v OF
    bool(b)       : booleanType,
    int(i)        : intType,
    null          : nullType,
    ref(tid, oid) : ct(tid)
  ENDCASES

typeof1: LEMMA
  FORALL (X: Value, tid: CTypeId):
  typeof(X) <= ct(tid) => null?(X) OR ref?(X)

typeof2: LEMMA
  FORALL (X: Value, tid: CTypeId):
  typeof(X) = ct(tid) => ref?(X) AND tid(X) = tid

% Note that we define null as initial value of the null type to be
% able to use init without producing TCCs

init(t: JavaType): Value =
  CASES t OF
    booleanType: bool(false),
    intType    : int(0)
    ELSE null
  ENDCASES

init1: LEMMA
   FORALL (t: JavaType): isprimtype(typeof(init(t)))

END Values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Locations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Locations
  [CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE,
  (IMPORTING Values[CTypeId, ATypeId, ITypeId])
  SimpleAttId: TYPE, AttId: TYPE, CAttId: TYPE FROM AttId,
  rtype:  [AttId -> JavaType],
  dtype:  [SimpleAttId -> JavaType],
  mkatt:  [sai: SimpleAttId, cid: {X: CTypeId| ct(X) <= dtype(sai)} -> CAttId],
  otypec: [CAttId -> (ct?)]                ]: THEORY

BEGIN

ASSUMING
mkatt_equality: ASSUMPTION
  FORALL (s1,s2: SimpleAttId,
          c1   : {c: CTypeId | ct(c) <= dtype(s1)},
          c2   : {c: CTypeId | ct(c) <= dtype(s2)}):
  mkatt(s1, c1) = mkatt(s2, c2) <=> s1 = s2 AND c1 = c2
```

```
obj_loc: ASSUMPTION
  FORALL (s: SimpleAttId, cid: {X: CTypeId | ct(X) <= dtype(s)}):
  cid(otypec(mkatt(s, cid))) = cid
ENDASSUMING

Location: DATATYPE
BEGIN
  mkloc(aid: CAttId, oid: ObjectId): mkloc?
END Location

ltype(l: Location): JavaType =
  CASES l OF
    mkloc(a, id): rtype(a)
  ENDCASES

obj(l: Location): Value =
  CASES l OF
    mkloc(a, id): ref(cid(otypec(a)), id)
  ENDCASES

loc(o: JavaObject, s: {X: SimpleAttId | ct(tid(o)) <= dtype(X)}): Location =
  CASES o OF
    ref(tid, oid): mkloc(mkatt(s, tid), oid)
  ENDCASES

loc_equality: LEMMA
  FORALL (o1: JavaObject, s1: {s: SimpleAttId | ct(tid(o1)) <= dtype(s)},
          o2: JavaObject, s2: {s: SimpleAttId | ct(tid(o2)) <= dtype(s)}):
  loc(o1, s1) = loc(o2, s2) <=> o1 = o2 AND s1 = s2

objloc: LEMMA
  FORALL (o: JavaObject, sai: {s: SimpleAttId | ct(tid(o)) <= dtype(s)}):
  obj(loc(o, sai)) = o

END Locations

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Stores
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Stores
  [CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE,
  (IMPORTING Subtype[CTypeId, ATypeId, ITypeId])
  SimpleAttId: TYPE, AttId: TYPE, CAttId: TYPE FROM AttId,
  rtype:  [AttId -> JavaType],
  dtype:  [SimpleAttId -> JavaType],
  mkatt:  [sai: SimpleAttId, cid: {X: CTypeId| ct(X) <= dtype(sai)} -> CAttId],
  otypec: [CAttId -> (ct?)]                ]: THEORY

BEGIN

ASSUMING
mkatt_equality: ASSUMPTION
  FORALL (s1,s2: SimpleAttId,
          c1   : {c: CTypeId | ct(c) <= dtype(s1)},
          c2   : {c: CTypeId | ct(c) <= dtype(s2)}):
  mkatt(s1, c1) = mkatt(s2, c2) <=> s1 = s2 AND c1 = c2
```

```
obj_loc: ASSUMPTION
  FORALL (s: SimpleAttId, cid: {X: CTypeId | ct(X) <= dtype(s)}):
  cid(otypec(mkatt(s, cid))) = cid
ENDASSUMING

IMPORTING Locations[CTypeId, ATypeId, ITypeId,
          SimpleAttId, AttId, CAttId, rtype, dtype, mkatt, otypec]



Store: TYPE+

S, S1, S2: VAR Store
L, L1, L2: VAR Location
T: VAR CTypeId
X, Y: VAR Value

update: [ Store, Location, Value -> Store ]
##(S, T): Store
@@(S, L): {X: Value | typeof(X) <= ltype(L)}
alive: [ Value, Store -> bool ]
new: [ Store, CTypeId -> Value ]

store1: AXIOM
  L1 /= L2 => update(S, L1, X)@@L2 = S@@L2

store2: AXIOM
  alive(obj(L), S) AND alive(X, S) => update(S, L, X)@@L = X

store3: AXIOM
  NOT alive(obj(L), S) => S@@L = init(ltype(L))

store4: AXIOM
  NOT alive(X, S) => update(S, L, X) = S

store5: AXIOM
  S##T@@L = S@@L

store6: AXIOM
  alive(X, update(S, L, Y)) = alive(X, S)

store7: AXIOM
  alive(X, S##T) = (alive(X, S) OR X = new(S, T))

store8: AXIOM
  alive(S@@L, S)

store9: AXIOM
  isprimtype(typeof(X)) => alive(X, S)

store10: AXIOM
  NOT alive(new(S, T), S)

store11: AXIOM
  typeof(new(S, T)) = ct(T)
```

```
store12: AXIOM
  new(S1, T) = new(S2, T)  <=>
  FORALL (X: Value): typeof(X) = ct(T) => (alive(X, S1) <=> alive(X, S2))

store13: AXIOM
  (FORALL (X: Value): alive(X, S1) <=> alive(X, S2)) AND
  (FORALL (L: Location): S1@@L = S2@@L) => S1 = S2

new1: LEMMA
  FORALL (S: Store, T: CTypeId): ref?(new(S, T))

new2: LEMMA
  FORALL (S: Store, T: CTypeId): new(S, T) /= null

END Stores

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% StoreProperties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

StoreProperties
  [CTypeId: TYPE, ATypeId: TYPE, ITypeId: TYPE,
  (IMPORTING Subtype[CTypeId, ATypeId, ITypeId])
  SimpleAttId: TYPE, AttId: TYPE,
  CAttId: TYPE FROM AttId, AAttId: TYPE FROM AttId,
  rtype:  [AttId -> JavaType],
  dtype:  [SimpleAttId -> JavaType],
  mkatt:  [sai: SimpleAttId, cid: {X: CTypeId| ct(X) <= dtype(sai)} -> CAttId],
  otypec: [CAttId -> (ct?)],
  otypea: [AAttId -> JavaType]                ]: THEORY

BEGIN

ASSUMING
dt: ASSUMPTION
    FORALL (aid: AttId): NOT CAttId_pred(aid) => AAttId_pred(aid)

mkatt_equality: ASSUMPTION
  FORALL (s1,s2: SimpleAttId,
          c1   : {c: CTypeId | ct(c) <= dtype(s1)},
          c2   : {c: CTypeId | ct(c) <= dtype(s2)}):
  mkatt(s1, c1) = mkatt(s2, c2) <=> s1 = s2 AND c1 = c2

obj_loc: ASSUMPTION
  FORALL (s: SimpleAttId, cid: {X: CTypeId | ct(X) <= dtype(s)}):
  cid(otypec(mkatt(s, cid))) = cid
ENDASSUMING

IMPORTING Stores[CTypeId, ATypeId, ITypeId,
                 SimpleAttId, AttId, CAttId, rtype, dtype, mkatt, otypec]


S, S1, S2: VAR Store
K, L, L1, L2: VAR Location
X, Y, Z: VAR Value
T: VAR CTypeId
A: VAR CAttId
OID: VAR ObjectId
```

```
T1, T2: VAR JavaType
SAI: VAR SimpleAttId


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lemmas to support proofs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Comm_h1: LEMMA
   alive(Y, update(S##T, L, X)) = alive(Y, update(S, L, X)##T)

Comm_h2: LEMMA
   K /= L => update(S##T, L, X)@@K = update(S, L, X)##T@@K

Comm_h4: LEMMA
   K = L AND obj(L) /= new(S, T) AND X /= new(S, T) =>
   update(S##T, L, X)@@K = update(S, L, X)##T@@K

%-----------------------------------------------------------------------

comm: LEMMA     % Lemma 3.1 in [PH97]
  obj(L) /= new(S, T) AND X /= new(S, T) =>
  update(S##T, L, X) = update(S, L, X)##T


reach(X, L, S): INDUCTIVE bool =
  X = obj(L) OR EXISTS (K: Location): obj(K) = X AND reach(S@@K, L, S)

reachR(X, L, S): INDUCTIVE bool =
  X = obj(L) OR EXISTS (K: Location): S@@K = obj(L) AND reachR(X, K, S)

reach_h1: LEMMA
  reach(X, L, S) AND obj(L) /= X  =>
  EXISTS K: reach(X, K, S) AND obj(L) = S@@K

reachR_h1: LEMMA
  reachR(X, L, S) AND obj(L) /= X  =>
  EXISTS K: reachR(S@@K, L, S) AND obj(K) = X

reach0: LEMMA
  reach(X, L, S) = reachR(X, L, S)

reach1: LEMMA   % Lemma 3.2 (i) in [PH97]
  NOT reach(X, L1, S) =>
  (reach(X, L2, S) <=> reach(X, L2, update(S, L1, Y)))

reach2: LEMMA   % Lemma 3.2 (ii)
  NOT reach(X, L, S) =>
  NOT reach(X, L, update(S, L, Y))

reach4: LEMMA   % Lemma 3.2 (iv)
  NOT reach(ref(cid(otypec(A)), OID), L, S) => mkloc(A, OID) /= L

reach5: LEMMA   % Lemma 3.2 (v)
  reach(X, L, S) <=> reach(X, L, S##T)

reach6: LEMMA   % Lemma 3.2 (vi)
  isprimtype(typeof(X)) => NOT reach(X, L, S)
```

```
reach3: LEMMA   % Lemma 3.2 (iii)
  NOT reach(Y, L1, S) AND NOT reach(X, L1, S) =>
  NOT reach(X, L1, update(S, L2, Y))

reach7: LEMMA   % Lemma 3.2 (vii)
  NOT alive(X, S) AND reach(X, L, S) => X = obj(L)

reach8: LEMMA   % Lemma 3.2 (viii)
  alive(X, S) AND reach(X, L, S) => alive(obj(L), S)

reach9: LEMMA   % Lemma 3.2 (ix)
  (FORALL (L: Location): reach(X, L, S1) => S1@@L = S2@@L) =>
  (FORALL (L: Location): reach(X, L, S1) <=> reach(X, L, S2))


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% oreach
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oreach(X: Value, Y: Value, S: Store): bool =
  EXISTS (L: Location): reach(X, L, S) AND Y = obj(L)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% treach
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

otype(aid: AttId): JavaType =
  IF CAttId_pred(aid) THEN otypec(aid)
  ELSE otypea(aid) ENDIF

treach_t1: AXIOM
  typeof(S@@K) <= rtype(aid(K))

treach_t2: AXIOM
  typeof(obj(K)) = otype(aid(K))

treach(T1: JavaType, T2: JavaType): INDUCTIVE bool =
  T2 <= T1 OR
  EXISTS (A: AttId): T2 <= rtype(A) AND treach(T1, otype(A))

treach1: LEMMA  % Lemma 4.6 in [PH97]
  typeof(X) <= T1 AND typeof(obj(L)) = T2 AND NOT treach(T1, T2) =>
  NOT reach(X, L, S)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% disj
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disj(X: Value, Y: Value, S: Store): bool =
  FORALL (L: Location): NOT reach(X, L, S) OR NOT reach(Y, L, S)

disj1: LEMMA    % Lemma 3.3 (i) in [PH97]
  disj(X, Y, S) AND NOT reach(X, L, S) AND NOT reach(Y, L, S) =>
  disj(X, Y, update(S, L, Z))
```

```
disj2: LEMMA    % Lemma 3.3 (ii)
  disj(X, Y, S) AND disj(X, Z, S) AND NOT reach(X, L, S) =>
  disj(X, Y, update(S, L, Z))

disj3: LEMMA    % Lemma 3.3 (iii)
   alive(X, S) => disj(X, new(S, T), S##T)

disj4: LEMMA    % Lemma 3.3 (iv)
  FORALL (X: JavaObject, SAI: {Y: SimpleAttId | ct(tid(X)) <= dtype(Y)}):
  typeof(X) = otype(mkatt(SAI, tid(X))) AND disj(X, Y, S) =>
  disj(S@@loc(X, SAI), Y, S)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% X-equivalence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

;
==(S1, S2): [ Value -> bool ] =
  LAMBDA (X: Value):
  (alive(X, S1) <=> alive(X, S2)) AND
  FORALL (L: Location): reach(X, L, S1) => S1@@L = S2@@L

Xequ1: LEMMA    % Lemma 3.4 (i) in [PH97]
  relations[Store].equivalence?(LAMBDA (S1: Store, S2: Store): (S1==S2)(X))

Xequ2: LEMMA    % Lemma 3.4 (ii)
  (FORALL (X: Value): (S1==S2)(X)) => S1 = S2

Xequ3: LEMMA    % Lemma 3.4 (iii)
  NOT reach(X, L, S) => (S==update(S, L, Y))(X)

Xequ4: LEMMA    % Lemma 3.4 (iv)
  X /= new(S, T) => (S==S##T)(X)

Xequ5: LEMMA    % Lemma 3.4 (v)
  (S1==S2)(X) => (reach(X, L, S1) <=> reach(X, L, S2))


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% T-equivalence
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tequ(T: JavaType, S1: Store, S2: Store): bool =
  FORALL (X: Value): typeof(X) <= T => (S1==S2)(X)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% less alive
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
<<(S1, S2): bool =
  FORALL (X: Value): alive(X, S1) => (S1==S2)(X)

lessalive1: LEMMA       % Lemma 3.5 (i) in [PH97]
  orders[Store].partial_order?(<<)

lessalive2: LEMMA       % Lemma 3.5 (ii)
  S1<<S2 AND alive(X, S1) => alive(X, S2)
```

```
lessalive3: LEMMA       % Lemma 3.5 (iii)
  S1<<S2 AND (alive(X, S1) OR NOT alive(X, S2)) =>
  (S1==S2)(X)

lessalive4: LEMMA       % Lemma 3.5 (iv)
  S1<<S2 AND NOT alive(obj(L), S2) => S1<<update(S2, L, X)

lessalive5: LEMMA       % Lemma 3.5 (v)
  S<<S##T

END StoreProperties
```

B The Program-Dependent PVS-Theories

To refer to program elements such as fields or types in specifications and proofs, we generate a number of PVS theories for every program. They are imported by the universal specification of the corresponding program. This section contains the theories for the list example.

```
TypeIds: THEORY
BEGIN
TypeId: DATATYPE WITH SUBTYPES CTypeId, ATypeId, ITypeId
BEGIN
  Node  : Node?  : CTypeId
  NodeL  : NodeL?  : CTypeId
  DList  : DList?  : CTypeId
  Interface  : Interface?  : ITypeId
  Object  : Object?  : CTypeId
  Operator  : Operator?  : ATypeId
END TypeId
END TypeIds


SubtypeEnum: THEORY
BEGIN
jive: LIBRARY = "/users/labeth/jive/TPC/theories"
IMPORTING TypeIds, jive@Subtype[CTypeId, ATypeId, ITypeId],
          jive@Values[CTypeId, ATypeId, ITypeId]

sub0: AXIOM
  ct(NodeL) < ct(Node) AND
  ct(Node) < it(Interface) AND
  ct(Node) < ct(Object) AND
  ct(NodeL) < it(Interface) AND
  ct(NodeL) < ct(Object) AND
  ct(DList) < it(Interface) AND
  ct(DList) < ct(Object) AND
  it(Interface) < ct(Object) AND
  at(Operator) < it(Interface) AND
  at(Operator) < ct(Object)

sub1: AXIOM
  NOT ct(Node) <= ct(NodeL) AND
  NOT ct(Node) <= ct(DList) AND
  NOT ct(DList) <= ct(Node) AND
  NOT it(Interface) <= ct(Node) AND
  NOT ct(Object) <= ct(Node) AND
  NOT ct(Node) <= at(Operator) AND
  NOT at(Operator) <= ct(Node) AND
  NOT ct(NodeL) <= ct(DList) AND
  NOT ct(DList) <= ct(NodeL) AND
  NOT it(Interface) <= ct(NodeL) AND
  NOT ct(Object) <= ct(NodeL) AND
  NOT ct(NodeL) <= at(Operator) AND
  NOT at(Operator) <= ct(NodeL) AND
  NOT it(Interface) <= ct(DList) AND
  NOT ct(Object) <= ct(DList) AND
  NOT ct(DList) <= at(Operator) AND
  NOT at(Operator) <= ct(DList) AND
  NOT ct(Object) <= it(Interface) AND
  NOT it(Interface) <= at(Operator) AND
  NOT ct(Object) <= at(Operator)
```

```
sub11: LEMMA
  ct(NodeL) <= ct(NodeL) AND
  ct(Node) <= ct(Node) AND
  ct(DList) <= ct(DList) AND
  ct(Object) <= ct(Object) AND
  it(Interface) <= it(Interface) AND
  at(Operator) <= at(Operator) AND
  ct(NodeL) <= ct(Node) AND
  ct(Node) <= it(Interface) AND
  ct(Node) <= ct(Object) AND
  ct(NodeL) <= it(Interface) AND
  ct(NodeL) <= ct(Object) AND
  ct(DList) <= it(Interface) AND
  ct(DList) <= ct(Object) AND
  it(Interface) <= ct(Object) AND
  at(Operator) <= it(Interface) AND
  at(Operator) <= ct(Object)

Node_closed: LEMMA
  FORALL (X: Value): typeof(X) <= ct(Node) =>
  typeof(X)=nullType OR typeof(X)=ct(NodeL) OR typeof(X)=ct(Node)

NodeL_closed: LEMMA
  FORALL (X: Value): typeof(X) <= ct(NodeL) =>
  typeof(X)=nullType OR typeof(X)=ct(NodeL)

DList_closed: LEMMA
  FORALL (X: Value): typeof(X) <= ct(DList) =>
  typeof(X)=nullType OR typeof(X)=ct(DList)

Interface_closed: LEMMA
  FORALL (X: Value): typeof(X) <= it(Interface) =>
  typeof(X)=nullType OR typeof(X)=it(Interface) OR typeof(X)=ct(Node) OR
  typeof(X)=ct(NodeL) OR typeof(X)=ct(DList) OR typeof(X)=at(Operator)

Object_closed: LEMMA
  FORALL (X: Value): typeof(X) <= ct(Object) =>
  typeof(X)=nullType OR typeof(X)=ct(Object) OR typeof(X)=ct(Node) OR
  typeof(X)=ct(NodeL) OR typeof(X)=ct(DList) OR typeof(X)=it(Interface) OR
  typeof(X)=at(Operator)

Operator_closed: LEMMA
  FORALL (X: Value): typeof(X) <= at(Operator) =>
  typeof(X)=nullType OR typeof(X)=at(Operator)

END SubtypeEnum


Attributes: THEORY
BEGIN
IMPORTING SubtypeEnum

SimpleAttId: DATATYPE
BEGIN
  Node?elem : Node?elem?
  Node?pred : Node?pred?
  Node?succ : Node?succ?
```

```
  DList?firstNode : DList?firstNode?
  DList?lastNode : DList?lastNode?

END SimpleAttId

dtype(aid: SimpleAttId): JavaType =
  CASES aid OF
    Node?elem: ct(Node),
    Node?pred: ct(Node),
    Node?succ: ct(Node),
    DList?firstNode: ct(DList),
    DList?lastNode: ct(DList)
  ENDCASES

AttId: DATATYPE WITH SUBTYPES CAttId
BEGIN
  Node?Node?succ : Node?Node?succ? : CAttId
  Node?Node?pred : Node?Node?pred? : CAttId
  Node?Node?elem : Node?Node?elem? : CAttId
  NodeL?Node?succ : NodeL?Node?succ? : CAttId
  NodeL?Node?pred : NodeL?Node?pred? : CAttId
  NodeL?Node?elem : NodeL?Node?elem? : CAttId
  DList?DList?lastNode : DList?DList?lastNode? : CAttId
  DList?DList?firstNode : DList?DList?firstNode? : CAttId
END AttId

mkatt(sai: SimpleAttId, cid: { X: CTypeId | ct(X) <= dtype(sai)}): CAttId =
  CASES cid OF
  Node:
    CASES sai OF
      Node?succ: Node?Node?succ,
      Node?pred: Node?Node?pred,
      Node?elem: Node?Node?elem
    ENDCASES,
  NodeL:
    CASES sai OF
      Node?succ: NodeL?Node?succ,
      Node?pred: NodeL?Node?pred,
      Node?elem: NodeL?Node?elem
    ENDCASES,
  DList:
    CASES sai OF
      DList?lastNode: DList?DList?lastNode,
      DList?firstNode: DList?DList?firstNode
    ENDCASES  ENDCASES

rtype(aid: AttId): JavaType =
  CASES aid OF
    Node?Node?succ: ct(Node),
    Node?Node?pred: ct(Node),
    Node?Node?elem: intType,
    NodeL?Node?succ: ct(Node),
    NodeL?Node?pred: ct(Node),
    NodeL?Node?elem: intType,
    DList?DList?lastNode: ct(NodeL),
    DList?DList?firstNode: ct(NodeL)
  ENDCASES
```

```
otypec(aid: CAttId): (ct?) =
  CASES aid OF
    Node?Node?succ: ct(Node),
    Node?Node?pred: ct(Node),
    Node?Node?elem: ct(Node),
    NodeL?Node?succ: ct(NodeL),
    NodeL?Node?pred: ct(NodeL),
    NodeL?Node?elem: ct(NodeL),
    DList?DList?lastNode: ct(DList),
    DList?DList?firstNode: ct(DList)
  ENDCASES

AAttId: TYPE FROM AttId
otypea: [AAttId -> (at?)]

END Attributes


dlPrelude: THEORY
BEGIN
jive: LIBRARY = "/users/labeth/jive/TPC/theories"
IMPORTING Attributes, jive@StoreProperties[CTypeId, ATypeId, ITypeId,
    SimpleAttId, AttId, CAttId, AAttId, rtype, dtype, mkatt, otypec,
    otypea]


END dlPrelude
```

C Program-Independent Lemmas

This section contains the program-independent lemmas that occur in the program proofs. We proved all lemmas presented here with the PVS system. However, the proofs are not presented here for brevity.

```
dl_independent: THEORY

  BEGIN

  IMPORTING dl

  Node1_1: LEMMA
    FORALL (i: (int?), I: JavaInt, env: Store):
    I=aI(i) => EXISTS (S: Store): env=S AND I=aI(i)

  Node1_2: THEOREM
    FORALL (i: (int?), n: Value, env,S: Store, I: JavaInt):
    I=aI(i) AND env=S##Node AND n=new(S, Node)
    => n/=null AND ANode(n, update(env, loc(n, Node?elem), i)) = I

  Node2_1: LEMMA
    FORALL (S,env: Store, T,this: JavaObject):
    NodeObj?(this) AND S=env AND T=this =>
    aI(env@@loc(this, Node?elem)) = ANode(T, S)

  Node4_1: LEMMA
    FORALL (S,env: Store, T,this: JavaObject):
    NodeObj?(this) AND S=env AND T=this =>
    env@@loc(this, Node?pred) = S@@loc(T, Node?pred)

  Node6_1: LEMMA
    FORALL (env,S: Store, this,T: Value):
    NodeRef?(this) AND S=env AND T=this =>
    env@@loc(this, Node?pred) = S@@loc(T, Node?pred)

  Node8_1: LEMMA
    FORALL (env,S: Store, this,T: Value):
    NodeRef?(this) AND S=env AND T=this =>
    env@@loc(this, Node?succ) = S@@loc(T, Node?succ)

  Node10_1: LEMMA
    FORALL (env: Store, this: Value, L: list[int]):
    NodeLRef?(this) AND ANodeL(this, env)=L =>
    ANodeL(env@@loc(this, Node?succ), env) = cdr(L)

  Node11_h1: LEMMA
    FORALL (i: (int?), n,X: Value, env,S,S1: Store):
    n=new(S, Node) AND X/=n AND
    env=S##Node AND S1=update(env, loc(n, Node?elem), i)
    => (S==S1)(X)

  Node11_1: THEOREM
    FORALL (env: Store):
    inv(env)
    => EXISTS (S: Store): env=S AND inv(env)

  Node11_2: THEOREM
    FORALL (env,S: Store):
```

```
  env=S AND inv(env)
  => S=env AND inv(S)

Node11_3: THEOREM
  FORALL (i: (int?), n: Value, env,S,S1: Store):
  n=new(S, Node) AND inv(S) AND
  env=S##Node AND S1=update(env, loc(n, Node?elem), i)
  => n/=null AND inv(S1)

NodeVirt1: LEMMA
  FORALL (this: Value): typeof(this) < ct(Node) => typeof(this) <= ct(NodeL)

NodeL1_1: THEOREM
  FORALL (i: (int?), I: JavaInt, env: Store):
  I=aI(i) => EXISTS (S: Store): env=S AND I=aI(i)

NodeL1_2: THEOREM
  FORALL (i: (int?), n: Value, env,S: Store, I: JavaInt):
  I=aI(i) AND env=S##NodeL AND n=new(S, NodeL)
  => n/=null AND ANode(n, update(env, loc(n, Node?elem), i)) = I

NodeL2_1: THEOREM
  FORALL (i: (int?), I: JavaInt, env: Store):
  I=aI(i) => EXISTS (S: Store): env=S AND I=aI(i)

NodeL2_2: THEOREM
  FORALL (i: (int?), n: Value, env,S: Store, I: JavaInt):
  I=aI(i) AND env=S##NodeL AND n=new(S, NodeL)
  => n/=null AND
     ANodeL(n, update(env, loc(n, Node?elem), i)) = cons(I, null)

NodeL3_h1: LEMMA
  FORALL (i: (int?), n,X: Value, env,S,S1: Store):
  n=new(S, NodeL) AND X/=n AND
  env=S##NodeL AND S1=update(env, loc(n, Node?elem), i)
  => (S==S1)(X)

NodeL3_1: THEOREM
  FORALL (env: Store):
  inv(env)
  => EXISTS (S: Store): env=S AND inv(env)

NodeL3_2: THEOREM
  FORALL (env,S: Store):
  env=S AND inv(env)
  => S=env AND inv(S)

NodeL3_3: THEOREM
  FORALL (i: (int?), n: Value, env,S,S1: Store):
  n=new(S, NodeL) AND inv(S) AND
  env=S##NodeL AND S1=update(env, loc(n, Node?elem), i)
  => n/=null AND inv(S1)

NodeL5_h1: LEMMA
  FORALL (X: Value ,n,this: NodeRef, env,S: Store):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this)
  => alive(X, env) = alive(X, S) AND
     alive(X, env) = alive(X, update(env,loc(this,Node?succ),n))
```

```
NodeL5_h2: LEMMA
  FORALL (this,n: NodeRef, env,S: Store):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
  alive(this, env) AND alive(n, env)
  => S@@loc(n, Node?pred) = this AND  S@@loc(this, Node?succ) = n

NodeL5_h3: LEMMA
  FORALL (X,n,this: NodeRef,
          sai: {s: SimpleAttId | ct(tid(X))<=dtype(s)},
          env,S: Store):
  (X/=n AND sai=Node?pred OR X/=this AND sai=Node?succ OR
   X=n AND sai/=Node?pred AND n/=this OR
   X=this AND sai/=Node?succ AND this/=n) AND
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this)
  => S@@loc(X, sai) = env@@loc(X, sai)

NodeL4_1: THEOREM
  FORALL (n,this,X: NodeLRef, L,M: list[int], env,S: Store, k: nat):
  n/=null AND n/=this AND fstNode?(n,env) AND lstNode?(n,env) AND
  this/=null AND lstNode?(this,env) AND
  alive(n,env) AND alive(this,env) AND alive(X,env) AND
  ANodeL(X,env)=L AND ANodeL(n,env) = M AND
  succn(X,env,k)=this AND
  invNodeL(env) AND
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this)
  =>
  this/=null AND n/=null AND ANodeL(X,S) = append(L,M)

NodeL5_h4: LEMMA
  FORALL (X: DListRef, this,n: NodeRef, env,S: Store):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this)
  => env@@loc(X, DList?firstNode) = S@@loc(X, DList?firstNode) AND
     env@@loc(X, DList?lastNode)  = S@@loc(X, DList?lastNode)

NodeL5_h5: LEMMA
  FORALL (X: NodeLObj, n,this: NodeRef, env,S: Store, k: nat):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
  X/=n AND alive(X, env) AND lstNode?(n, env) AND invNodeL(env)
  => predn(X, env, k) /= n

NodeL5_h6: LEMMA
  FORALL (X: NodeObj, this,n: NodeRef, env,S: Store, k: nat):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
 (FORALL (m: {x:nat | x<=k}):  predn(X,env,m)/=n)
  => predn(X,env,k)=predn(X,S,k)

NodeL5_h7: LEMMA
  FORALL (X: NodeObj, this,n: NodeRef, env,S: Store, k: nat):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
  (FORALL (m: {x:nat | x<=k}):  succn(X,env,m)/=this)
  => succn(X,env,k)=succn(X,S,k)

NodeL5_h8: LEMMA
  FORALL (X: NodeObj, n,this: NodeRef, env,S: Store, k: nat):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
  alive(X, S) AND alive(this, env) AND alive(n, env) AND
  lstNode?(this, env) AND succn(X, env, k) = this
```

```
  => succn(X, S, k) = this

NodeL5_1: THEOREM
  FORALL (n,this: NodeLRef, env,S: Store):
  S=update(update(env,loc(this,Node?succ),n), loc(n,Node?pred),this) AND
  n/=this AND fstNode?(n, env) AND lstNode?(n, env) AND
  lstNode?(this, env) AND alive(this, env) AND alive(n, env) AND
  inv(env)
  => n/=null AND n/=this AND inv(S)

NodeL6_1: LEMMA
  FORALL (n: Value, env,S: Store, i: (int?)):
  env=S##NodeL AND n=new(S,NodeL)
  =>
  n/=null AND
  update(env, loc(n, Node?elem), i)=update(S##NodeL,loc(n, Node?elem), i) AND
  n=new(S,NodeL)

NodeL7_1: LEMMA
  FORALL (this,n,T,N: NodeObj, env,S: Store):
  this/=null AND n/=null AND n/=this AND
  fstNode?(n,env) AND lstNode?(n,env) AND lstNode?(this,env) AND
  env=S AND T=this AND N=n
  =>
  this/=null AND n/=null AND
  update(update(env, loc(this,Node?succ), n), loc(n,Node?pred), this)=
  update(update(S, loc(T,Node?succ), N), loc(N,Node?pred), T)

NodeL8_1: LEMMA
  FORALL (this,T: NodeObj, env,S: Store):
  this/=null AND T=this AND env=S
  =>
  this/=null AND env=S AND
  (EXISTS (N: nat): succn(T,S,N)=this AND
                    succn(T,S,N+1)=env@@loc(this,Node?succ))

NodeL8_2: LEMMA
  FORALL (f,l,T: NodeObj, env,S: Store, b: (bool?)):
  aB(b) AND aB(b)=(l/=null) AND env=S AND f/=null AND
  (EXISTS (N: nat): succn(T,S,N)=f AND succn(T,S,N+1)=l)
  =>
  env=S AND l/=null AND
  (EXISTS (N: nat): succn(T,S,N)=l AND succn(T,S,N+1)=env@@loc(l,Node?succ))

NodeL8_3: LEMMA
  FORALL (f,l,T: NodeObj, env,S: Store, b: (bool?)):
  aB(b)=(l/=null) AND env=S AND f/=null AND NOT aB(b) AND
  (EXISTS (N: nat): succn(T,S,N)=f AND succn(T,S,N+1)=l)
  =>
  env=S AND f/=null AND
  (EXISTS (N: nat): succn(T,S,N)=f AND succn(T,S,N+1)=null)

NodeL9_1: LEMMA
FORALL (this: Value, env: Store):
inv(env)
=>
EXISTS (S: Store, T: Value):
this=T AND env=S AND inv(env)
```

```
NodeL9_2: LEMMA
FORALL (this,T: Value, env,S: Store):
T=this AND env=S AND inv(env)
=>
T=this AND env=S AND inv(S)

NodeL9_3: LEMMA
FORALL (T,result: NodeObj, env,S: Store):
env=S AND result/=null AND
(EXISTS (N: nat): succn(T,S,N)=result AND succn(T,S,N+1)=null)
=>
env=S

NodeL9_4: LEMMA
FORALL (env,S: Store):
env=S AND inv(S)
=>
inv(env)

NodeLVirt1: LEMMA
FORALL (this: Value):
typeof(this)<ct(NodeL) AND this/=null => FALSE

DList1_1: THEOREM
  FORALL (this,T: DListRef, env: Store):
  this/=null AND alive(this, env) AND T=this
  => this/=null AND
     ADList(T, update(env, loc(this, DList?firstNode), null)) = null

DList2_1: THEOREM
  FORALL (this: DListRef, env,S: Store):
  alive(this, env) AND inv(env) AND
  S = update(env, loc(this, DList?firstNode), null)
  => inv(S)

DList3_1: THEOREM
  FORALL (env: Store):
  TRUE => EXISTS (S: Store): env=S

DList3_2: THEOREM
  FORALL (d: Value, env,S: Store):
  env=S##DList AND d=new(S, DList) => ADList(d, env) = null

DList4_2: THEOREM
  FORALL (d,X: Value, env,S: Store):
  env=S##DList AND d=new(S, DList) AND alive(X, S)
  => (env==S)(X)

DList5_1: THEOREM
  FORALL (env: Store):
  inv(env)
  => EXISTS (S: Store): env=S AND inv(env)

DList5_2: THEOREM
  FORALL (env,S: Store):
  env=S AND inv(env)
  => env=S AND inv(S)
```

```
DList5_3: THEOREM
  FORALL (d: Value, env,S: Store):
  env=S##DList AND d=new(S, DList) AND inv(S)
  => inv(env)

DList6_1: THEOREM
  FORALL (this: DListRef, L: list[int], env: Store):
  ADList(this, env)/=null AND ADList(this, env) = L
  => this/=null AND env@@loc(this, DList?firstNode)/=null AND
     ADList(this, env) = L

DList6_2: THEOREM
  FORALL (this: DListRef, L: list[int], env: Store):
  this/=null AND env@@loc(this, DList?firstNode)/=null
  AND ADList(this, env) = L
  => EXISTS (S: Store, T: Value):
     S = env AND T = env@@loc(this, DList?firstNode) AND
     this/=null AND env@@loc(this, DList?firstNode)/=null AND
     ADList(this, env) = L AND T/=null

DList6_3: THEOREM
  FORALL (this: DListRef, T: Value, L: list[int], env,S: Store):
  this/=null AND env@@loc(this, DList?firstNode)/=null AND
  env=S AND T=env@@loc(this, DList?firstNode) AND
  ADList(this, S)=L AND T/=null
  =>
  this/=null AND env@@loc(this, DList?firstNode)/=null AND
  env=S AND T=env@@loc(this, DList?firstNode) AND
  ADList(this, S)=L AND T/=null AND
  T=S@@loc(this, DList?firstNode)

DList6_4: THEOREM
  FORALL (k: (int?), this: DListRef, T: Value, L: list[int], S: Store):
  aI(k) = ANode(T, S) AND ADList(this, S) = L AND T/=null AND
  T = S@@loc(this, DList?firstNode)
  => aI(k) = car(L)

DList7_1: THEOREM
  FORALL (this: DListRef, env,S:Store):
  this/=null AND env=S AND ADList(this, env)/=null
  => this/=null AND env=S

DList8_1: THEOREM
  FORALL (this: DListRef, env: Store):
  this/=null AND ADList(this, env)/=null AND inv(env)
  => EXISTS (S: Store): env=S AND this/=null AND
     ADList(this, env)/=null AND inv(env)

DList8_2: THEOREM
  FORALL (this: DListRef, env,S: Store):
  env=S AND this/=null AND ADList(this, env)/=null AND inv(env)
  => env=S AND this/=null AND ADList(this, env)/=null AND
     inv(S)

DList8_3: THEOREM
  FORALL (env,S: Store):
  env=S AND inv(S)
```

```
  => inv(env)

DList9_1: THEOREM
  FORALL (f,l: NodeObj, X: Value, env,S: Store):
  wf2Nodes(f, l, env) AND S=env AND alive(X, S)
  => env=S AND alive(X, S)

DList9_2: THEOREM
  FORALL (f,l: NodeObj, d,X: Value, env,S,S2: Store):
  env=S##DList AND d=new(S, DList) AND alive(X, S) AND
  S2 = update(update(env, loc(d, DList?firstNode), f),
                          loc(d, DList?lastNode),  l)
  => d/=null AND (S==S2)(X)

DList10_1: THEOREM
  FORALL (f,F: NodeLObj, l: NodeObj, d: Value, env,S,S2: Store):
  S=env AND F=f AND wf2Nodes(f, l, env) AND
  alive(f, env) AND invNodeL(env)
  => S=env AND F=f AND alive(F, S) AND invNodeL(S)

DList10_2: THEOREM
  FORALL (f,F: NodeLObj, l: NodeObj, d: Value, env,S,S2: Store):
  env=S##DList AND d=new(S, DList) AND
  F=f AND alive(F, S) AND invNodeL(S) AND
  S2 = update(update(env, loc(d, DList?firstNode), f),
                          loc(d, DList?lastNode),  l)
  => d/=null AND ADList(d, S2) = ANodeL(F, S)

DList11_1: THEOREM
  FORALL (f,l: NodeObj, env: Store):
  wf2Nodes(f, l, env) AND alive(f, env) AND alive(l, env) AND inv(env)
  => EXISTS (S: Store):
     env=S AND wf2Nodes(f, l, env) AND
     alive(f, env) AND alive(l, env) AND inv(env)

DList11_2: THEOREM
  FORALL (f,l: NodeObj, env,S: Store):
  wf2Nodes(f, l, env) AND env=S AND
  alive(f, env) AND alive(l, env) AND inv(env)
  => wf2Nodes(f, l, S) AND env=S AND
     alive(f, S) AND alive(l, S) AND inv(S)

DList11_3: THEOREM
  FORALL (f: NodeLObj, l: NodeObj, d: Value, env,S,S2: Store):
  env=S##DList AND d=new(S, DList) AND wf2Nodes(f, l, S) AND
  alive(f, S) AND alive(l, S) AND inv(S) AND
  S2 = update(update(env, loc(d, DList?firstNode), f),
                          loc(d, DList?lastNode),  l)
  => d/=null AND inv(S2)

DList12_1: THEOREM
  FORALL (this: DListObj, env: Store, L: list[int]):
  this/=null AND ADList(this,env)/=null AND ADList(this,env)=L
  =>
  EXISTS (S: Store):
  env=S AND
  this/=null AND
  env@@loc(this,DList?firstNode)/=null AND
```

```
  ANodeL(env@@loc(this,DList?firstNode),env)=L AND
  L/=null

DList12_2: THEOREM
  FORALL (env,S: Store, f: NodeObj, s: NodeLObj, L: list[int]):
  env=S AND f/=null AND L/=null AND
  s=S@@loc(f,Node?succ) AND ANodeL(s,env)=cdr(L)
  =>
  f/=null AND env=S AND s=S@@loc(f,Node?succ) AND
  L/=null AND ANodeL(s,S)=cdr(L)

DList12_3: THEOREM
  FORALL (f,l: NodeObj, s: NodeLObj, env,S: Store, L: list[int]):
  env=S AND f/=null AND l/=null AND L/=null AND
  s=S@@loc(f,Node?succ) AND ANodeL(s,S)=cdr(L) AND
  (EXISTS (N: nat): succn(f,S,N)=l AND succn(f,S,N+1)=null)
  =>
  wf2Nodes(s,l,env) AND S=env AND ANodeL(s,S)=cdr(L) AND L/=null

DList12_4: THEOREM
  FORALL (d: DListObj, s: NodeObj, env,S: Store, L: list[int]):
  L/=null AND
  ADList(d,env)=ANodeL(s,S) AND ANodeL(s,S)=cdr(L)
  =>
  ADList(d,env)=cdr(L)

DList13_1: THEOREM
  FORALL (this: DListObj, X: Value, env,S: Store):
  this/=null AND ADList(this,env)/=null AND alive(X,env) AND env=S
  =>
  this/=null AND env=S AND
  env@@loc(this,DList?firstNode)/=null AND
  alive(X,S)

DList13_2: THEOREM
  FORALL (f,s: NodeObj, X: Value, env,S: Store):
  f/=null AND env=S AND s=S@@loc(f,Node?succ) AND
  alive(X,S)
  =>
  f/=null AND env=S AND s=S@@loc(f,Node?succ) AND
  alive(X,S)

DList13_3: THEOREM
  FORALL (f,l,s: NodeObj, X: Value, env,S: Store):
  f/=null AND l/=null AND env=S AND
  s=S@@loc(f,Node?succ) AND alive(X,S) AND
  (EXISTS (N: nat): succn(f,S,N)=l AND succn(f,S,N+1)=null)
  =>
  wf2Nodes(s,l,env) AND S=env AND alive(X,env)

DList13_4: THEOREM
  FORALL (X: Value, env,S: Store):
  (S==env)(X) => (env==S)(X)

DList14_1: THEOREM
  FORALL (this: DListObj, env: Store):
  this/=null AND ADList(this,env)/=null AND inv(env)
  =>
```

```
  EXISTS (S: Store): env=S AND
  this/=null AND env@@loc(this,DList?firstNode)/=null AND inv(env)

DList14_2: THEOREM
  FORALL (f,s: NodeObj, env,S: Store):
  f/=null AND env=S AND s=S@@loc(f,Node?succ) AND inv(env)
  =>
  f/=null AND env=S AND inv(env) AND s=S@@loc(f,Node?succ)

DList14_3: THEOREM
  FORALL (f,l,s: NodeObj, env,S: Store):
  f/=null AND l/=null AND inv(env) AND env=S AND s=S@@loc(f,Node?succ) AND
  (EXISTS (N: nat): succn(f,S,N)=l AND succn(f,S,N+1)=null)
  =>
  wf2Nodes(s,l,env) AND inv(env)

DList15_1: THEOREM
  FORALL (this: DListObj, env: Store, L: list[int]):
  this/=null AND L=ADList(this,env)
  =>
  EXISTS (S: Store): env=S AND
  this/=null AND L=ADList(this,env)

DList15_2: THEOREM
  FORALL (this: DListObj, env,S: Store, L: list[int]):
  env=S AND this/=null AND L=ADList(this,env)
  =>
  env=S AND this/=null AND L=ADList(this,env) AND
  env@@loc(this,DList?firstNode)=S@@loc(this,DList?firstNode)

DList15_3: THEOREM
  FORALL (this: DListObj, f: NodeObj, b: bool, env,S: Store, L: list[int]):
  this/=null AND b=(f=null) AND env=S AND L=ADList(this,S) AND
  f=S@@loc(this,DList?firstNode)
  =>
  b=null?(L)

DList16_1: THEOREM
  FORALL (f: Value, b: bool, env,S: Store):
  b=(f=null) AND env=S
  =>
  env=S

DList18_h1: LEMMA
  FORALL (env,S,S1: Store, l: NodeLRef, i: (int?), I: int):
  env=S AND S=update(S1##NodeL, loc(l,Node?elem), i) AND
  l=new(S1,NodeL) AND I=aI(i)
  =>
  S@@loc(l,Node?pred)=null AND S@@loc(l,Node?succ)=null AND
  ANodeL(l,S)=cons(I,null)

DList18_1: LEMMA
  FORALL (env,S1: Store, this,T: DListObj, i: (int?), I: int, L: list[int]):
  this/=null AND
  alive(this,env) AND
  ADList(this,env) = L AND
  I=aI(i) AND
  T=this AND
```

```
  inv(env)
  =>
  EXISTS (S1: Store): env=S1 AND
  inv(env) AND
  this/=null AND
  T=this AND
  I=aI(i) AND
  L=ADList(this,env) AND
  alive(this,S1)

DList18_2: LEMMA
  FORALL (env,S1: Store, this,T: DListObj, i: (int?), I: int, L: list[int]):
  env=S1 AND
  inv(env) AND
  this/=null AND
  T=this AND
  I=aI(i) AND
  L=ADList(this,env) AND
  alive(this,S1)
  =>
  env=S1 AND
  inv(env) AND
  this/=null AND
  T=this AND
  I=aI(i) AND
  L=ADList(this,S1) AND
  alive(this,S1) AND
  inv(S1)

DList18_3: LEMMA
  FORALL (env,S1: Store, T,this: DListObj, l: NodeLRef,
          i: (int?), I: int, L: list[int]):
  env=update(S1##NodeL, loc(l,Node?elem), i) AND
  l=new(S1,NodeL) AND
  inv(env) AND
  this/=null AND
  T=this AND
  I=aI(i) AND
  L=ADList(this,S1) AND
  alive(this,S1) AND
  inv(S1)
  =>
  EXISTS (S: Store): env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  L=ADList(this,env) AND
  l=new(S1,NodeL) AND
  T=this AND
  I=aI(i) AND inv(S) AND
  L=ADList(this,S) AND
  alive(this,S)

DList18_4: LEMMA
  FORALL (env,S,S1: Store, T,this: DListObj, l: NodeLRef,
          i: (int?), I: int, L: list[int], b: bool):
  env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
```

```
  l=new(S1,NodeL) AND
  T=this AND I=aI(i) AND inv(S) AND
  L=ADList(this,S) AND b=null?(L) AND
  alive(this,S)
  =>
  EXISTS (X: Value): X=S@@loc(this,DList?firstNode) AND
  env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  l=new(S1,NodeL) AND
  T=this AND I=aI(i) AND inv(S) AND
  L=ADList(this,env) AND b=null?(L) AND
  alive(this,env)

DList18_5: LEMMA
  FORALL (env,S,S1: Store, T,this: DListObj, X: NodeObj, l: NodeLRef,
          i: (int?), I: int, L: list[int], b: bool):
  env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  l=new(S1,NodeL) AND X=S@@loc(this,DList?firstNode) AND
  T=this AND I=aI(i) AND inv(S) AND
  L=ADList(this,env) AND b=null?(L) AND b AND
  alive(this,env)
  =>
  this/=null AND
  ADList(T,update(update(env, loc(this,DList?lastNode), l),
                  loc(this,DList?firstNode),
                  l))
  =append(L,cons(I,null))

DList18_6: LEMMA
  FORALL (env,S,S1: Store, T,this: DListObj, X: NodeObj, l: NodeLRef,
          i: (int?), I: int, L: list[int], b: bool):
  env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  l=new(S1,NodeL) AND X=S@@loc(this,DList?firstNode) AND
  T=this AND I=aI(i) AND inv(S) AND
  L=ADList(this,env) AND b=null?(L) AND NOT b AND
  alive(this,env)
  =>
  env@@loc(this,DList?firstNode)/=null AND
  env=S AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  l=new(S1,NodeL) AND
  env@@loc(this,DList?firstNode)=S@@loc(this,DList?firstNode) AND
  X=env@@loc(this,DList?firstNode) AND
  T=this AND I=aI(i) AND inv(S) AND
  ANodeL(X,S)=L

DList18_7: LEMMA
  FORALL (env,S,S1: Store, T,this: DListRef, f,h,X: NodeRef, l: NodeLRef,
          i: (int?), I: int, L: list[int]):
  env=S AND S=update(S1##NodeL, loc(l,Node?elem), i) AND
  l=new(S1,NodeL) AND I=aI(i) AND
  f=S@@loc(this,DList?firstNode) AND X=f AND T=this AND
```

```
  ANodeL(X,S)=L AND inv(S) AND
  (EXISTS (n: nat): succn(f,S,n)=h AND succn(f,S,n+1)=null)
  =>
  EXISTS (k: nat):
  succn(X,env,k)=h AND ANodeL(X,S)=L AND
  l/=null AND l/=h AND fstNode?(l,env) AND lstNode?(l,env) AND
  h/=null AND lstNode?(h,env) AND succn(X,env,k)=h AND env=S AND
  inv(env) AND ANodeL(l,S)=cons(I,null) AND this/=null AND
  X=update(update(S,loc(h,Node?succ),l),
           loc(l,Node?pred),h)@@loc(T,DList?firstNode)

DList18_8: LEMMA
  FORALL (env,S: Store, T,this: DListRef, h,X: NodeRef, l: NodeLRef,
          I: int, L: list[int], k: nat):
  l/=null AND l/=h AND fstNode?(l,env) AND lstNode?(l,env) AND
  h/=null AND lstNode?(h,env) AND succn(X,env,k)=h AND
  env=S AND inv(env) AND
  X=update(update(S,loc(h,Node?succ),l),
           loc(l,Node?pred),h)@@loc(T,DList?firstNode) AND
  ANodeL(X,S)=L AND ANodeL(l,S)=cons(I,null) AND this/=null
  =>
  l/=null AND l/=h AND fstNode?(l,env) AND lstNode?(l,env) AND
  h/=null AND lstNode?(h,env) AND succn(X,env,k)=h AND
  ANodeL(X,env)=ANodeL(X,S) AND ANodeL(l,env)=ANodeL(l,S) AND
  env=S AND inv(env) AND
  X=update(update(S,loc(h,Node?succ),l),
           loc(l,Node?pred),h)@@loc(T,DList?firstNode) AND
  ANodeL(X,S)=L AND ANodeL(l,S)=cons(I,null) AND this/=null

DList18_9: LEMMA
  FORALL (env,S: Store,
          T,this: DListRef, l: NodeLRef, h: NodeRef, X: NodeObj,
          L: list[int], I: int):
  ANodeL(X,S)=L AND ANodeL(l,S)=cons(I,null) AND
  ANodeL(X,env)=append(ANodeL(X,S),ANodeL(l,S)) AND
  env=update(update(S,loc(h,Node?succ),l),
           loc(l,Node?pred),h) AND
  X=env@@loc(T,DList?firstNode) AND
  inv(env)
  =>
  this/=null AND
  ADList(T,update(env,loc(this,DList?lastNode),l))=append(L,cons(I,null))

DList19_2: LEMMA
  FORALL (env: Store, this: DListObj):
  inv(env) AND
  this/=null AND
  alive(this,env)
  =>
  EXISTS (S1: Store):
  (EXISTS (L: list[int]):
  env=S1 AND
  inv(env) AND
  this/=null AND
  ADList(this,S1)=L AND
  alive(this,S1))

DList19_3: LEMMA
```

```
  FORALL (env,S1: Store, l: NodeObj, this: DListObj,
          i: (int?), L: list[int]):
  l=new(S1,NodeL) AND
  env=update(S1##NodeL, loc(l,Node?elem), i) AND
  inv(env) AND
  this/=null AND
  ADList(this,S1)=L AND
  alive(this,S1) AND
  inv(S1)
  =>
  EXISTS (S: Store):
  env=S AND
  this/=null AND
  ADList(this,env)=L AND
  inv(env) AND
  alive(this,env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i)

DList19_4: LEMMA
  FORALL (env,S,S1: Store, l: NodeObj, this: DListObj,
          i: (int?), L: list[int]):
  env=S AND
  this/=null AND
  ADList(this,env)=L AND
  inv(env) AND
  alive(this,env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i)
  =>
  env=S AND
  this/=null AND
  ADList(this,env)=L AND
  inv(S) AND
  alive(this,S) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  ADList(this,S)=L

DList19_5: LEMMA
  FORALL (env,S,S1: Store, l: NodeObj, this: DListObj,
          i: (int?), L: list[int], b: bool):
  env=S AND
  b = null?(L) AND
  inv(S) AND
  this/=null AND
  alive(this,S) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  ADList(this,S)=L
  =>
   env=S AND
  inv(env) AND
  this/=null AND
  alive(this,env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  ADList(this,env)=L AND
```

```
  b=null?(L)

DList19_6: LEMMA
  FORALL (env,S,S1: Store, l: NodeObj, this: DListObj,
          i: (int?), b: bool, L: list[int]):
  env=S AND
  inv(env) AND
  this/=null AND
  alive(this,env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  ADList(this,env)=L AND
  b=null?(L) AND
  b
  =>
  this/=null AND
  inv(update(update(env, loc(this,DList?lastNode), l),
             loc(this,DList?firstNode),
             l))

DList19_7: LEMMA
  FORALL (env,S,S1: Store, l: NodeObj, this: DListObj,
          i: (int?), b: bool, L: list[int]):
  env=S AND
  inv(env) AND
  this/=null AND
  alive(this,env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  ADList(this,env)=L AND
  b=null?(L) AND
  NOT b
  =>
  env=S AND
  inv(env) AND
  this/=null AND
  env@@loc(this,DList?firstNode)/=null AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  l=new(S1,NodeL) AND
  env@@loc(this,DList?firstNode)=S@@loc(this,DList?firstNode)

DList19_8: LEMMA
  FORALL (env,S,S1: Store, f,h,l: NodeObj, this: DListObj, i: (int?)):
  env=S AND
  h/=null AND
  (EXISTS (N: nat):
   succn(f,S,N)=h AND
   succn(f,S,N+1)=null) AND
  inv(env) AND
  l=new(S1,NodeL) AND
  S=update(S1##NodeL, loc(l,Node?elem), i) AND
  this/=null AND
  f=S@@loc(this,DList?firstNode) AND
  f/=null
  =>
  EXISTS (k: nat):
  h/=null AND l/=null AND
  l/=h AND
```

```
    fstNode?(l,env) AND
    lstNode?(l,env) AND
    lstNode?(h,env) AND
    inv(env) AND
    env=S AND
    this/=null AND f/=null AND
    f=S@@loc(this,DList?firstNode) AND
    succn(f,S,k)=h AND
    lstNode?(h,S) AND
    alive(l,S) AND
    alive(h,S)

  DList19_9h1: LEMMA
    FORALL (f,h,l: NodeObj, env,S: Store, n: nat):
    h/=null AND
    lstNode?(h,S) AND
    f/=null AND
    l/=null AND
    env=update(update(S,loc(h,Node?succ),l),loc(l,Node?pred),h) AND
    succn(f,S,n)=h
    =>
    succn(f,env,n)=h

  DList19_9h2: LEMMA
    FORALL (h,l: NodeRef, X,this: DListRef, S,S1,S2: Store):
    S1=update(update(S, loc(h, Node?succ), l), loc(l, Node?pred), h) AND
    S2=update(S1, loc(this, DList?lastNode), l)
    =>
    S1@@loc(X, DList?firstNode) = S@@loc(X, DList?firstNode) AND
    S1@@loc(X, DList?lastNode)  = S@@loc(X, DList?lastNode)  AND
    S2@@loc(X, DList?firstNode) = S1@@loc(X, DList?firstNode) AND
    (this=X AND alive(X, S2) AND alive(l, S1) =>
       S2@@loc(X, DList?lastNode) = l) AND
    (this/=X =>
       S2@@loc(X, DList?lastNode) = S1@@loc(X, DList?lastNode)) AND
    (alive(h, S1) AND alive(l, S1) =>
       S1@@loc(h, Node?succ) = l) AND
    (FORALL (Y: Value):
       alive(Y, S2) = alive(Y, S1) AND
       alive(Y, S1) = alive(Y, S))

  DList19_9: LEMMA
    FORALL (h,f,l: NodeLObj, this: DListObj, env,S: Store, k: nat):
    this/=null AND
    h/=null AND f/=null AND l/=null AND
    inv(env) AND
    env=update(update(S,loc(h,Node?succ),l),loc(l,Node?pred),h) AND
    f=S@@loc(this,DList?firstNode) AND
    succn(f,S,k)=h AND
    lstNode?(h,S) AND
    alive(l, env) AND
    alive(h, env)
    =>
    this/=null AND
    inv(update(env,loc(this,DList?lastNode),l))

END dl_independent
```

References

[COR+95] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. *A Tutorial Introduction to PVS*, April 1995.

[GH93] J. V. Guttag and J. J. Horning. *Larch: Languages and Tools for Formal Specification.* Springer-Verlag, 1993.

[MMPH97] P. Müller, J. Meyer, and A. Poetzsch-Heffter. Programming and interface specification language of JIVE — specification and design rationale. Technical Report 223, Fernniversität Hagen, 12 1997.

[MPH97] P. Müller and A. Poetzsch-Heffter. Formal specification techniques for object-oriented programs. In M. Jarke, K. Pasedach, and K. Pohl, editors, *Informatik 97: Informatik als Innovationsmotor*, Informatik Aktuell. Springer-Verlag, 1997.

[MPH00a] J. Meyer and A. Poetzsch-Heffter. An architecture for interactive program provers. In S. Graf and M. Schwartzbach, editors, *Tools ans Algorithms for the Construction and Analysis of Software (TACAS)*, volume 276 of *Lecture Notes in Computer Science*, pages 63–77, 2000.

[MPH00b] J. Meyer and A. Poetzsch-Heffter. Strategies for the verification of object-oriented programs. Presented at 'The 4th Workshop on Tools for System Design and Verification'. Available from `www.informatik.fernuni-hagen.de/pi5/publications.html`, April 2000.

[OSR93] S. Owre, N. Shankar, and J. M. Rushby. The PVS specification language (beta release). Technical report, Computer Science Laboratory SRI International, April 1993.

[PH97] A. Poetzsch-Heffter. Specification and verification of object-oriented programs. Habilitation thesis, Technical University of Munich, Jan. 1997. URL: `www.informatik.fernuni-hagen.de/pi5/publications.html`.

[PHM98] A. Poetzsch-Heffter and P. Müller. Logical foundations for typed object-oriented languages. In D. Gries and W. De Roever, editors, *Programming Concepts and Methods (PROCOMET)*, 1998.

[PHM99] A. Poetzsch-Heffter and P. Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *Programming Languages and Systems (ESOP '99)*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176. Springer-Verlag, 1999.