CINV

The CINV tool

Edition 0.1, January 2010

by Cezara Dragoi and Mihaela Sighireanu

Table of Contents

1 CINV Copying Conditions (LGPL)

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does *Less* to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

   c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

   d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those

countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and an idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or modify it
under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2.1 of the License, or (at
your option) any later version.

This library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
USA.
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the library
'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

2 Introduction

The CINV tool provides several *abstract domains* for abstract reachability analysis of programs manipulating singly linked lists with numerical contents.

CINV generates for each control point specifications which constrain both the shape of the list and the data inside the list. In the present version, two kinds of specifications can be generated: (1) specifications relating data, lenghts, and sums of the data of the list and (2) specifications relating lenghts, data, and universal properties on the list segments.

The input of CINV is an SPL program containing an initial condition on the lists used by the program. Another input of CINV is the `cinv.txt` file giving the maximum number of simple nodes on the heap graph.

The output is the program annotated by program specifications given on files with extension `.shp`. These files contain a list of constrained heap graphs, i.e., in constraint is given in the form of a graph and a numerical or logical constraint relating the data, the sum of data, and the length of list segments in the graph.

We provide in the following more details on the inputs and output of CINV as well as the presentation of the results obtained when applying CINV on our benchmark.

2.1 C code

Each example is given as a C function. The function has at least one list parameter of type `intlist`. The C definition of type `intlist` corresponds to a singly linked list with an integer data field as follows:

```
#include <stdio.h>
typedef struct intlist_ * intlist;
struct intlist_ {
  int data;
  intlist next;
};
```

The C code given for examples corresponds to a desired future input of the tool. However, it cannot be used for the moment as it is because the statements and the expressions allowed are not elementary. For instance, composed terms (e.g., `x->next->data`) and statements (e.g., `x=y` with `x` not pointing to `NULL`) are used.

The C functions are specified using the logic presented in Section 2.3 [Specification logic], page 10.

2.2 Spl encoding

The Spl language is the input language of the Interproc tool. Since Spl deals only with numeric (integer or real) variables, we encode our programs on lists as follows:

- Variables of type `intlist` are coded by real variables.
- Data variables are encoded by integer variables. By convention, length variables are the first **two** integer variables. (This is a constant fixed in the code.) The other integer variables are considered data variables. This separation of length and data variables is used only by the domains which deal differently with these variables, e.g., the `LSUM-PRD` or the `MSET-PRD` domains (see [Domains], page 11).
- The following real variables shall be present in any Spl encoding program in the first positions of the declaration list for real variables: `_data`, `_free`, `_len`, `_new`, `_next`, and `_null`. They are used to encode operations on list variables, e.g., the `data` field access for a list

variable `x`, `x->data`, is encoded into the expression `x*_data`. Similarly, the `_next` variable is used to encode the `next` field access. The `_free` (resp. `_new`) variable is used to encode the `free` (resp. `new`) statement for the memory deallocation (resp. allocation) of pointers. The `_len` variable cannot be used for the moment. The `_null` variable encodes the predefined `NULL` constant in C.

- All statements are elementary: (1) the only terms used on pointer variables are `x`, `x->data`, and `x->next`, (2) the statements have as left hand side one of the terms above, and (3) when terms `x` and `x->next` are assigned, they have to be `NULL`.
- Since Spl considers only numerical variables, the left hand side of an assignment shall be a variable. However, to assign fields of list variables, we need expressions for the left hand sides of assignments, e.g., `x->data` encoded into `x * _data`. To encode such assignments we use the divisibility operation on reals, i.e., `x->field=expr` is encode by `x=expr/field`.
- The specification properties (see Section 2.3 [Specification logic], page 10) of the code are encoded into an initial `assume` statement of the form `assume(x==<code>);` with the following semantics:

x==0 $acyclic(x)$ and $l[x] = l$ and $data(x)$, e.g. $data(x) : S[x] = S$ with S a program data variable, or $data(x) : M[x] = M$ with M a ghost multiset variable

x==1 $acyclic(x)$ and $l[x] + l[y] = l$ and $l >= 1$ and $data(x, y)$ and $reach(x, y)$, e.g., $data(x, y) : S[x] + S[y] = S$

x==2 $acyclic(x)$ and $l[x] = l$ and $data(x)$ and $acyclic(y)$ and $l[y] = l$ and $data(y)$ and $_l >= 1$ and $disjoint(x, y)$

x==3 $acyclic(x)$ and $l[x] = l$ and $data(x)$ and $acyclic(y)$ and $l[y]+1 <= l$ and $data(y)$ and $_l >= 1$ and $disjoint(x, y)$

x==4 $acyclic(x)$ and $l[x] = l$ and $data(x)$ and $acyclic(y)$ and $l[y] = l$ and $data(y)$ and $acyclic(z)$ and $l[z] = l$ and $data(z)$ and $l >= 1$ and $disjoint(x, y, z)$

2.3 Specification logic

The initial constraint on the program analysed is given in a logic which is a restriction of the CSL logic defined in [Bouajjani and al. CONCUR-09]. This logic is a multi-sorted first order logic with reachability predicates. More precisely, in this logic one can use the following terms:

l[n] the length of the heap segment stating from node `n`, i.e., the number of edges of the segment.

d(n) the data stored in the node `n`.

S(n) the sum of the data stored in the heap segment starting from node `n` except `n` itself; we denote by `S[n]=S(n)+d(n)`.

M(n) the multiset of data stored in the heap segment starting from node `n` except `n` itself; we denote by `M[n]=M(n) U d(n)`.

The atomic constraints of the logic are the following:

x(n) variable `x` is labeling a node of a heap called `n`.

expr op 0 where op in $=, ! =, <=, >=, ! =, <, >$ is a linear constraints on terms.

acyclic(x)

variable `x` labels a node from which starts a segment which is acyclic.

reach(x,y)
: variable `x` labels a node from which starts a segment which reaches another node labeled by `y`.

disjoint(x,y)
: variable `x` labels a node from which starts a segment which is disjoint from (does not share nodes with) the list segment which starting node is labeled by `y`.

2.4 Parameters of the analysis

The analysis done by the CINV tool is parametrized by the following inputs:

- *Domain:* The abstract domain used to represent heap segments. This domain is used by the global domain of `Shapes`. The following domains are implemented in CINV:

  LSUM-PRD the domain of sums over heap segments which is a Cartesian product of a domain for lengths of segments and a domain for data of segments.

  LSUM-REL the domain of sums over heap segments where lengths and data are put together.

  MSET-PRD the domain of multisets over heap segments which is a Cartesian product of a domain for lengths of segments, a domain for data of segments, and a multiset domain.

  MSET-REL the domain of multisets over heap segments where lengths and data are put together.

  UCONS the domain of universally constrained heap segments; this domain is parametrized by the set of patterns used by the universally quantified constraints. Actually, for efficiency reasons, the following patterns are implemented:

  P11 $\forall y \in n \Rightarrow ...$

  P21 $\forall y1 \in n, y2 \in m, y1 = y2 \Rightarrow ...$

  P12 $\forall y1, y2 \in n, y1 < y2 \Rightarrow ...$

  P13 $\forall y1, y2, y3 \in n, y1 <_1 y2 <_1 y3 \Rightarrow ...$

- *Number of simple nodes:* The computation of the post abstract transformer is parameterized by the maximum number of simple nodes (nodes not labeled by a program variable or representing a sharing point) in the heap graph. In CINV, this number is obtained from the following two parameters:

  max_anon the maximum number of simple nodes in a heap segment, and

  segm_anon
  the number of segments shall divide the number of simple nodes.

  These two parameters shall be given (in this order) in the file `cinv.txt` in the directory chosen for the execution of CINV.

2.5 Results

The results are given for each domain and each parameter using:

- *Log file*: is a directory in `sample/log` containing a log file and the files storing the shapes generated

- *Constraint*: is the most interesting constraint synthesized by the analysis; this constraint is given in the specification language (see Section 2.3 [Specification logic], page 10).

In the HTML version of this manual, it is possible to browse the log directory.

3 Examples

3.1 Computing on data

Examples in this class iterate over a list to return some information (data value, pointer inside the list, etc.) on the current list.

3.1.1 First not null

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
intlist fstNot0(intlist x) {
  intlist xi = x;
  while (xi != NULL && xi->data==0) {
    xi = xi->next;
  }
  return xi;
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real,
    _l:int, _k:int, S:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xi = x;
  while xi != _null and (xi* _data == 0) do
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-fstNot0-lsum-prd-01
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge S(n1) = 0 \wedge S[n2] = S \wedge l = l[n1] + l[n2]$
LSUM-REL	Anon=(0,1)	log/intlist-fstNot0-lsum-rel-01
		same as above
MSET	Anon=(0,1)	log/intlist-fstNot0-mset-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge M[n1] + M[n2] = M \wedge l = l[n1] + l[n2]$
UCONS	Anon=(0,1), P11	log/intlist-fstNot0-uconspoly-P11-01
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge \forall y \in n1 \Rightarrow d(y) = 0$

Because we did experiments only with numerical abstract domains which are not able to represent the inequality constraints (e.g., polyhedron), the invariant obtained at the control point corresponding to the end of the loop does not contain the constraint `xi->data!=0`.

3.1.2 Get maximum

C code

Spl encoding

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x)  */
int listMax(intlist x) {
  intlist xi = x;
  int max = x->data;
  while (xi != NULL) {
    if (max < xi->data)
      max = xi->data;
    xi = xi->next;
  }
  return max;
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k:int, S:int, max:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xi = x;
  max = x * _data;
  while xi != _null do
    if (max+1 <= xi* _data) then
      max = xi * _data;
    endif;
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-getMax-lsum-prd-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq max \wedge l = l[n1] + l[n2]$
LSUM-REL	Anon=(0,1)	log/intlist-getMax-lsum-rel-01
		same as above
MSET	Anon=(0,1)	log/intlist-getMax-mset-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq max \wedge M[n1]+M[n2] = M \wedge l = l[n1]+l[n2]$
UCONS	Anon=(0,1), P11	log/intlist-getMax-uconspoly-P11-01
		$x(n1) \wedge d(n1) \leq max \wedge l = l[n1] + l[n2] \wedge \forall y \in n1 \Rightarrow d(y) \leq max$

3.1.3 Sentinel

In its original version *Halbwach-Peron-08*, this program uses a test `xi->data!=m`. We have changed it below to `xi->data<=m` in order to avoid disequality constraints.

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
intlist sentinel(intlist x, int m) {
  intlist xi = x;
  while (xi != NULL && xi->data <= m) {
    xi = xi->next;
  }
  return xi;
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k:int, S:int, m:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xi = x;
  while (xi != _null and xi * _data <= m) do
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-sentinel-lsum-prd-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq m \wedge d(n2) \leq m \wedge l = l[n1] + l[n2]$
LSUM-PRD	Anon=(0,1), m=2	log/intlist-sentinel2-lsum-prd-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq 2 \wedge d(n2) \leq 2 \wedge l = l[n1] + l[n2]$
LSUM-REL	Anon=(0,1)	log/intlist-sentinel-lsum-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq m \wedge d(n2) \leq m \wedge l = l[n1] + l[n2]$
LSUM-REL	Anon=(0,1), m=2	log/intlist-sentinel2-lsum-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq 2 \wedge S[n1] \leq 2l[n1] \wedge d(n2) \leq 2 \wedge l = l[n1]+l[n2]$
MSET		log/intlist-sentinel-mset-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq m \wedge d(n2) \leq m \wedge M = M[n1]+M[n2] \wedge l = l[n1] + l[n2]$
UCONS	Anon=(0,1), P11	log/intlist-sentinel-uconspoly-P11-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq m \wedge d(n2) \leq m \wedge l = l[n1] + l[n2] \wedge \forall y \in n1 \Rightarrow d(y) \leq m$
UCONS	Anon=(0,1), P11, m=2	log/intlist-sentinel2-uconspoly-P11-01
		$x(n1) \wedge xi(n2) \wedge d(n1) \leq 2 \wedge d(n2) \leq 2 \wedge l = l[n1] + l[n2] \wedge \forall y \in n1 \Rightarrow d(y) \leq 2$

3.1.4 List equality

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]==_l and data(y) and
 * disjoint(x,y) */
int equal(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL && yi != NULL &&
         xi->data == yi->data) {
    xi = xi->next;
    yi = yi->next;
  }
  if (xi==NULL && yi==NULL)
    return 1;
  else
    return 0;
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S:int;
begin
  assume (x == 2);
  xi = _null; yi = _null; z= _null;
  xi = x;
  yi = y;
  while (xi != _null and yi != _null and
         xi * _data == yi * _data) do
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
    z = yi * _next;
    yi = _null;
    yi = z;
    z = _null;
  done;
  if (xi == _null and yi == _null) then
    _k = 1;
  else
    _k = 0;
  endif;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-equal-lsum-prd-01/ $x(n1) \ \wedge\ y(n3) \ \wedge\ d(n1) = d(n3) \ \wedge\ S(n1) = S(n3) \ \wedge\ l = l[n1] = l[n3]$
LSUM-REL	Anon=(0,1)	log/intlist-equal-lsum-rel-01/ same as above
MSET	Anon=(0,1)	log/intlist-equal-mset-rel-01/ $x(n1) \ \wedge\ y(n2) \ \wedge\ M[n1] = M[n2] \ \wedge\ l = l[n1] = l[n2]$
UCONS	Anon=(0,2), P21	log/intlist-equal-uconspoly-P21-02/ $x(n1) \ \wedge\ y(n2) \ \wedge\ d(n1) = d(n2) \ \wedge\ \forall\ y1 \in n1,\ y2 \in n2,\ y1 = y2 \ \Rightarrow\ d(y1) = d(y2)$

3.1.5 Sum of elements

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x)  */
int listSum(intlist x) {
  intlist xi = x;
  int sum = 0;
  while (xi != NULL) {
    sum = sum + xi->data;
    xi = xi->next;
  }
  return sum;
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k:int, S:int, sum:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xi = x;
  sum = 0;
  while xi != _null do
    sum = sum + xi * _data;
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-sum-lsum-prd-01/
		$x(n1) \ \wedge \ l = l[n1] \ \wedge \ S = S[n1] = v$
LSUM-REL	Anon=(0,1)	log/intlist-sum-lsum-rel-01/
		same as above
MSET	Anon=(0,1)	log/intlist-sum-mset-rel-01/
		$x(n1) \ \wedge \ l = l[n1] \ \wedge \ M[n1] = M$
UCONS	Anon=(0,1), P11	log/intlist-sum-uconspoly-P11-01/
		$x(n1) \ \wedge \ l = l[n1]$

3.2 Initializing data

The examples in this class iterate over a list from its begining and initialize the data fields from scratch, i.e., without using the initial data values of the list.

3.2.1 Initialization modulo 2

The encoding of this example in Spl has been changed in order to replace the boolean variable by an integer variable. The test used in the `if` statement has been changed to avoid disequality constraints.

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void initMod2(intlist x) {
  intlist xi = x;
  bool k = true;
  while (xi != NULL) {
    if (k) xi->data = 1;
    else   xi->data = 0;
    xi = xi->next;
    k = not(k);
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k:int, S:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
   _k = 0;
  xi = x;
  while xi != _null do
    if (_k<=0) then
      xi = 0 / _data;
      _k = 1;
    else
      xi = 1 / _data;
      _k = 0;
    endif;
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-initMod2-lsum-prd-01
		$x(n1) \wedge 0 \leq d(n1) \leq 1 \wedge S(n1) \geq 0$
LSUM-REL	Anon=(0,1)	log/intlist-initMod2-lsum-rel-01
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge 0 \leq k \leq 1 \wedge 2{*}S(n1){+}k \geq l \wedge l \geq S(n1){+}1$
LSUM-REL	Anon=(1,1)	log/intlist-initMod2-lsum-rel-11
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge 0 \leq k \leq 1 \wedge 2 * S(n1) + 1 = l[n1]$
MSET	Anon=(1,1)	log/intlist-initMod2-mset-rel-11
		none
UCONS	Anon=(1,1), P11	log/intlist-initMod2-uconspoly-P11-11
		$x(n1) \wedge xi(n2) \wedge d(n1) = 0 \wedge 0 \leq k \leq 1 \wedge \forall y1 <_1 y2 <_1 y3 \in n1 \Rightarrow d(y1) + d(y2) = 1 \wedge d(y2) + d(y3) = 1 \wedge l = l[n1] + l[n2]$

3.2.2 Initialization with first integers

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void initN(intlist x) {
  intlist xi = x;
  int m = 0;
  while (xi != NULL) {
    xi->data = m;
    xi = xi->next;
    m = m+1;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k: int, S:int, m:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  m = 0;
  xi = x;
  while xi != _null do
    xi = m / _data;
    y = xi*_next;
    xi = _null;
    xi = y;
    y = _null;
    m = m+1;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-initN-lsum-prd-01
		$x(n1) \ \wedge \ d(n1) = 0$
LSUM-REL	Anon=(0,1)	log/intlist-initN-lsum-rel-01
		$x(n1) \ \wedge \ xi(n2) \ \wedge \ d(n1) = 0 \ \wedge \ l[n1] = m \ \wedge \ l = l[n1] + l[n2]$
MSET	Anon=(0,1)	log/intlist-initN-mset-rel-01
		none
UCONS	Anon=(0,1), P11	log/intlist-initN-uconspoly-P11-01
		$x(n1) \wedge \ l = l[n1] \ \wedge \ \forall \ y \ \in \ n1 \ \Rightarrow \ d(y) = y$

3.2.3 Initialization with first even numbers

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void init2N(intlist x) {
  intlist xi = x;
  int m = 0;
  while (xi != NULL) {
    xi->data = m;
    xi = xi->next;
    m = m+2;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, z:real,
    _l:int, _k: int, S:int, m:int;
begin
  assume (x == 0);
  xi = _null; z = _null;
  m = 2;
  xi = x;
  while xi != _null do
    xi = m / _data;
    z = xi*_next;
    xi = _null;
    xi = z;
    z = _null;
    m = m+2;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-init2N-lsum-prd-01
		$x(n1) \ \wedge \ d(n1) = 0$
LSUM-REL	Anon=(0,1)	log/intlist-init2N-lsum-rel-01
		$x(n1) \ \wedge \ 2l[n1] = m - 2 \ \wedge \ d(n1) = 0 \ \wedge \ l = l[n1]$
MSET	Anon=(0,1)	log/intlist-init2N-mset-rel-01
		none
UCONS	Anon=(0,1), P11	log/intlist-init2N-uconspoly-P11-01
		$x(n1) \ \wedge \ d(n1) = 0 \ \wedge \ l = l[n1] \ \wedge \ \forall\, y \ \in \ n1 \ \Rightarrow \ d(y) = 2y$
UCONS	Anon=(1,1), P11	log/intlist-init2N-uconspoly-P11-11
		$x(n1) \ \wedge \ d(n1) = 0 \ \wedge \ l = l[n1] \ \wedge \ \forall\, y \ \in \ n1 \ \Rightarrow \ d(y) = 2y$

3.2.4 Initialization in sequence

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void seqInit(intlist x, int m) {
  int mp = m;
  intlist xi = x;
  while (xi != NULL) {
    xi->data = mp;
    mp = mp+1;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, z:real,
    _l:int, _k: int, S:int, m:int, mp:int;
begin
  assume (x == 0);
  xi = _null; z = _null;
  mp = m;
  xi = x;
  while xi != _null do
    xi = mp / _data;
    z = xi*_next;
    xi = _null;
    xi = z;
    z = _null;
    mp = mp+1;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-initSeq-lsum-prd-01
		$x(n1) \wedge d(n1) = m \wedge mp \geq m+1$
LSUM-REL	Anon=(0,1)	log/intlist-initSeq-lsum-rel-01
		$x(n1) \wedge d(n1) = m \wedge l = l[n1] = mp - m$
MSET	Anon=(0,1)	log/intlist-initSeq-mset-rel-01
		none
UCONS	Anon=(0,1), P11	log/intlist-initSeq-uconspoly-P11-01
		$x(n1) \wedge d(n1) = m \wedge \forall y \in n1 \Rightarrow d(y) = y + m$

3.2.5 Initialization with Fibonacci

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void initFibo(intlist x) {
  int m1 = 1;
  int m2 = 0;
  intlist xi = x;
  while (xi != NULL) {
    xi->data = m1+m2;
    m1 = m2;
    m2 = xi->data;
    xi = xi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l:int, _k:int, S:int, m1:int, m2: int;
begin
  assume (x == 0);
  m1 = 1;
  m2 = 0;
  y = _null;  xi = _null;
  xi = x;
  while xi != _null do
    xi = (m1 + m2)/ _data;
    m1 = m2;
    m2 = xi * _data;
    y = xi * _next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-initFibo-lsum-prd-01
		$x(n1) \wedge d(n1) = 1 \wedge S(n1) + 2 = m1 + 2m2 \wedge m2 \geq m1 \wedge 2m1 + 1 \geq m2 \geq 1$
LSUM-PRD	Anon=(2,1)	log/intlist-initFibo-lsum-prd-21
		$x(n1) \wedge d(n1) = 1 \wedge S(n1) + 2 = m1 + 2m2 \wedge m2 \geq m1 \wedge 2m1 + 1 \geq m2 \geq 15 \wedge 5m1 - 3m2 + 3 \geq 0$
LSUM-REL	Anon=(0,1)	log/intlist-initFibo-lsum-rel-01
		$x(n1) \wedge d(n1) = 1 \wedge S(n1) + 2 = m1 + 2m2 \wedge m2 \geq m1 \wedge 2m1 + 1 \geq m2 \geq 1$
MSET	Anon=(0,1)	log/intlist-initFibo-mset-rel-01
		none
UCONS	Anon=(0,1), P11	log/intlist-initFibo-uconspoly-P11-01
		$x(n1) \wedge d(n1) = 1 \wedge \forall y \in n1 \Rightarrow d(y) \geq y$

3.2.6 Partial reset

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and
 * l[x]+l[y]==_l and data(xy) and
 * reach(x,y) */
void partialInit(intlist x,
                 intlist y) {
  intlist yi = y;
  while (yi != NULL) {
    yi->data = 0;
    yi = yi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 1);
  xi = _null; yi = _null;
  yi = y;
  while yi != _null do
    yi = 0 / _data;
    xi = yi*_next;
    yi = _null;
    yi = xi;
    xi = _null;
  done;
end
```

Results

Domain	**Param.**	**Log file / Interesting constraint**
LSUM-PRD	Anon=(0,1)	log/intlist-pInit-lsum-prd-01/
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1]+l[n2]=l \ \wedge \ S(n2)=0 \ \wedge \ d(n2)=0$
LSUM-REL	Anon=(0,1)	log/intlist-pInit-lsum-rel-01/
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1]+l[n2]=_l \ \wedge S(n2)=0 \ \wedge \ d(n2)=0$
MSET	Anon=(0,1)	log/intlist-pInit-mset-01/
		none
UCONS	Anon=(0,1),P11	log/intlist-pInit-uconspoly-P11-01/
		$x(n1) \ \wedge \ y(n2) \ \wedge \ \forall\, y1 \in \ n2 \ \Rightarrow \ d(y1)=0$

3.2.7 Sum of lists

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and _l==l[x] and data(x) and
 * acyclic(y) and _l==l[y] and data(y) and
 * acyclic(z) and _l==l[z] and data(z) and
 * disjoint(x,y,z) */
void initSum(intlist x,
             intlist y,
             intlist z) {
  intlist xi = x;
  intlist yi = y;
  intlist zi = z;
  while (xi != NULL && yi != NULL && zi != NULL) {
    zi->data = xi->data + yi->data;
    xi = xi->next;
    yi = yi->next;
    zi = zi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real, zi:real, zii:re
    _l:int, _k:int, S: int, T:int;
begin
  assume (x == 4);
  xi = _null; yi = _null; zi = _null; zii = _null;
  xi = x;
  yi = y;
  zi = z;
  while xi != _null and yi != _null and
        zi != _null do
    zi = (xi * _data + yi * _data) / _data;
    zii = xi * _next;
    xi = _null;
    xi = zii;
    zii = _null;
    zii = yi * _next;
    yi = _null;
    yi = zii;
    zii = _null;
    zii = zi * _next;
    zi = _null;
    zi = zii;
    zii = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-initSum-lsum-prd-01/
		$x(n1) \wedge y(n2) \wedge z(n3) \wedge d(n3) = d(n1)+d(n2) \wedge S(n3) = S(n1)+S(n2)$
LSUM-REL	Anon=(0,1)	log/intlist-initSum-lsum-rel-01/
		$x(n1) \wedge y(n2) \wedge z(n3) \wedge d(n3) = d(n1)+d(n2) \wedge S(n3) = S(n1)+S(n2)$
MSET	Anon=(0,1)	log/intlist-initSum-mset-rel-01/
		none
UCONS	Anon=(0,3)	NYI
		$x(n1) \wedge y(n2) \wedge z(n3) \wedge \forall y1 \in n1,\ y2 \in n2,\ y3 \in n3\ y1 = y2 = y3 \Rightarrow d(y3) = d(y1) + d(y2)$

3.3 Changing data

The examples in this class iterate over one or several lists and update the data field based on its old value.

3.3.1 Copy a list (1)

Copy the data of a list into another equal length list.

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]==_l and data(y) and
 * disjoint(x,y) */
void listCopy(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL) {
    yi->data = xi->data;
    xi = xi->next;
    yi = yi->next;
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 2);
  xi = _null; yi = _null; z = _null;
  xi = x; yi = y;
  while xi != _null do
    yi = (xi* _data) / _data;
    z = xi*_next;
    xi = _null;
    xi = z;
    z = _null;
    z = yi*_next;
    yi = _null;
    yi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,2)	log/intlist-copy-eq-lsum-prd-02/
		$x(n1) \;\wedge\; y(n2) \;\wedge\; d(n1) = d(n2) \;\wedge\; d(n1) + S(n1) \;=\; d(n2) + S(n2) \;=\; S$
LSUM-REL	Anon=(0,2)	log/intlist-copy-eq-lsum-rel-02/
		$x(n1) \;\wedge\; y(n2) \;\wedge\; d(n1) = d(n2) \;\wedge\; d(n1) + S(n1) \;=\; d(n2) + S(n2) \;=\; S$
MSET	Anon=(0,2)	log/intlist-copy-eq-mset-rel-02/
		$x(n1) \;\wedge\; y(n2) \;\wedge\; d(n1) = d(n2) \;\wedge\; M[n1] = M[n2] = M$
UCONS	Anon=(0,2), P21	log/intlist-copy-eq-uconspoly-P21-02/
		$x(n1) \wedge y(n2) \wedge d(n1) = d(n2) \wedge \forall\, y1 \in\; n1,\; y2 \in\; n2\; y1 = y2 \;\Rightarrow\; d(y1) = d(y2)$

3.3.2 Copy a list (2)

This example is the correct version of copying the data of a list into another list of different length.

C code

Spl encoding

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]+1<=_l and data(y) and
 * disjoint(x,y) */
void listCopy(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL && yi != NULL) {
    yi->data = xi->data;
    xi = xi->next;
    yi = yi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 3);
  xi = _null; yi = _null; z = _null;
  xi = x; yi = y;
  while xi != _null and yi != _null do
    yi = (xi* _data) / _data;
    z = xi*_next;
    xi = _null;
    xi = z;
    z = _null;
    z = yi*_next;
    yi = _null;
    yi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-copy-neq-lsum-prd-01/ $x(n1) \wedge xi(n2) \wedge y(n3) \wedge yi = null \wedge d(n1) = d(n3) \wedge S(n1) = S(n3)$
LSUM-REL	Anon=(0,1)	log/intlist-copy-neq-lsum-rel-01/ $x(n1) \wedge xi(n2) \wedge y(n3) \wedge yi = null \wedge d(n1) = d(n3) \wedge S(n1) = S(n3)$
MSET	Anon=(0,1)	log/intlist-copy-neq-mset-rel-01/ $x(n1) \wedge xi(n2) \wedge y(n3) \wedge yi = null \wedge d(n1) = d(n3) \wedge M(n1) = M(n3)$
UCONS	Anon=(0,1),P21	log/intlist-copy-neq-uconspoly-P21-01/ $x(n1) \wedge xi(n2) \wedge y(n3) \wedge yi = null \wedge d(n1) = d(n3) \wedge \forall\, y1 \in n1,\ y2 \in n2.\ y1 = y2 \Rightarrow d(y1) = d(y2)$

3.3.3 Copy a list (3)

This example is the erroneous version of copying the data of a list into another list of different length. CINV reports a dereference of a NULL pointer. The invariant generated at the end of the loop is bottom.

C code **Spl encoding**

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]+1<=_l and data(y) and
 * disjoint(x,y) */
void listCopy(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL /* error */) {
    yi->data = xi->data;
    xi = xi->next;
    yi = yi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 3);
  xi = _null; yi = _null; z = _null;
  xi = x; yi = y;
  while xi != _null do
    yi = (xi* _data) / _data;
    z = xi*_next;
    xi = _null;
    xi = z;
    z = _null;
    z = yi*_next;
    yi = _null;
    yi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-copy-neq-err-lsum-prd-01/
		null pointer dereference at line z = yi*_next
LSUM-REL	Anon=(0,1)	log/intlist-copy-neq-err-lsum-rel-01/
		null pointer dereference at line z = yi*_next

3.3.4 Add some constant

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void add2(intlist x) {
  intlist xi = x;
  while (xi != NULL) {
    xi->data = xi->data + 2;
    xi = xi->next;
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real,
    _l: int, _k: int, S: int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xi = x;
  while xi != _null do
    xi = (xi * _data + 2)/ _data;
    y = xi * _next;
    xi = _null;
    xi = y;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-add2-lsum-prd-01/
		$x(n1)$ *and* $S[n1] >= S + 2$
LSUM-REL	Anon=(0,1)	log/intlist-add2-lsum-rel-01/
		$x(n1)$ *and* $S[n1] = S + 2 * l[n1]$
MSET	Anon=(0,1)	log/intlist-add2-mset-rel-01/
		none
UCONS	Anon=(0,1),P11	log/intlist-add2-uconspoly-P11-01/
		$x(n1)$

3.3.5 Copy a list and add some constant (1)

This program copy the data of a list into another list by adding a constant. The two lists have the same length.

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]==_l and data(y) and
 * disjoint(x,y) */
void add2copy_eq(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL) {
    yi->data = xi->data + 2;
    xi = xi->next;
    yi = yi->next;
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k: int, S: int;
begin
  assume (x == 2);
  xi = _null; yi = _null; z = _null;
  xi = x;
  yi = y;
  while xi != _null do
    yi = (xi * _data + 2)/ _data;
    z = xi* _next;
    xi = _null;
    xi = z;
    z = _null;
    z = yi * _next;
    yi = _null;
    yi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-add2copy-eq-lsum-prd-01/
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1] = l[n2] \ \wedge \ d(n1) + 2 = d(n2) \ \wedge \ S(n1) <= S(n2)$

LSUM-REL	Anon=(0,1)	log/intlist-add2copy-eq-lsum-rel-01/
		$x(n1) \wedge xi(n2) \wedge y(n3) \wedge yi(n4) \wedge l[n1] = l[n3] \wedge l[n2] = l[n4] \wedge d(n1){+}2 = d(n3) \ \wedge\ S(n1) + 2l(n1) = S(n3) + 2$
MSET	Anon=(0,1)	log/intlist-add2copy-eq-mset-rel-01/
		none
UCONS	Anon=(0,2),P21	log/intlist-add2copy-eq-uconspoly-P21-02/
		$x(n1) \ \wedge\ y(n2) \ \wedge\ \forall\ y1 \in\ n1,\ y2 \in\ n2\ y1 = y2 \ \Rightarrow\ d(y2) = d(y1) + 2$

3.3.6 Copy a list and add some constant (2)

This program copy the data of a list into another list by adding a constant. The two lists have different lengths, but the program correctly tests this case.

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) and
 * acyclic(y) and l[y]+1<=_l and data(y) and
 * disjoint(x,y) */
void add2copy_neq(intlist x, intlist y) {
  intlist xi = x;
  intlist yi = y;
  while (xi != NULL && yi != NULL) {
    yi->data = xi->data + 2;
    xi = xi->next;
    yi = yi->next;
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z1:real, z2:real,
    _l:int, _k: int, S: int;
begin
  assume (x == 3);
  xi = _null; yi = _null; z1 = _null; z2 = _null;
  xi = x;
  yi = y;
  while xi != _null and yi != _null do
    yi = (xi * _data + 2)/ _data;
    z1 = xi* _next;
    z2 = yi* _next;
    xi = _null; yi = _null;
    xi = z1; yi = z2;
    z1 = _null; z2 = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-add2copy-neq-lsum-prd-01/
		$x(n1) \ \wedge\ y(n2) \ \wedge\ l[n1] = l[n2] \ \wedge\ d(n1) + 2 = d(n2) \ \wedge\ S(n1) \geq\ S(n2)$
LSUM-REL	Anon=(0,1)	log/intlist-add2copy-neq-lsum-rel-01/
		$x(n1) \ \wedge\ xi(n2) \ \wedge\ y(n3) \ \wedge\ yi(n4) \ \wedge\ l[n1] = l[n3] \ \wedge\ l[n4] \leq\ l[n2] - 1 \ \wedge\ d(n1) + 2 = d(n3) \ \wedge\ S(n1) + 2l(n1) = S(n3) + 2$
MSET	Anon=(0,1)	log/intlist-add2copy-neq-mset-rel-01/
		none

UCONS	Anon=(0,2),P21	log/intlist-add2copy-neq-uconspoly-P21-02/
		$x(n1) \ \wedge \ y(n2) \ \wedge \ \forall \ y1 \in \ n1, \ y2 \in \ n2 \ y1 = y2 \ \Rightarrow \ d(y2) = d(y1) + 2$

3.3.7 Set the flag

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void setFlag(intlist x) {
  intlist xi = x;
  while (xi != NULL) {
    if (!xi->data) {
      xi->data = 1;
    }
    xi = xi->next;
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, z:real,
    _l:int, _k:int, S: int;
begin
 assume (x == 0);
 xi = _null; z = _null;
 xi = x;
 while xi != _null do
    if (xi* _data == 0) then
      xi = 1 / _data;
    endif;
    z = xi *_next;
    xi = _null;
    xi = z;
    z = _null;
 done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-setFlag-lsum-prd-01/
		$x(n1) \ \wedge \ S(n1) + d(n1) \ \geq \ S$
LSUM-REL	Anon=(0,1)	log/intlist-setFlag-lsum-rel-01/
		$x(n1) \ \wedge \ S(n1) + d(n1) \ \geq \ S \ \wedge \ S(n1) + d(n1) \ \leq \ S + l[n1]$
MSET	Anon=(0,1)	log/intlist-setFlag-mset-rel-01/
		none
UCONS	Anon=(0,1),P11	log/intlist-setFlag-uconspoly-P11-01/
		$x(n1) \ \wedge \ \forall \ y1 \in \ n1 \ \Rightarrow \ d(y1)! = 0$

3.3.8 Insertion sort array

This version of the insertion sort algorithm does not move cells of the list but only moves data between cells. Then, it simulates the insertion sort algorithm on arrays.

C code

Spl encoding

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void insertSortArr(intlist x) {
  intlist xi, y;
  int m, n;
  xi = y = NULL;
  xi = x->next;
  while (xi != NULL) {
    y = x;
    while (y != xi && y->data <= xi->data) {
      y = y->next;
    }
    m = xi->data;
    while (y != xi) {
      n = y->data;
      y->data = m;
      m = n;
      y = y->next;
    }
    xi->data = m;
    xi = xi->next;
  }
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real,
    _l:int, _k:int, S:int, m:int, n:int;
begin
  assume (x == 0);
  xi = _null;
  y = _null; yi = _null;
  xi = x * _next;
  while xi != _null do
    y = x;
    while y != xi and y * _data <= xi * _data do
      yi = y * _next;
      y = _null;
      y = yi;
      yi = _null;
    done;
    m = xi * _data;
    while y != xi do
      n = y * _data;
      y = m / _data;
      m = n;
      yi = y * _next;
      y = _null;
      y = yi;
      yi = _null;
    done;
    y = _null;
    xi = m / _data;
    yi = xi * _next;
    xi = _null;
    xi = yi;
    yi = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-insertSortArr-lsum-prd-01/
		$x(n1) \;\wedge\; S(n1) + d(n1) = S \;\wedge\; l[n1] = l$
LSUM-REL	Anon=(0,1)	log/intlist-insertSortArr-lsum-rel-01/
		$x(n1) \;\wedge\; S(n1) + d(n1) = S \;\wedge\; l[n1] = l$
MSET	Anon=(0,1)	log/intlist-insertSortArr-mset-rel-01/
		$x(n1) \;\wedge\; M[n1] = M \;\wedge\; l[n1] = l$
UCONS	Anon=(0,1),P11	log/intlist-insertSortArr-uconspoly-P11-01/
		$x(n1) \;\wedge\; \forall\, y1 \in\; n1 \;\Rightarrow\; d(n1) \;\leq\; d(y1)$

3.3.9 Bubble sort array

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void bubbleSortArr(intlist x) {
  intlist xi, xin;
  int v;
  int k = 1;
  while (k==1) {
    k = 0;
    xi = x;
    xin = x->next;
    while (xi != NULL && xin != NULL) {
      if (xi->data >= xin->data+1) {
        v = xi->data;
         xi->data = xin->data;
        xin->data = v;
        k = 1;
      }
      xi = xin;
      xin = xin->next;
    }
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, xin:real,
    _l:int, _k:int, S:int, v:int;
begin
  assume (x == 0);
  xi = _null; xin = _null;
  _k = 1;
  while _k==1 do
    _k = 0;
    xi = x;
    xin = x * _next;
    while xi != _null and xin != _null do
      if (xi * _data >= xin * _data +1) then
        v = xi * _data;
        xi = (xin * _data) / _data;
        xin = v / _data;
        _k = 1;
      endif;
      xi = _null;
      xi = xin;
      xin = _null;
      xin = xi * _next;
    done;
    xi = _null;
    xin = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-bubbleSortArr-lsum-prd-01 $x(n1) \wedge xi(n2) \wedge xin(n3) \wedge l = l[n1] + l[n3] + 1 \wedge l[n2] = 1 \wedge S = S[n1] + S[n2] + S[n3]$
LSUM-REL	Anon=(0,1)	log/intlist-bubbleSortArr-lsum-rel-01 $x(n1) \wedge xi(n2) \wedge xin(n3) \wedge l = l[n1] + l[n3] + 1 \wedge l[n2] = 1 \wedge S = S[n1] + S[n2] + S[n3]$
MSET	Anon=(0,1)	log/intlist-bubbleSortArr-mset-rel-01 $x(n1) \wedge xi(n2) \wedge xin(n3) \wedge l = l[n1] + l[n3] + 1 \wedge l[n2] = 1 \wedge M = M[n1] + M[n2] + M[n3]$
UCONS	Anon=(2,1),P21	log/intlist-bubbleSortArr-uconspoly-p21-21 $x(n1) \wedge xi(n2) \wedge xin(n3) \wedge l = l[n1] + l[n3] + 1 \wedge l[n2] = 1 \wedge \forall\, y1, y2 \in n3,\ y1 < y2 \Rightarrow d(y1) \leq d(y2)$

3.4 Changing structure

The examples in this class create, destroy, or change the position of cells in the list.

3.4.1 New copy of a list

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x)  */
intlist listCopy(intlist x) {
  intlist xi = x;
  intlist y = NULL;
  intlist yi = NULL;;
  intlist z = NULL;
  while (xi != NULL) {
    z = new();
    z->data = xi->data;
    z->next = NULL;
    if (y == NULL)
      y = z;
    else
      yi->next = z;
    yi = z;
    xi = xi->next;
  }
  return y;
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 0);
  xi = _null; z = _null;
  yi = _null; y = _null;
  xi = x;
  while xi != _null do
    z = _new;
    z = (xi* _data)/ _data;
    z = (_null)/_next;
    if (y == _null) then
      y = z;
    else
      yi = z / _next;
    endif;
    yi = _null;
    yi = z;
    z = _null;
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-newCopy-lsum-prd-01 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l = l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n1) = d(n2) \wedge S(n3) = 0 \wedge S[n1] = S[n2] + d(n3)$
LSUM-REL	Anon=(0,1)	log/intlist-newCopy-lsum-rel-01 same as above
MSET	Anon=(0,1)	log/intlist-newCopy-mset-rel-01 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l = l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n1) = d(n2) \wedge M[n1] = M[n2] + d(n3)$
UCONS	Anon=(0,2),P21	log/intlist-newCopy-uconspoly-P21-02 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l = l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n1) = d(n2) \wedge \forall\, y1 \in n1,\ y2 \in n2,\ y1 = y2 \Rightarrow d(y1) = d(y2)$

3.4.2 New copy and add

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
intlist add2new(intlist x) {
  intlist xi = x;
  intlist yi, y, z;
  yi = y = z = NULL;
  while (xi != NULL) {
    z = new();
    z->data = xi->data + 2;
    if (yi == NULL)
      y = z;
    else {
      yi->next = z;
      yi = NULL;
    }
    yi = z;
    z = NULL;
    xi = xi->next;
  }
  return y;
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int , _k: int, S:int;
begin
  assume (x == 0);
  y = _null;
  yi = _null;
  z = _null;
  xi = _null;
  xi = x;
  while xi != _null do
    z = _new;
    z = (xi * _data + 2) / _data;
    z = _null / _next;
    if (yi == _null) then
      y = z;
    else
      yi = _null/_next;
      yi = z/_next;
    endif;
    yi = _null;
    yi = z;
    z = _null;
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-add2new-lsum-prd-01 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n2) = d(n1)+2 \wedge S(n3) = 0 \wedge S[n2]+d(n3) \geq S+2 \wedge S = S[n1]$
LSUM-REL	Anon=(0,1)	log/intlist-add2new-lsum-rel-01 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l = l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n2) = d(n1)+2 \wedge S(n3) = 0 \wedge S[n2]+d(n3)+2 = S+2l \wedge S[n1] = S$
MSET	Anon=(0,1)	log/intlist-add2new-mset-rel-01 none
UCONS	Anon=(0,2),P21	log/intlist-add2new-uconspoly-P21-02 $x(n1) \wedge y(n2) \wedge yi(n3) \wedge l = l[n1] = l[n2]+1 \wedge l[n3] = 1 \wedge d(n1) = d(n2) \wedge \forall\, y1 \in n1,\ y2 \in n2,\ y1 = y2 \Rightarrow d(y1) = d(y2)$

3.4.3 New copy on condition

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void copyAllGeV(intlist x, int v) {
  intlist z;
  intlist y = null;
  intlist xi = x;
  while (xi != NULL) {
    if (xi->data >= v) {
      z = new();
      z->data = xi->data;
      z->next = y;
      y = z;
    }
    xi = xi->next;
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, yi:real, z:real,
    _l:int, _k:int, S:int, v: int;
begin
  assume (x == 0);
  xi = _null; y = _null; yi = _null; z = _null;
  xi = x;
  while xi != _null do
    if (xi* _data >= v) then
      yi = _new;
      yi = (xi * _data) / _data;
      yi = y / _next;
      y = _null;
      y = yi;
      yi = _null;
    endif;
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-copyAllGeV-lsum-prd-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1] + 1 \wedge S(n2) = 0 \wedge d(n1) + 1 \leq v \wedge d(n2) + 1 \leq v$
LSUM-PRD	Anon=(0,1), v=5	log/intlist-copyAllGe5-lsum-prd-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1] + 1 \wedge S(n2) = 0 \wedge d(n1) + 1 \leq 5 \wedge d(n2) + 1 \leq 5 \wedge S \geq d(n1) + d(n2) + S(n1)$
LSUM-REL	Anon=(0,1)	log/intlist-copyAllGeV-lsum-rel-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1] + 1 \wedge S(n2) = 0 \wedge d(n1) + 1 \leq v \wedge d(n2) + 1 \leq v$
LSUM-REL	Anon=(0,1), v=5	log/intlist-copyAllGe5-lsum-rel-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1] + 1 \wedge S(n2) = 0 \wedge d(n1) + 1 \leq 5 \wedge d(n2) + 1 \leq 5 \wedge 4l[n1] \geq S(n1) + 4$
MSET	Anon=(0,1), v=5	log/intlist-copyAllGe5-mset-rel-01 none

UCONS	Anon=(0,1),P11	log/intlist-copyAllGeV-uconspoly-P11-01
		$x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge d(n1)+1 \leq v \wedge d(n2)+1 \leq v \wedge \forall y1 \in n1 \Rightarrow d(y1)+1 \leq v$
UCONS	Anon=(0,1),P11,v=5	log/intlist-copyAllGe5-uconspoly-P11-01
		$x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge d(n1) \leq 4 \wedge d(n2) \leq 4 \wedge \forall y1 \in n1 \Rightarrow d(y1) \leq 4$

3.4.4 Delete on condition

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
void delAllGeV(intlist x, int v) {
  intlist z;
  intlist y = null;
  intlist xi = x;
  while (xi != NULL) {
    if (xi->data >= v) {
      z = xi;
      xi = xi->next;
      free (z);
      if (y==NULL)
          x = xi;
      else
          y->next = xi;
    }
    else {
      y = xi;
      xi = xi->next;
    }
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, z:real,
    _l:int, _k:int, S:int, v: int;
begin
  assume (x == 0);
  y = _null; xi = _null; z = _null;
  xi = x;
  while xi != _null do
    if (xi* _data >= v) then
      z = xi;
      xi = _null;
      xi = z * _next;
      if (y == _null) then
        x = _null;
        x = xi;
      else
        y = _null / _next;
        y = xi / _next;
      endif;
      z = _free;
      z = _null;
    else
      y = _null;
      y = xi;
      z = xi * _next;
      xi = _null;
      xi = z;
      z = _null;
    endif;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-delAllGeV-lsum-prd-01
		$x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge S(n2) = 0 \wedge d(n1)+1 \leq v \wedge d(n2)+1 \leq v$

LSUM-PRD	Anon=(0,1), v=5	log/intlist-delAllGe5-lsum-prd-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge S(n2) = 0 \wedge d(n1) \leq 4 \wedge d(n2) \leq 4 \wedge S \geq d(n1)+d(n2)+S(n1)$
LSUM-REL	Anon=(0,1)	log/intlist-delAllGeV-lsum-rel-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge S(n2) = 0$ *and* $d(n1)+1 \leq v \wedge d(n2)+1 \leq v$
LSUM-REL	Anon=(0,1), v=5	log/intlist-delAllGe5-lsum-rel-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge S(n2) = 0 \wedge d(n1) \leq 4 \wedge d(n2) \leq 4 \wedge 4l[n1] >= S(n1)+4$
MSET	Anon=(0,1)	log/intlist-delAllGeV-mset-rel-01 none
UCONS	Anon=(0,1),P11	log/intlist-delAllGeV-uconspoly-P11-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge d(n1)+1 \leq v \wedge d(n2)+1 \leq v \wedge \forall y1 \in n1 \Rightarrow d(y1)+1 \leq v$
UCONS	Anon=(0,1),P11,v=5	log/intlist-delAllGe5-uconspoly-P11-01 $x(n1) \wedge xi(n2) \wedge l[n2] = 1 \wedge l \geq l[n1]+1 \wedge d(n1) \leq 4 \wedge d(n2) \leq 4 \wedge \forall y1 \in n1 \Rightarrow d(y1) \leq 4$

3.4.5 Insertion sort list

This version of the insertion sort algorithm changes position of cells.

C code **Spl encoding**

```c
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
intlist insertSortLst(intlist x) {
  intlist xi, y, yi, z, r;
  z = xi = yi = y = NULL;
  r = z = x;
  xi = x->next;
  while (xi != NULL) {
    yi = NULL;
    y = r;
    while (y != xi && y->data < xi->data) {
      yi = y;
      y = y->next;
    }
    if (yi == NULL) {
      z->next = xi->next;
      xi->next = r;
      r = xi;
    }
    else {
      z->next = xi->next;
      yi->next = xi;
      xi->next = y;
    }
    xi = NULL;
    xi = z->next;
  }
  return r;
}
```

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, xip:real, y:real, yp:real, z:real,
    _l:int, _k:int, S:int;
begin
  assume (x == 0);
  xi = _null; y = _null;
  xip = _null; yp = _null; z = _null;
  xip = x;
  xi = x * _next;
  while xi != _null do
    y = x;
    while y != xi and y * _data <= xi * _data do
      yp = _null;
      yp = y;
      z = y * _next;
      y = _null;
      y = z;
      z = _null;
    done;
    if y != xi then
      xip = _null / _next;
      z = xi * _next;
      xip = z / _next;
      z = _null;
      if yp == _null then
        xi = _null / _next;
        xi = x / _next;
        x = _null;
        x = xi;
      else
        yp = _null / _next;
        yp = xi / _next;
        xi = _null / _next;
        xi = y / _next;
        yp = _null;
      endif;
      y = _null;
      xi = _null;
    else
      xip = _null;
      xip = xi;
      yp = _null;
      y = _null;
      xi = _null;
    endif;
    xi = xip * _next;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(2,1)	log/intlist-insertSortLst-lsum-prd-21
		$x(n1) \ \wedge \ xi(n2) \ \wedge \ l = l[n1] + l[n2] \ \wedge \ S(n2) = 0 \ \wedge \ S[n1] + S[n2] = S$

LSUM-REL	Anon=(2,1)	log/intlist-insertSortLst-lsum-rel-21
		$x(n1) \wedge xi(n2) \wedge l = l[n1] + l[n2] \wedge S(n2) = 0 \wedge S[n1] + S[n2] = S$
MSET	Anon=(2,1)	log/intlist-insertSortLst-mset-rel-21
		$x(n1) \wedge xi(n2) \wedge l = l[n1] + l[n2] \wedge M[n1] + M[n2] = M$
UCONS	Anon=(1,1), P11	log/intlist-insertSortLst-uconspoly-P11-11
		$x(n1) \wedge xi(n2) \wedge l = l[n1] + l[n2] \wedge \forall y1 \in n1 \Rightarrow d(n1) \leq d(y1)$

3.4.6 Dispatch lists

C code

```
#include "intlist.h"

/* acyclic(x) and _l==l[x] and data(x) */
void dispatch(intlist x,
              intlist xgtv,
              intlist xlev,
              int v) {
  intlist xi = x;
  intlist y;
  xgtv=NULL;  xlev=NULL;
  while (xi != NULL) {
    y=xi;
    xi=xi->next;
    if (y->data<=v) {
      y->next = xlev;
      xlev = y;
    }else {
      y->next = xgtv;
      xgtv = y;
    }
  }
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xgtv:real, xi:real, xlev:real, y:real, z:real,
    _l:int, _k:int, S: int, v:int;
begin
  assume (x == 0);
  xgtv = _null; xi = _null; xlev = _null; y = _null; z = _nul
  xi = x;
  x = _null;
  while xi != _null do
    y = xi;
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
    y = _null/_next;
    if (y * _data <= v) then
      y = xlev / _next;
      z = xlev;
      xlev = _null;
      xlev = y;
    else
      y = xgtv / _next;
      z = xgtv;
      xgtv = _null;
      xgtv = y;
    endif;
    z = _null;
    y = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,1)	log/intlist-dispatch-lsum-prd-01
		$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1] + l[n2] \wedge l[n1] \geq 1 \wedge l[n2] \geq 1 \wedge S = S[n1] + S[n2] \wedge v \geq d(n2) \wedge v + 1 \leq d(n1)$

LSUM-REL Anon=(0,1) log/intlist-dispatch-lsum-rel-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \; and \; l[n2] \geq 1 \wedge S = S[n1] + S[n2] \wedge v \geq d(n2) \wedge v+1 \leq d(n1)$

LSUM-PRD Anon=(0,1), v=5 log/intlist-dispatch5-lsum-prd-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \wedge l[n2] \geq 1 \wedge S = S[n1] + S[n2] \wedge 5 \geq d(n2) \wedge 6 \leq d(n1)$

LSUM-REL Anon=(0,1), v=5 log/intlist-dispatch5-lsum-rel-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \; and \; l[n2] \geq 1 \wedge S = S[n1]+S[n2] \wedge 5 \geq d(n2) \wedge 6 \leq d(n1) \wedge S[n1] \geq 6l[n1] \; and \; S[n2] \leq 5l[n2]$

MSET Anon=(0,1) log/intlist-dispatch-mset-rel-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \; and \; l[n2] \geq 1 \wedge M = M[n1] + M[n2] \wedge v \geq d(n2) \wedge v+1 \leq d(n1)$

UCONS Anon=(0,1),P11 log/intlist-dispatch-mset-P11-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \; and \; l[n2] \geq 1 \wedge v \geq d(n2) \wedge v+1 \leq d(n1) \wedge \forall y1 \in n1 \Rightarrow d(y1) \geq v+1 \wedge \forall y1 \in n2 \Rightarrow d(y1) \leq v$

UCONS Anon=(0,1),P11,v=5 log/intlist-dispatch5-uconspoly-P11-01

$x(null) \wedge y(n1) \wedge z(n2) \wedge l = l[n1]+l[n2] \wedge l[n1] \geq 1 \; and \; l[n2] \geq 1 \wedge v \geq d(n2) \wedge 6 \leq d(n1)$

3.4.7 Copy and reverse

C code

```
#include "intlist.h"

/* acyclic(x) and l[x]==_l and data(x) */
intlist copyRevList(intlist x) {
  intlist xi = x;
  intlist y, z = NULL;
  while (xi != NULL) {
    z = new();
    z->data = xi->data;
    z->next = y;
    y = z;
    xi = xi->next;
  }
  return y;
}
```

Spl encoding

```
var _data:real, _free:real, _len:real,
    _new:real, _next:real, _null:real,
    x:real, xi:real, y:real, z:real,
    _l:int, _k:int, S: int;
begin
  assume (x == 0);
  xi = _null; y = _null; z = _null;
  xi = x;
  while xi != _null do
    z = _new;
    z = (xi * _data) / _data;
    z = y / _next ;
    y = _null;
    y = z;
    z = _null;
    z = xi * _next;
    xi = _null;
    xi = z;
    z = _null;
  done;
end
```

Results

Domain	Param.	Log file / Interesting constraint
LSUM-PRD	Anon=(0,2)	log/intlist-copyRev-lsum-prd-02
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1] = l = l[n2] \geq \ 1 \ \wedge \ S = S[n1] = S[n2]$
LSUM-REL	Anon=(0,2)	log/intlist-copyRev-lsum-rel-02
		same as above
MSET	Anon=(0,2)	log/intlist-copyRev-mset-prd-02
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1] = l = l[n2] \geq \ 1 \ \wedge \ M = M[n1] = M[n2]$
UCONS	Anon=(0,2),P11	log/intlist-copyRev-uconspoly-P11-02
		$x(n1) \ \wedge \ y(n2) \ \wedge \ l[n1] = l = l[n2] \geq \ 1$