UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

SERIE “A”

TRABAJOS DE INFORMÁTICA

Nº 4/2011

A Suite of Tools for Analyzing ACL2 Books

Gabriel Infante Lopez - Alejandro E. Orbe - J Strother Moore

Editores: Luciana Benotti – Laura Brandan Briones

CIUDAD UNIVERSITARIA – 5000 CÓRDOBA

REPÚBLICA ARGENTINA

A Suite of Tools for Analyzing ACL2 Books

Gabriel Infante-Lopez[1,2], Alejandro Ezequiel Orbe[1], and J Strother Moore[3]

[1] Grupo de Procesamiento de Lenguaje Natural
Universidad Nacional de Córdoba, Argentina
[2] Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina
[3] Department of Computer Science
University of Texas at Austin

Abstract. We present a tool that inspects and analyzes ACL2 books. The tool provides useful information that might help the user to improve or to optimize a book. For any event e in a book, the tool looks for all the subsets of events in the book that make e admissible by ACL2. All sets that are found have the particular property that no subset with one fewer element makes e admissible. We show that our algorithm is exponential in the number of sets that have this property. We also show that it is correct and we prove that if the events in the book behave monotonically, the algorithm also finds all sets and that those sets are in fact the smallest sets that make an event admissible. We also describe some uses this information might have; in particular we show that there are books in the ACL2 standard distribution that can have 40% of their local lemmas eliminated.

1 Introduction

In this paper we present a tool, called **acl2-book-tools**, designed to inspect and to analyze ACL2 books [1], that can help an ACL2 developer to get a new perspective on the book he or she develops. So far, the tool contains two different scripts, BOOK_INSPECTOR and BOOK_ANALYZER, that are used for different purposes. The main objective of BOOK_INSPECTOR is to "inspect" a book and to establish dependencies among the events in it. This script inspects a given book looking for one particular type of information: it tries to find, for every event e in the book, all the sets made of events occurring before e that can make e admissible by ACL2 and that have the particular property that no proper subset with one fewer element makes e admissible.By "admissible" here we mean ACL2 will discover the requisite proofs from the events in the subset.

BOOK_INSPECTOR builds and explores the lattice (see [2] for a introduction to lattices and order) of subsets of events in a book. More precisely, if a target event t_e is an event in a given book, and $e_0, \ldots, e_k$ are the events occurring before t_e, the tool inspects the lattice of $\mathcal{P}(e_0, \ldots, e_k)$ looking for sets $M = \{e_i : i \in \{0, \ldots, k\}\}$ such that t_e is admitted by ACL2 after all elements of M are added to a new ACL2 world as `skip-proof` events. Moreover, BOOK_INSPECTOR looks for sets M such that proper subsets of M containing one fewer element makes event t_e not admissible anymore. We call M a *1-irreducible set* of t_e in the given book.

There might be more than one 1-irreducible sets for any event t_e in any given book; it is the task of our algorithm to find as many as it can. In principle, the search space is exponentially big on the number of events that occur before the target event, but our algorithm is in fact exponential on the number of 1-irreducible set that exist in a book. Since ACL2 books usually have only a few 1-irreducible sets, the algorithm becomes feasible. The reduction of complexity is due to the use of the following hypothesis. If a set M of events makes an event admissible, then all supersets of M make it also admissible. Clearly, this property is not true in general in ACL2 given that, for instance, a new theorem t can break the admissibility of an event that was otherwise admissible. If a certified book contains a set of events that do not satisfy this hypothesis, then our tool might not find all 1-irreducible sets and consequently, the information gathered from the book might not be complete. That is, there might some 1-irreducible sets that the tool cannot find but, nevertheless, all sets that are returned by the tool are guaranteed to be 1-irreducible. In contrast, if the set of events in a book behaves monotonically, then our algorithm finds them all. The algorithm complexity is still a problem whenever a book contains too many 1-irreducible sets for a given event. We have implemented some heuristics that are especially suitable for analyzing books containing 1-irreducible sets that share many elements in common. The more similar a 1-irreducible set is to one that has already been found, the faster the algorithm finds it.

After the tool has found the 1-irreducible sets for all the events in a book, different types of information can be mined from them using BOOK_ANALYZER. In particular, BOOK_ANALYZER can detect local lemmas that are not needed for proving any non-local event. We experiment with a number of books that come with the ACL2 distribution. Our experiments shows that, over a sample of 160 books randomly taken, 20 (11.85%) books presented an average of 60% redundant local events. More surprisingly, there exists events like `defthm mod-two` in book `arithmetic-2/floor-mod/floor-mod.lisp` that has 18 1-irreducible sets, that is, there are 18 different combinations of events that make this event admissible. BOOK_ANALYZER can also detect events that play a central role in a book. These events are those that belong to the 1-irreducible sets of many other events. Since it might be the case that not all 1-irreducible sets were found, the gathered information is consistent but might be incomplete.

The tool handles almost all events that might occur in a book with the exception of `deftheory`, `in-theory` and `encapsulate` events. The tool does not compute 1-irreducible sets for these events but they are used as part of the search space for foregoing events.

The rest of the paper is organized as follows. Section 2 shows a use case, we use the tool to inspect the book `list-defthms` which can be found in the standard distribution of ACL2. Our tool computes all 1-irreducible sets for the 211 events contained in the book in about an hour and a half. Moreover, BOOK_ANALYZER shows that 24% of the local theorems are in fact redundant. Section 3 describes in detail the algorithm used by BOOK_INSPECTOR. Section 4 shows describes the algorithm for detecting local events that are redundant. Section 5 shows that the

algorithm finds all 1-irreducible sets if the events in a book behaves monotonically, and that it runs in an exponential time on the number of 1-irreducible sets. Section 6 shows experiments on some ACL2 standard books. Section 7 concludes the paper.

2 A Use Case

In this section we briefly explain how to use of BOOK_INSPECTOR to analyze `list-defthms.lisp` from the standard ACL2 distribution. After calling BOOK_INSPECTOR with `list-defthms.lisp` as argument, the script starts the setup phase, where the input file is loaded and parsed to build the XML representation of it. Also, the corresponding .acl2 file is also loaded if it exists. Figure 1 shows the tool output at this stage.

```
... - INFO - Parsing command line options...
... - INFO - Starting the setting up of the environment ...
... - INFO - Creating results directory at: /home/ ...
... - INFO - Checking the existence of file: /.../list-defthms.acl2 .
... - INFO - File not found,  moving on...
... - INFO - Loading events from file: /.../list-defthms.lisp
... - INFO - 211 events were found...
...
... - INFO - Setting up finished successfully ...
... - INFO - Starting inspection of all found events ...
...
```

Fig. 1. A fraction of the output during the setup phase.

One at a time, each one of the events in the book is taken as *target event* and it is fed, with all its preceding events, to the exploration algorithm. The algorithm starts by testing if the target event is admissible in the initial theory of ACL2. If it is, then the event is skipped and the algorithm continues to deal with the next event in the file. This is the case for the 210-th event of `list-defthms.lisp`. Figure 2 shows the tool output for this event.

```
... - INFO - Skipping in-theory event...
... - INFO - ====================  Inspecting Event  ====================
... - INFO - Target id: 210 ...
... - INFO - Target definition: (defthm update-nth-update-nth ...
... - INFO - Target event type: defthm ...
... - INFO - Number of events in the initial set: 209
... - INFO - The event its admissible in the ACL2's initial theory...
... - INFO - 1-irreducible sets for the event: []
```

Fig. 2. Event 210 is skipped because it is accepted in ACL2's initial theory.

If the target event is not admissible in ACL2's initial theory, then the algorithm tries to admit it in the ACL2 theory that results from extending the ACL2's initial theory with all the events preceding the target event. In order to do so, the algorithm encloses each of them in a *skip-proof* form. All skipped event conform what it is called the *initial set* of the target event. If the target event is admitted in this way, the exploration starts. Figure 3 illustrate this situation for event 201 in the book `list-defthms.lisp`. In this case, the algorithm

explores the lattice of subsets that can be built using the preceding 200 events. For this particular case, the exploration algorithm finds two 1-irreducible sets.

```
... - INFO - ====================  Inspecting Event  ====================
... - INFO - Target id: 201 ...
... - INFO - Target definition: (defthm put-nth-firstn ...
... - INFO - Target event type: defthm ...
... - INFO - Number of events in the initial set: 200
... - INFO - The event is admissible...Looking for 1-irreducible sets...
... - INFO - A 1-irreducible set was founded: [2, 31, 200] ...
... - INFO - A 1-irreducible set was founded: [2, 105, 200] ...
... - INFO - 1-irreducible sets for the event: [(3, [2, 31, 200]), (3, [2, 105, 200])]
```

Fig. 3. The exploration algorithm has found two 1-irreducible sets for event 201.

Once the tool has inspected all the events in a book, it outputs an XML and a DOT file. The XML file contains all the information that was gathered during the exploration. It contains an entry for each event in the book specifying the 1-irreducible sets that were found for them. Figure 4 shows a small fraction of the XML file that results from exploring `list-defthms.lisp`. It states that there are two 1-irreducible sets that can prove event 201, that the tool took 1 hour 25 minutes for computing all the 1-irreducible sets for the 211 events in the book and that it took 2 minutes 37 seconds to find the 1-irreducible sets for event 201.

```
<book-inspection exploration-time="1:25:52.00" ...>
  <book filename="/home/.../list-defthms.lisp"/>
  ...
  <events>
    <event book-id="1" id="201" local="False" type="defthm">
      <definition>
        (defthm put-nth-firstn ...
      </definition>
      <irreducible-sets calls-to-prover="112" exploration-time="0:02:23.768467" ...>
        <irreducible-set calls-to-prover="53" ... set="[2, 31, 200]" .../>
        <irreducible-set calls-to-prover="51" ... set="[2, 105, 200]" .../>
      </irreducible-sets>
    </event>
  ...
  <events>
</book-inspection>
```

Fig. 4. Fraction of an XML file showing information about event 201 of `list-defthms.lisp`

The DOT file contains an *event dependency graph* (see Fig. 5) that is built using the information contained in the XML file. Conceptually, an event dependency graph is a directed acyclic graph that contains a node for each event in the book and arc from event i to event j if event i occurs in a 1-irreducible set of event j. In this graph the shadowed octagon node corresponds to a `local` event, the ellipse to a non- `local` theorem, and the box to every other event. Also, every node has a label of the form "x/y/z" where x is an id of a book where the event occurs, y is the event id and z is the event type. Books id are used because our tool accepts more than one book as argument allowing the user to test how the events in one book interact with the events in another book.

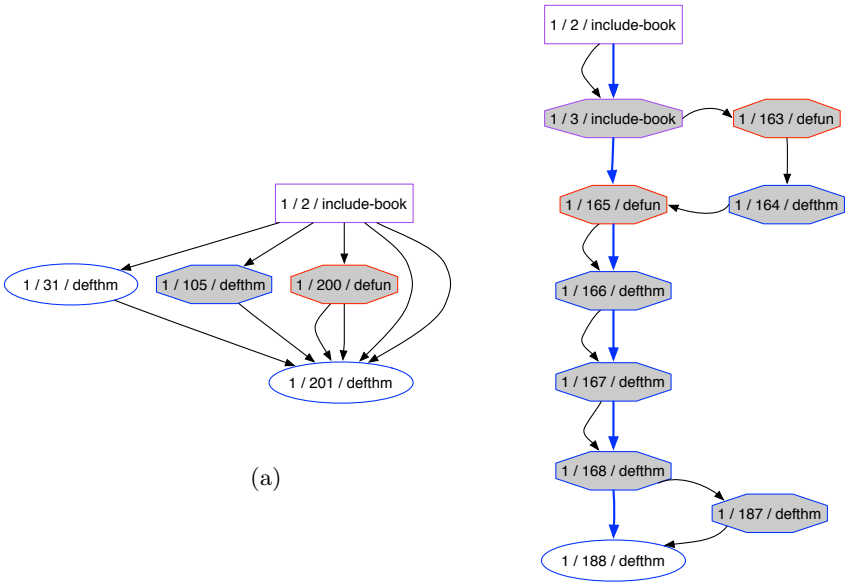


Fig. 5. (a) A sub-graph of the event dependency graph corresponding to book `list-defthms.lisp`, (b) A sub-graph of the event dependency graph corresponding to book `list-defthms.lisp`.

Figure 5 (a) shows a sub-graph, centered on the event 201, of the event dependency graph corresponding to the `list-defthms.lisp` book. With the information contained in the XML file, BOOK_ANALYZER can determine all the available proofs that can be built for a given event and find the shortest one. Figure 5 (b) shows two different proof for event 188. Different proofs are built using the information for all 1-irreducible sets. For example, the graph in Fig. 5 (b) states that event 1/2 does not have any 1-irreducible set, i.e., it is accepted in ACL2 initial world. Event 1/3 has one 1-irreducible set containing solely event 1/2, event 1/165 has two different 1-irreducible sets, one containing 1/164 and the other one containing event 1/3. Clearly, event 1/165 can be proved without using events 1/164 and 1/163. Our tool can detect the existence of *redundant events* that can be deleted from the book. A redundant event is an event that can be deleted from the book without modifying the admissibility of other events. Our tool uses a rather shallow analysis to detect redundant events that does not guarantee that all of them are found. The algorithm for finding redundant lemmas is explained in more detail in Sect. 4.

Figure 6 shows the output of BOOK_ANALYZER for `list-defthms.lisp`. The output indicates that are at least 13 local theorems that could be removed from the book without risk, moreover, it states that event 2 is the most important event in the book. A more detailed report is saved in an XML file, which in-

cludes information about the alternative proofs for a given event, successors and predecessors count, degree and degree centrality among others.

```
... - INFO - ============== SUMMARY ===============
... - INFO - Event Type: in-package - Count: 1 - Local Count: 0 -  Redundants Count: 0
... - INFO - Event Type: defthm - Count: 201 - Local Count: 50 -  Redundants Count: 13
... - INFO - Event Type: in-theory - Count: 3 - Local Count: 2 -  Redundants Count: 2
... - INFO - Event Type: defun - Count: 4 - Local Count: 4 -  Redundants Count: 0
... - INFO - Event Type: include-book - Count: 2 - Local Count: 1 -  Redundants Count: 0
... - INFO - Central Event: Id: 2 - Type: include-book - Successors Count: 82 - ...
... - INFO - ============== END SUMMARY ===============
... - INFO - Complete statistics are available at:/.../list-defthms-stats.xml
```

Fig. 6. Our tool output for the analysis of the dependency graph.

3 Exploration Algorithm

This section presents a detailed description of the algorithm that finds the 1-irreducible sets for a given event t_e and it also introduces some concepts that are required later.

The BOOK_INSPECTOR algorithm takes as input a *target event* t_e and an *initial set* $I = \{e_0, \ldots, e_k\}$ of events, all of them occurring before t_e in the book file. The goal of the algorithm is to find all the sets $M = \{e_i : i \in \{0, \ldots, k\}\}$ such that t_e is admitted by ACL2 after extending ACL2's initial theory with all events in M but no subset of M with one fewer element makes t_e admissible. Those sets are called the *1-irreducible sets* of the target event t_e.

To achieve its goal, the algorithm explores the lattice $\mathcal{P}(I)$ containing one node for each subset of I. I_i identifies an element in the lattice, it denotes the i-th subset of a family of subsets $\mathcal{F}_I$, containing 1 fewer element than the set I. The ordering among the elements of $\mathcal{F}_I$ is not really important and can be any.

In order to test if a set I makes the target theorem t_e admissible, every event in I is submitted to ACL2 embedded in a `skip-proofs` form, thus forcing ACL2 to omit its proofs obligations. Once all events have been introduced to the ACL2's initial theory, the algorithm submits the target event t_e to ACL2 and checks if t_e is or is not admitted. The subset I is called a *positive set* for t_e if ACL2 admits the target event, and is called *negative set* otherwise. It is worth mentioning that in this context, our notion of the *admissibility* of a target event, like ACL2's, requires that certain proof obligations be dispatched by the prover. For example, a definition event requires proofs of the termination conjectures (and the guard conjectures if guards are being verified). A `defthm` event requires proof of the stated theorem.

If set I behaves monotonically, that is, any superset of a positive set is also positive, then the 1-irreducible sets are in fact the smallest sets that make an event admissible. To illustrate this point, suppose that I is a set such that any superset of a positive set is also positive. Let M be a 1-irreducible set and let M' be a proper subset of M that is also positive. If M' is not M, then M can not be a 1-irreducible set given that there has to be a subset of M containing one fewer

element that is a superset of M' contradicting the definition of 1-irreducible sets. Nevertheless, since ACL2 books might not satisfy this property, it might be the case that a set M is a 1-irreducible set and yet it is not the smallest set that makes an event admissible.

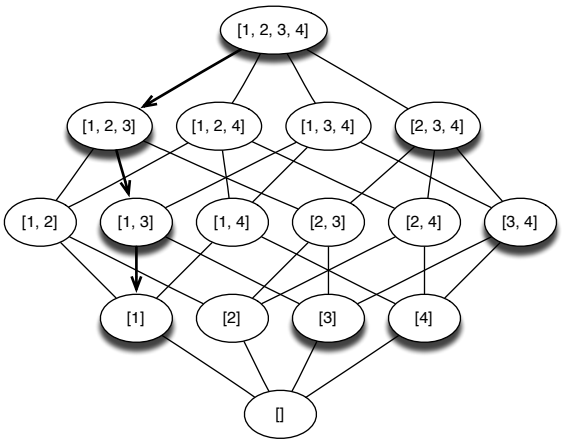


Fig. 7. The lattice to be explored for an initial set containing 4 events.

In fact, BOOK_INSPECTOR can be thought of as containing two different phases. One that starts in a positive set and looks for the set below the starting one that is a 1-irreducible set. The algorithm enters its second phase every time the first phase produces a new 1-irreducible set. The second phase is in charge of looking a new positive set in the lattice that does not contain any of the 1-irreducible sets that were already found. The first phase starts with an initial set I and a target event t_{e}. The algorithm starts exploring the first family of subsets of I:

$$\mathcal{F}_{\mathrm{I}} = \{I_i : I_i \subset I \wedge |I_i| = |I| - 1\} \ ,$$

where $|I|$ is the cardinality of the set I. Sets in $\mathcal{F}_{\mathrm{I}}$ are easily obtained from I by removing one element at a time. I is called the *generating set* of the family $\mathcal{F}_{\mathrm{I}}$. The exploration algorithm tests sets in $\mathcal{F}_{\mathrm{I}}$ until it finds a positive set. If none of these sets is positive, then the algorithm has found that I is a 1-irreducible set. If a positive set I_i is found, then a new family $\mathcal{F}_{\mathrm{I}_i}$ of sets is inspected. This process goes on recursively until it finds a set I that is a 1-irreducible set. The first phase of the algorithm guarantees that a 1-irreducible set is produced if this phase is started in a positive initial set.

In order to illustrate this phase Fig. 7 shows the lattice that the first phase has to explore. Shadowed nodes are positive, while others are negative. The first phase explores nodes $[1, 2, 3]$, $[1, 2]$, $[1]$ and $[]$ concluding that $[1]$ is a 1-irreducible set. Note that nodes $[2, 3, 4]$, $[3, 4]$, $[3]$ and $[4]$ are positive, but $[2, 4]$ and $[2, 3]$ are not. This is the case because $[1, 2, 3, 4]$ does not behave monotonically, and

event 2 interferes with events 3 and 4. Once the 1-irreducible set [1] has been found, the algorithm has to start its search over.

Relocalization A key step during the lattice exploration process is to determine the next subset that must be explored by the algorithm once a 1-irreducible set has been found. In particular the algorithm must avoid all the supersets of the 1-irreducible sets that have been already found because the first phase of algorithm will deterministically converge to those entering in a endless cycle.

The exploration algorithm implements a *relocalization* process every time a new 1-irreducible set is found. This process consists of finding a subset of the initial set I that does not contain any of the 1-irreducible sets that have been already found. A relocalization set is generated by removing elements from I that are in some 1-irreducible sets. Specifically, a relocalization set is built by first building a set containing one element from each 1-irreducible set, and second, removing this set from the original set I. Clearly, there are more than one relocalization set. The algorithm looks for the first relocalization set that makes the target event admissible; if it cannot find such a set, it terminates.

A more formal definition of this process can be provided by defining the family $\mathcal{R}$ of relocalization sets as:

$$\mathcal{R} = \{I - k_i : k_i \in \mathcal{K}\} \ , \tag{1}$$

where $\mathcal{K}$ is defined as:

$$\mathcal{K} = M_1 \times M_2 \times \ldots \times M_k \ ,$$

the Cartesian product of the k 1-irreducible sets found so far by the algorithm. This product is calculated every time a new 1-irreducible set is found thus assuring that the next relocalization set to explore does not lead to a known 1-irreducible set.

Let us illustrate how the relocalization works using the previous example. After 1-irreducible set [1] has been found, the relocalization process generates and tests set $[2, 3, 4]$, starts its first phase again, and finds 1-irreducible set [3]. Then the algorithm tries to relocalize again, testing $[2, 4]$, but since $[2, 4]$ is negative, the algorithm terminates without finding the 1-irreducible set [4]. Had $[1, 2, 3, 4]$ been monotonic, then $[2, 4]$ would have had to be positive.

4 Finding Redundant Local Events

The algorithm in charge of finding redundant local events builds a directed labeled graph with the information produced by BOOK_INSPECTOR for a given book. These graphs contain a node for each event in a book and an arc from node i to node j labeled with a number n if the n-proof for event j requires event i to be admitted before. Nodes that do not have incoming edges are called *starting nodes* and correspond to events that are admissible in ACL2's initial theory. A *proof path* is a path that starts at a starting node and has all of its

arcs labeled with the same number. The algorithm process the graph one node at the time following backwards the order of events in the given book. For each node, the algorithm tries to find a proof path ending in the current node that does not include any local event . Proof paths that meet this requirement are called *local-free paths*. If there is more than one local-free path for the current node, the algorithm picks the shortest one. If there is no local-free path, then the algorithm picks the shortest proof path that exists. Depending on whether the node being analyzed is a non-local or local event, local nodes in the path proof are marked as *required*, or *maybe-required* respectively. Once a proof path has been selected for each event, the algorithm checks the marks of all local nodes in the graph. Nodes that are neither required nor maybe-required correspond to events that the algorithm reports as redundant. Moreover, nodes that are marked as maybe-required are also reported as redundant if the local events that required them were previously marked as redundant. Our algorithm does not backtrack and consequently, there might better paths in the graph – that is, requiring fewer local events – than the ones that are found by our algorithm.

Let us illustrate this algorithm using a small example. Suppose that BOOK_INSPECTOR inspects a book with 5 events and that it returns the description shown in Fig. 8 (a). First the algorithm looks for a local-free path to node 5. In our example, such a path is the one starting at node 2 labeled with 3. Then it goes to node 4, and again path 3 is selected given that does not include any local node. For event 3, which is a local node, the local-free path labeled 2 is selected. The local node 1 is also a starting node so no path proof is searched. Once a path has been selected for each node the algorithm identifies events 1 and 3 as redundant given that they are not required by any non-local event. A more interesting situation arises when path 3 is not present in the graph. In this case, event 5 has three different paths, namely 1, 2 and 4, all of them containing local nodes. The algorithm chooses the shortest one containing the fewer local nodes and marks all of them as required. Since all paths have the same length, the algorithm selects path 1 given that it is the first one it finds. Consequently, local node 1 is marked as "required". Finally, the algorithm only identifies node 3 as redundant because there is no other node requiring it.

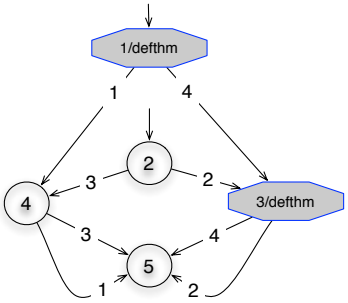
5 Properties of book_inspector

We start by showing that if the set of events in a book behaves monotonically, i.e., a superset of a positive set is also positive, the BOOK_INSPECTOR finds all 1-irreducible sets and moreover all found 1-irreducible sets are in fact the smallest sets that make the target event admissible.

Lemma 1. *If the target event is not admissible in ACL2's initial theory, the initial set is a positive set, and the initial set behaves monotonically, then the first phase of the* BOOK_INSPECTOR *algorithm always finds one 1-irreducible set and it also corresponds to one of the smallest sets that make the target event admissible.*

event number	1-irreducible sets
1	$\emptyset$
2	$\emptyset$
3	{2}, {1}
4	{1} , {2}
5	{3}, {4}

(a)



(b)

Fig. 8. (a) Small example of the possible for a book containing 5 events. The table shows the hypothetical 1-irreducible sets for each event. (b) The labeled graph that this built using the information in (b) specifically for event 5.

Proof. The proof is direct. The first phase goes down the lattice only when a positive set has been found. When the algorithm stops going down, it has found a set M such that every subset with exactly 1 less element is negative, i.e., M is 1-irreducible set. □

The previous lemma states that the first phase always produce a new 1-irreducible set. On the other hand, the second phase is in charge of looking for the right initial set to start the first phase over. In order to show that the algorithm finds all 1-irreducible sets, we need to show that there is always a relocalization set that contains an unexplored part of the lattice. The following lemma states precisely this result.

Lemma 2. *A set not containing a 1-irreducible set is either a relocalization set or a subset of a relocalization set.*

Proof. Let I be the initial set for some target event t_e, and let $M_1, \ldots, M_k$ be the k 1-irreducible sets found so far by the algorithm. Now take a set $S \subseteq I$ and suppose that it does not contains any 1-irreducible sets found by the algorithm. Then, S must differ in at least one element with each of the 1-irreducible sets found, that is,

$$S \subseteq I - \{m_{i1}, \ldots, m_{jk}\} \ , \tag{2}$$

where $m_{jk} \in M_k$. Nevertheless, the r.h.s of Equation 2 corresponds to the definition of relocalization set given by Equation 1, and therefore, $S \subseteq R_i$, where R_i is a relocalization set belonging to the family $\mathcal{R}$ of relocalization sets generated by the k-th 1-irreducible sets. □

Corollary 1. *If the initial sets behaves monotonically then* BOOK_INSPECTOR *finds all existing 1-irreducible sets.*

The idea behind a 1-irreducible set can be extended to k-irreducible, that is, a positive set is k-irreducible if all subsets having at most k fewer elements are negative. If the set behaves monotonically them there is no point of using k-irreducible because if a set is 1-irreducible set then it is k-irreducible for all possible k. But on the other hand, if the set does not behave monotonically there might be a set that is k-irreducible but that might have a subset containing $k+1$ fewer elements that is again positive. If the set does not behave monotonically, and one wants to find the smallest set that makes the target event admissible, then it has to traverse the whole lattice given that, unfortunately, a negative set does not state that all of its subsets are negative. The parameter k controls the order of the first phase of BOOK_INSPECTOR, the bigger the k the more inefficient is the algorithm. Fortunately for k equal to 1 this phase is still polynomial.

5.1 Order of the Algorithm

We mainly count the number of elements in the lattice that are inspected in each phase. Suppose that the size of the initial set is I and that there is one 1-irreducible set whose size is k. The worst case occurs when the 1-irreducible set occurs as the last element in all orderings of intermediate families. If in Fig. 7 families are explored from left to right , if $k = 2$, then the first phase tests $[1, 2, 3]$, $[1, 2, 4]$,$[1, 3, 4]$ as negative, $[2, 3, 4]$ as positive, $[2, 3]$, $[2, 4]$ as negative, $[3, 4]$ as positive, and finally $[3]$, $[4]$ as negative. In the worst case, the algorithm makes

$$\left(\sum_{i=1}^{I-k} I - i\right) + (I - k) + 1 + k$$

tests. $\sum_{i=1}^{I-k} I - i$ accounts for the number of negative test the algorithm has to make before it reaches the 1-irreducible set, $I - k + 1$ accounts for the positive paths that leads to the 1-irreducible set and k accounts for the negative tests below the 1-irreducible set. Then, the resulting number of tests is

$$\left(\sum_{i=1}^{I-k} I - i\right) + I + 1 < I^2 + I + 1 \ .$$

The second phase of the algorithm looks for new sets on which to start the first phase.The number of relocalization sets in the second phase is given by the number of minimals that have been already found. Let us suppose that there are m 1-irreducible sets. The worst case scenario occurs when the algorithm starts its second phase after having found all m 1-irreducible sets. At this point the algorithm starts its first phase again looking for a positive set. Clearly, since all 1-irreducible sets have been already found, all relocalization sets are negative and consequently, the algorithm has to explore all of them. The algorithm will end when it has explored all of them. In order to determine the complexity of the second phase, we count the number of relocalization sets that are generated after the first phase has found all the m minimals. If we suppose that each 1-irreducible

set contains at most k elements, then the number of relocalization sets is smaller than k^m. This means that the algorithm has to explore an exponential number of relocalization set on the number of 1-irreducible sets. Moreover, every time it finds a new one, the relocalization sets have to be recomputed. In practice, the second phase does not generate all relocalization sets; it does not generate the relocalization sets that are contained by other relocalization sets. Using this simple heuristic, and because 1-irreducible sets for ACL2 books tend to share a lot of their members, the number of relocalization sets diminishes dramatically. In order to hint why this is the case, suppose that A and B are two 1-irreducible sets. The number of relocalization sets that have to be checked is $|A| \times |B - A|$, while if only the upper most sets are generated, then the number is $|A \cap B| + |A - (A \cap B)| \times |B - (A \cap B)|$. If A and B have k elements each and they differ in one element, the number of relocalization sets is $k - 1 + 1 \times 1 = k$, while if all relocalization sets are generated, the number of relocalization sets becomes $k \times k$. But on the other hand, if two 1-irreducible sets A and B do not have any element in common, the number of relocalization sets is $A \times B$ and the worst case complexity is indeed exponential, but it turns out that it is still feasible for ACL2 books. The worst case complexity arises when the book is such that all sets with even cardinality are positive and all others are negative. Moreover, any algorithm trying to find all 1-irreducible sets for this pathological case has to traverse the whole lattice.

It easy to see that our algorithm does not find all 1-irreducible sets. For instance, if nodes $[1, 2, 3, 4]$, $[1, 2, 3]$, $[1, 3]$, $[1]$ and $[3]$ are the only positive nodes of Figure 7, our algorithm correctly finds $[1]$ but it does not find $[3]$, but any of the relocalization sets is again positive. Relocalization sets work as a random sample of nodes. As a consequence of Lemma 2, we know that they are a sample that covers the whole lattice, but we also know the sample is rather big.

6 Experiments

In this section we present the outcome of having executed our tool over a random sample of 160 books from the ACL2 standard distribution. All books in the sample were inspected using BOOK_INSPECTOR and the information it generated was analyzed using BOOK_ANALYZER. Experiments were executed on an Intel Core Duo T7100 PC with 4 GB of RAM memory using ACL2 version 3.4. Table 1 shows a small summary of all the information gathered by BOOK_INSPECTOR.

Table 1. Number of events and books that were processed by our tool.

Number of Books	160	Number of Events	4493
Number of Theorems	2766	Number of Functions	495
Number of Books w/ Theorems	127		

Our experiments show that only 19 books – representing a 11.85% of the total – contain local lemmas and that 60% of the local lemmas are redundant. More-

over, 6 books have *all* their local lemmas redundant. Table 2 shows a summary of the data that was collected for some of books in the sample.

Table 2. Columns shows the name, the number of events, the number of theorems, the number of local theorems, the number of redundant theorems, the sum of the numbers of 1-irreducible sets and the time it took to analyze the book respectively.

book	events	theorems	locals	redundant	1-irr	time
array1	69	54	38	20	39	19:04:37.87
list-defthms	211	201	50	13	143	01:25:52.00
alist-defuns	57	10	10	6	36	00:03:10.13
arithmetic-3/.../floor-mod	92	63	7	4	80	32:00:00.00
arithmetic-4/.../expt	33	26	3	3	53	00:10:00.10
more-floor-mod	28	18	3	3	23	00:17:16.04
simple-equ-and-inequ-helper	12	6	4	3	9	00:00:53.88
binomial	35	23	7	3	28	00:08:06.97
arithmetic-3/.../expt	33	26	3	3	53	00:12:34.69
alist-defthms	162	153	20	3	159	01:37:48.45
prefer-times	16	11	3	2	9	00:00:11.78
arithmetic-2/.../expt	31	25	2	2	50	00:10:52.97
building-blocks-helper	14	7	4	2	9	00:00:33.21
perm	12	7	7	2	11	00:01:14.50
collect-terms-meta	9	1	1	1	4	00:00:11.72
types-helper	10	8	5	1	5	00:00:10.66
memtree	65	32	3	1	51	02:42:09.87
integerp	94	82	3	1	95	01:19:20.17
cancel-terms-meta	65	6	1	1	34	12:00:00.00

The results state that there are local lemmas that might be eliminated but elimination of these local lemmas might not be as direct as just deleting the lemmas from the book. The information generated by BOOK_INSPECTOR not only states which local events can be eliminated but also states how to modify the theory of events that depend on it. We did test which books could have their redundant events removed: 10 out of 19 books that contain local lemmas can have all their redundant lemmas deleted without any further changes in the book. Some events in the remaining 9 requires that hints be added in order to make them admissible again.

This small analysis raises the question of why the authors of these book introduce redundant lemmas. The question is easily answered when the redundant lemmas cannot be directly deleted: the lemmas guide ACL2. It is often easier and produces a more robust script for an author to guide the prover with a local lemma than a goal-specific hint. But it is harder to understand the presence of irrelevant lemmas that can be directly eliminated. Perhaps ACL2 has been improved since the book was created so that local lemmas once needed to the guide the proof are no longer required.

Central events are those that belong to 1-irreducible sets of many other events in a book. The centrality of an event is computed from the same graph that is

Table 3. A few central events from the books that contain the higher number of events.

Book	# Events	Event	# Succ.
list-defthms	211	(include-book "list-defuns")	82
alist-defthms	162	(include-book "alist-defuns")	129
arithmetic-4/.../arithmetic-theory	110	(local (include-book "expt"))	32
arithmetic-3/.../arithmetic-theory	108	(local (include-book "basic"))	41
arithmetic-3/.../integerp	105	(encapsulate () (defthm not-integerp-helper ...) (defthm not-integerp-1a...))	64

used for finding redundant lemmas, the centrality of a node is the number of outgoing arcs it has. Table 3 shows some nodes with highest centrality values for the biggest 5 books. Usually the most required event is the `include-book` event which masks the real interaction between the events in the book being included and the events that need that book.

7 Conclusions

We have presented a set of tools to inspect and analyze ACL2 books. The algorithm we presented takes polynomial time on the size of the initial set to find a 1-irreducible set if the initial set is positive and it takes an exponential time on the number of 1-irreducible sets contained in a book to find a new positive starting set. We showed that if the set of events in a book behaves monotonically, the algorithm finds all 1-irreducible set and that those 1-irreducible sets are in fact the smallest sets that can make an event admissible. We showed empirically, that even though the algorithm is exponential, the nature of ACL2 books allows us to produce statistics for 160 books. A possible application of the information gathered by BOOK_INSPECTOR that we did not explore is to compare different versions of ACL2. With the information that our algorithm produces we can not only compare the time that different version of ACL2 take to accept events but we can also see how 1-irreducible sets have evolved with ACL2 version. It may be the case that older versions of ACL2 had fewer 1-irreducible sets or that they had fewer redundant lemmas.

References

1. Kaufmann, M., Manolios, P., Moore, J.: Computer– Aided Reasoning: An Approach. Kluwer (2000)
2. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press (2002)