Using the INKA Prover to Automate Safety Proofs in Abstract Interpretation – A Case Study

Dennis Dams[1], Dieter Hutter[2], and Natalia Sidorova[1]

[1] Dept. of Electrical Eng.,
Eindhoven University of Technology
{dennis, natalia}@ics.ele.tue.nl

[2] German Research Center
for Artificial Intelligence
hutter@dfki.de

CONTENTS

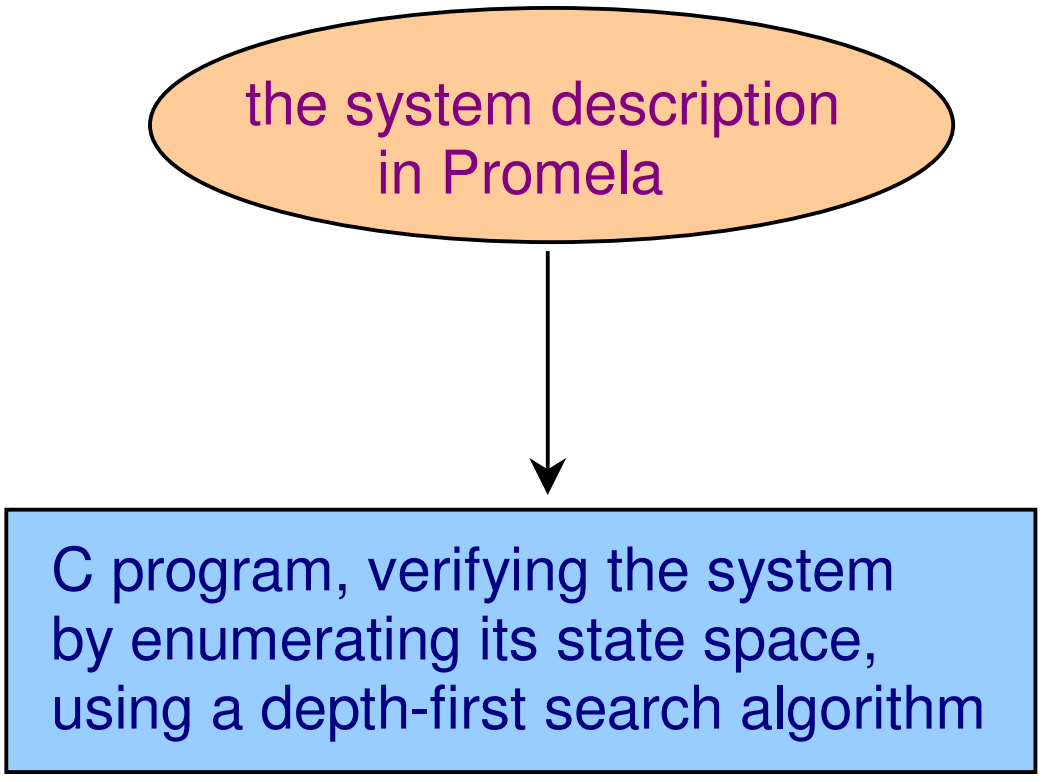
- Motivation
- Data abstractions
- A case study: Philips' Bounded Retransmission Protocol
- Abstracting lists
- Tuning-up INKA
- Conclusions and future work

The Spin model-checker

Spin: a software tool for analysis and verification of concurrent systems

Input language: Promela

a high-level language based on Dijkstra's guarded command language notation and borrowing the notation for i/o operations from Hoare's CSP language



Defining data types in Promela

- *bit, bool, byte, short, int*
- *arrays*
- symbolic constants

mtype = {*OK, READY, ACK*}

- user-defined data types (*typedef*)

Operations on data in Promela

- operations can be defined using simple function mechanism (*#define* …)

```
typedef List{
  byte nonempty;   /* Length of list. */
  byte lst[n];   /* The list. */
  byte i;   /* Auxiliary. */
};

#define Append(e,l) \
  l.i = l.nonempty; \
  do \
  :: l.i > 0 -> \
     l.lst[l.i] = l.lst[l.i - 1]; \
     l.i = l.i - 1 \
  :: else -> break \
  od; l.lst[l.i] = e; \
  l.nonempty = l.nonempty + 1
```

A Trade-off:

A limitation of model-checking:
its applicability to relatively small
finite-state systems only

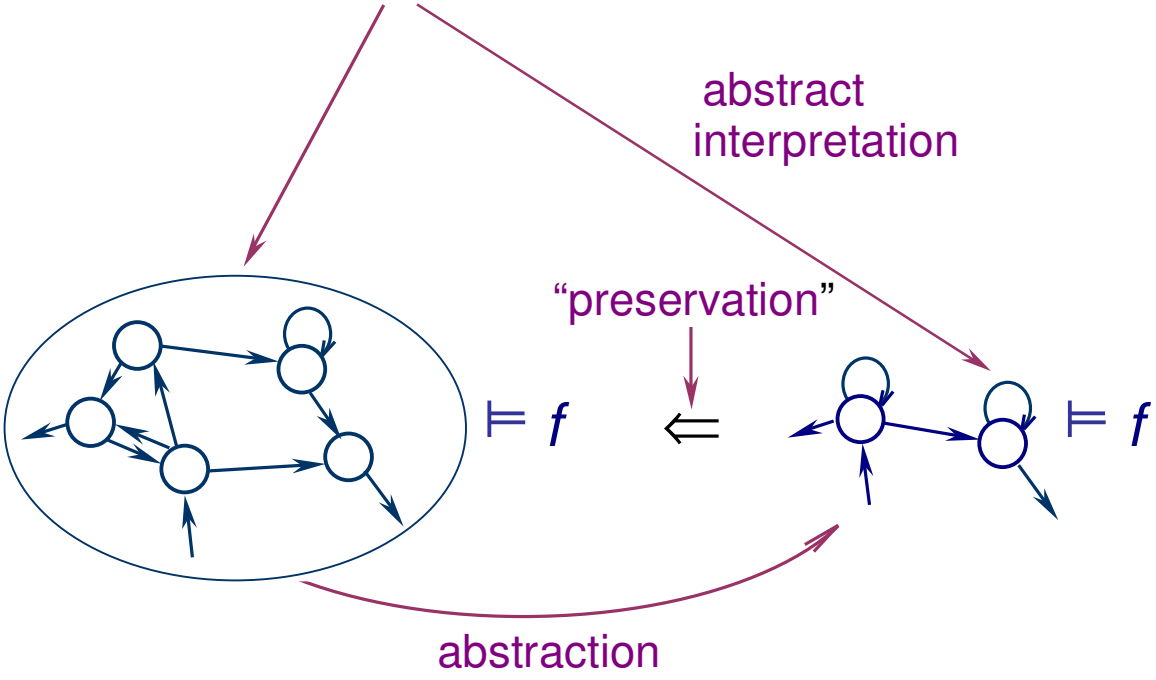
a verification model should be
"abstract enough"

The more abstract the model is the more
difficult is to relate it to the "real" system

tendency of users to specify
too much detail in their models

Abstraction

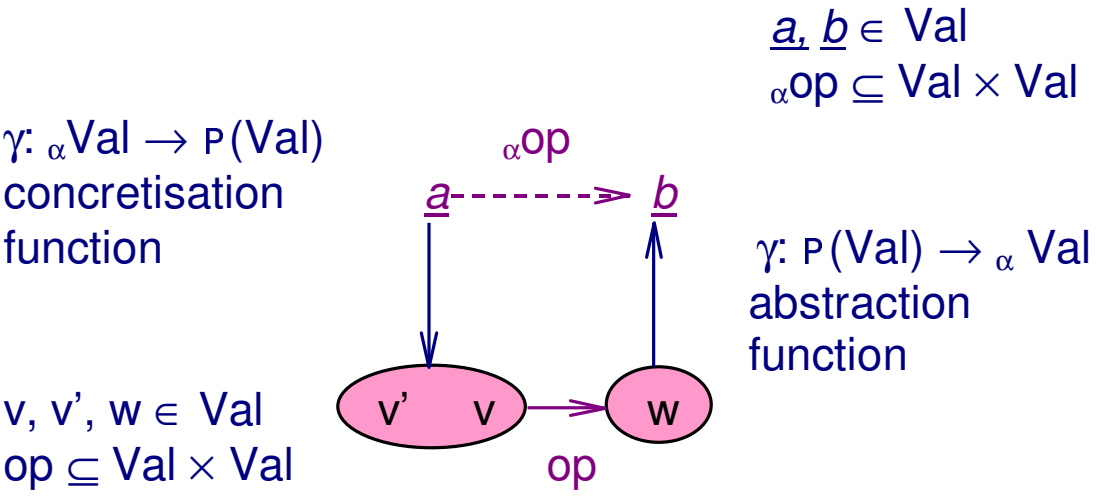
Does system *S* satisfy property *f* ?



Abstraction (continued)

Idea:

Replace data values by descriptions (abstract values) and "mimic" operations. Then every universal property checked to be true on the abstract system holds for the concrete system as well.



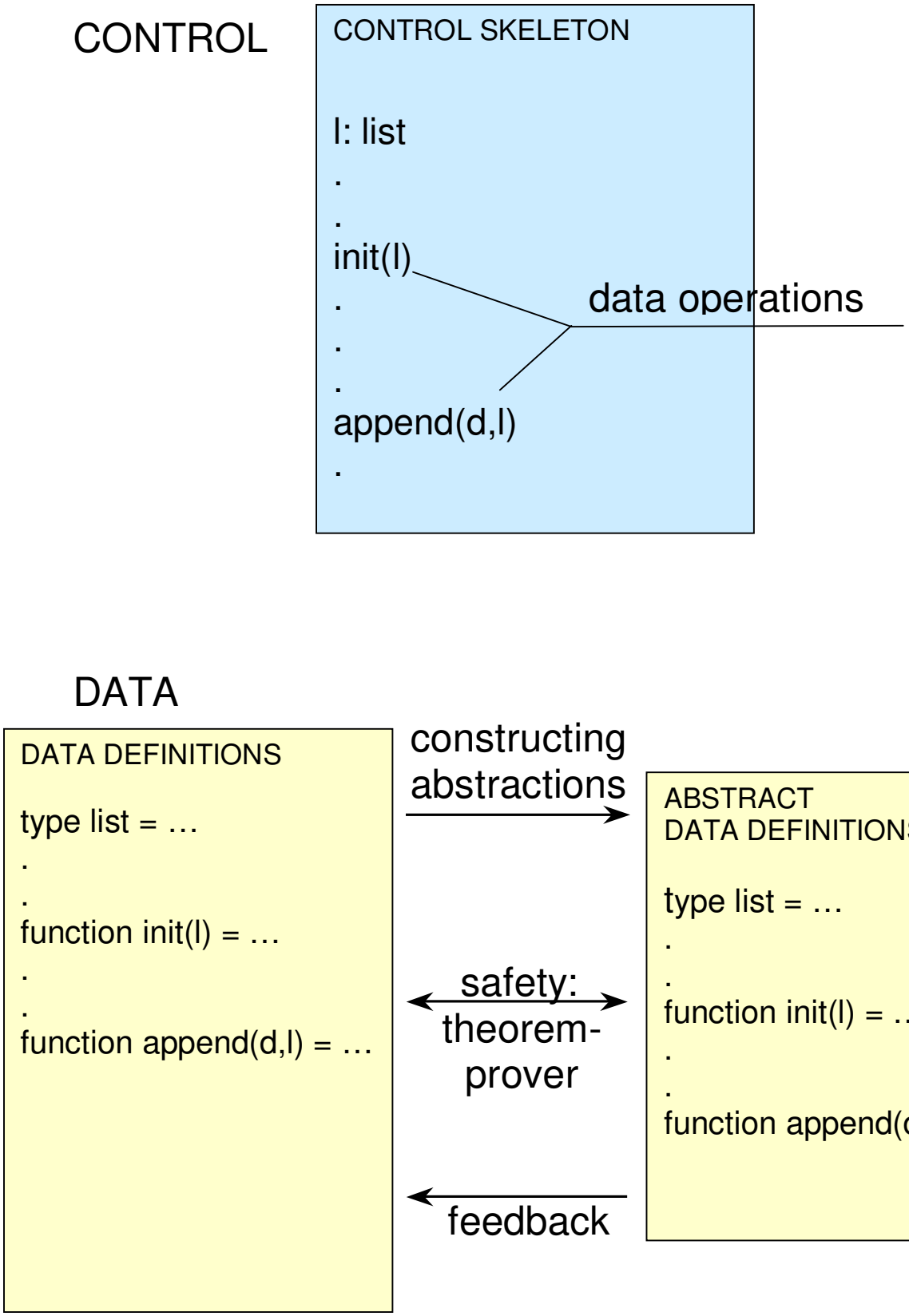
(α, γ) Galois Insertion

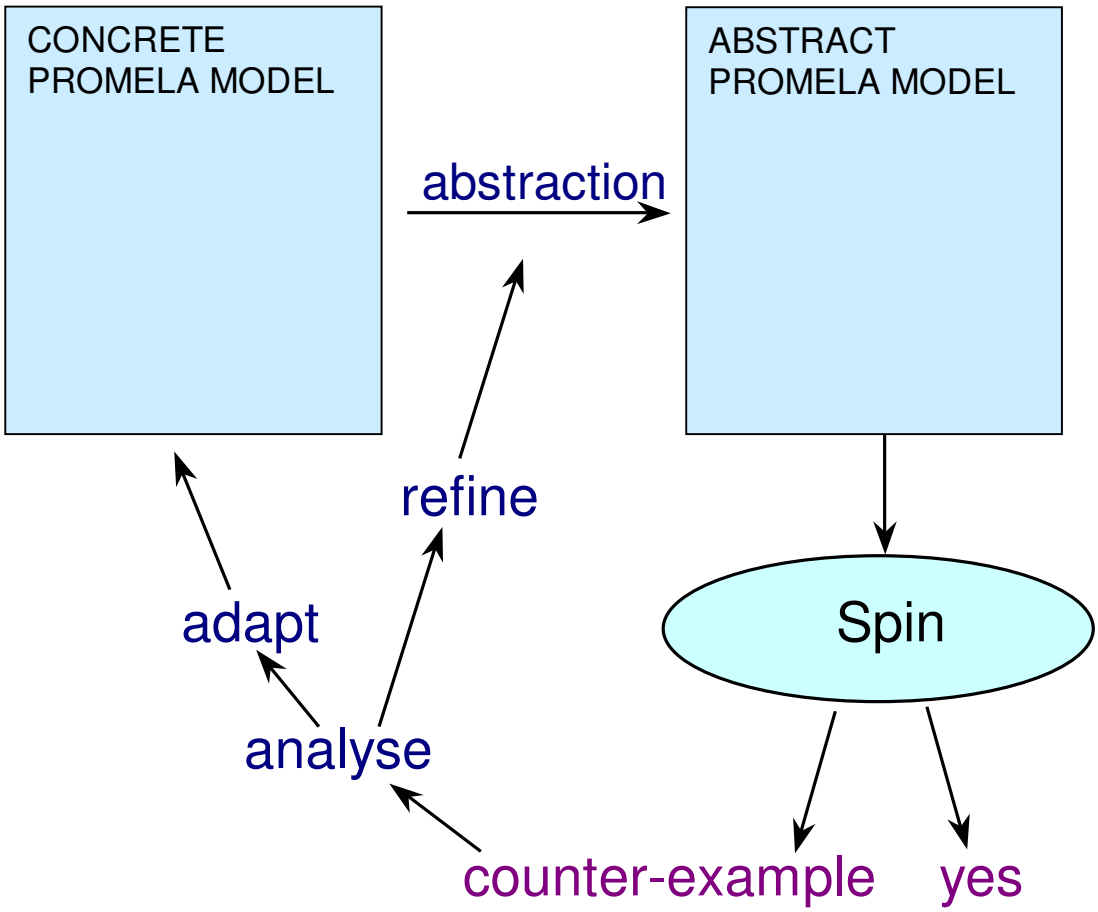
[Cousot & Cousot 77
Clarke, Grumberg & Long 92
Graf & Loiseaux 92
Dams, Gerth & Grumberg 94]

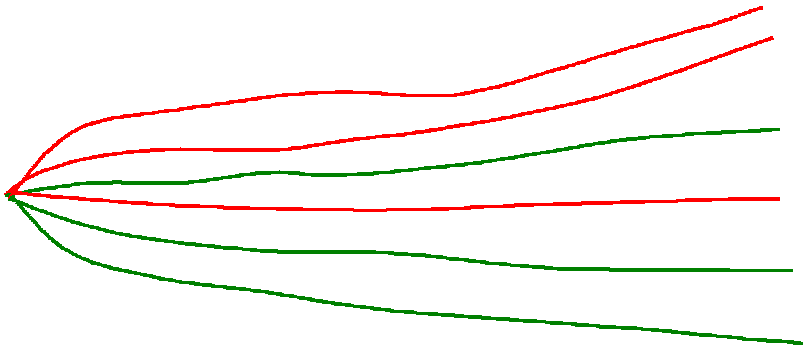
Safety

Problem: constructing abstractions for complex systems is an error-prone process while arguing their faithfulness is often done at an informal level.

Our goal: to provide an extension for Spin to support construction and justification of abstractions.


CONTROL
CONTROL SKELETON
l: list
.
.
init(l)
.
.
.
append(d,l)
.
data operations
DATA
DATA DEFINITIONS
type list = …
.
.
function init(l) = …
.
.
function append(d,l) = …
constructing
abstractions
safety:
theorem-
prover
feedback
ABSTRACT
DATA DEFINITIONS
type list = …
.
.
function init(l) = …
.
.
function append(d,l) = …


CONCRETE
PROMELA MODEL
ABSTRACT
PROMELA MODEL
abstraction
refine
adapt
Spin
analyse
counter-example
yes



INKA

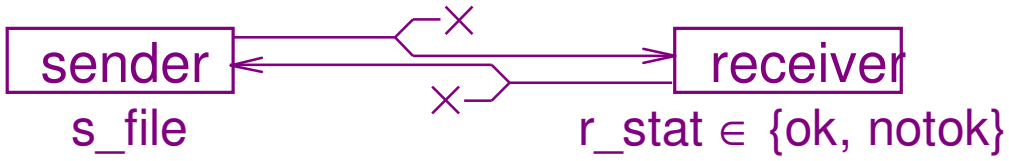
- Logic: first-order logic
- Calculus: resolution & paramodulation & induction schemes
- User interaction

Why INKA

- Data types and data operations are naturally defined using an inductive style

BRP: what?

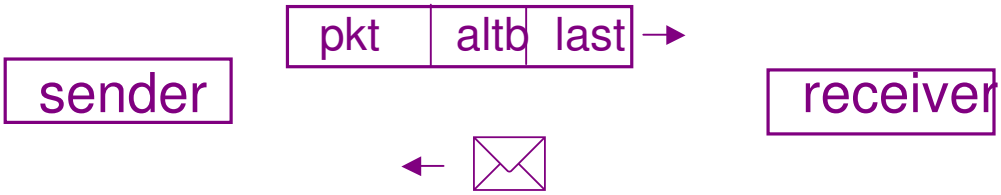
- Transmit large files in chunks
- Channels can either loose messages or deliver correctly within bounded time



s_stat ∈ {ok, notok}

- Constraint:
  limited transmission time per packet
- Requirement:
  “if the first packet of s_file arrives at the receiver, then at the end r_file contains a prefix of s_file”
  [Prefix Property]

BRP: how?



- tries to send packet up to MAX time
- waits Vresend for ack every time
- aborts if still no ack after MAX'th
- then waits Vrecover for receiver
- acks every packet
- waits Vgiveup for next

Abstracting Lists

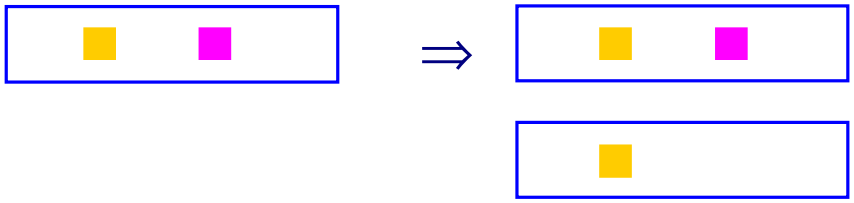
Prefix property:
If the sender is transmitting a list l and the first packet of l arrives at least once at the receiver, then the sequence of packets that the receiving client receives forms a prefix of l

---

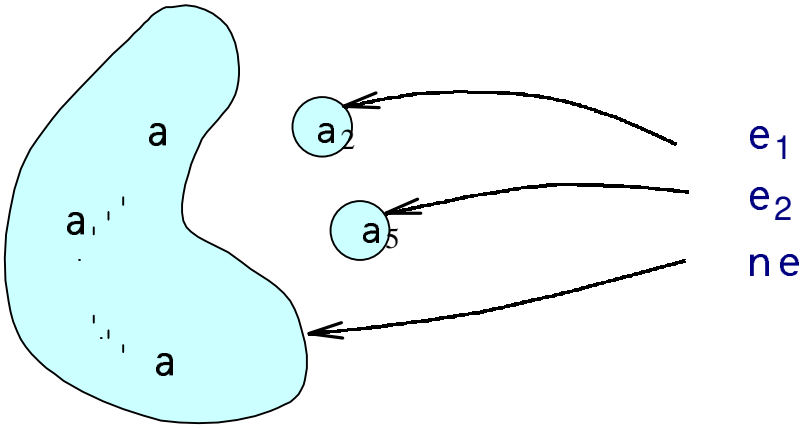
In a list of non-repeating naturals for any two values e_1 and e_1 on positions i and j resp. in the input list, with $i < j$, if the following properties hold for the output list:

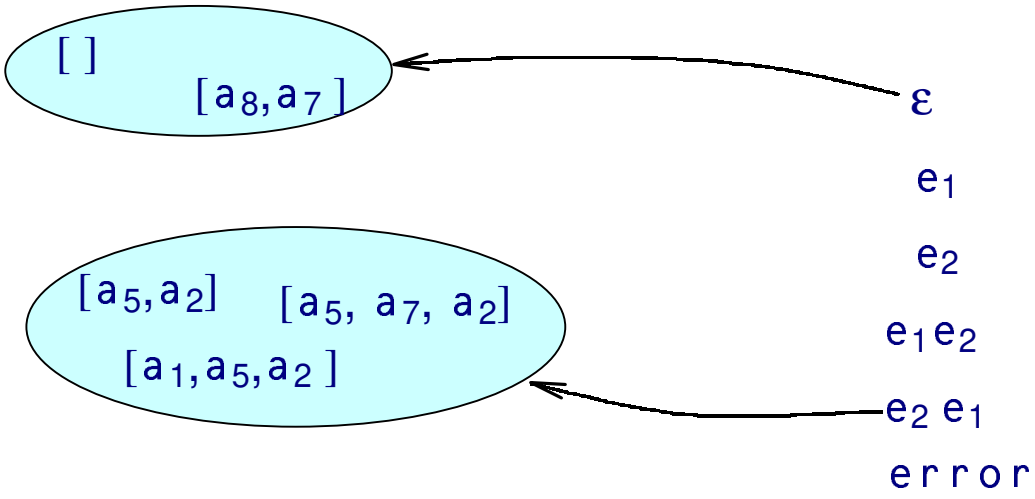
- Either e_2 does not occur, or
  e_1 and e_2 occur in the output list on positions i', j' resp., with $i' < j'$
- For any two values e_1 and e_2 on positions i' and j' resp. in the output list, with $i' < j'$, e_1 and e_2 occur on positions i and j resp. in the input list with $i < j$

Abstracting Lists

1. 

2.





```
% Standard lists
structure nil, cons(car:nat, cdr:list):list;

% append(e,l) appends the single element e to the right-most end of
% list l
function append(e:nat, l:list):list =
if l = nil then cons(e,nil)
otherwise cons(car(l), append(e, cdr(l)));

% Abstracted lists
enum structure epsl, e1l, e2l, e1e2l, e2e1l, error : atail;

structure quad(nonempty:bool, oneelem:bool, hd:aelem, tl:atail):alist;

...

...

...

% The abstract version of the operation append
function comba(a:aelem, l:atail):atail =
if a = ne then l
if a = e1 and l = e2l then e2e1l
if a = e1 and l = epsl then e1l
if a = e2 and l = e1l then e1e2l
if a = e2 and l = epsl then e2l
otherwise error;

function aappend(e: aelem, l:alist):alist =
if nonempty(l) = ff then quad(tt, tt, e, epsl)
otherwise quad(tt, ff, hd(l), check(hd(l), comba(e, tl(l))));


...

...

...
```

% Abstraction function:

```
function alpha(l:list):alist =
if l = nil then quad(ff, ff, ne, epsl)
otherwise
{ f cdr(l) = nil then quad(tt, tt, alphaelem(car(l)), epsl)
  otherwise quad(tt, ff, alphaelem(car(l)),
                 check(alphaelem(car(l)), alphatail(cdr(l))))
};
```

% The statement of the safety proof looks as follows:

```
all l:list  all n:nat
alpha(append(n,l)) = aappend(alphaelem(n),alpha(l));
```

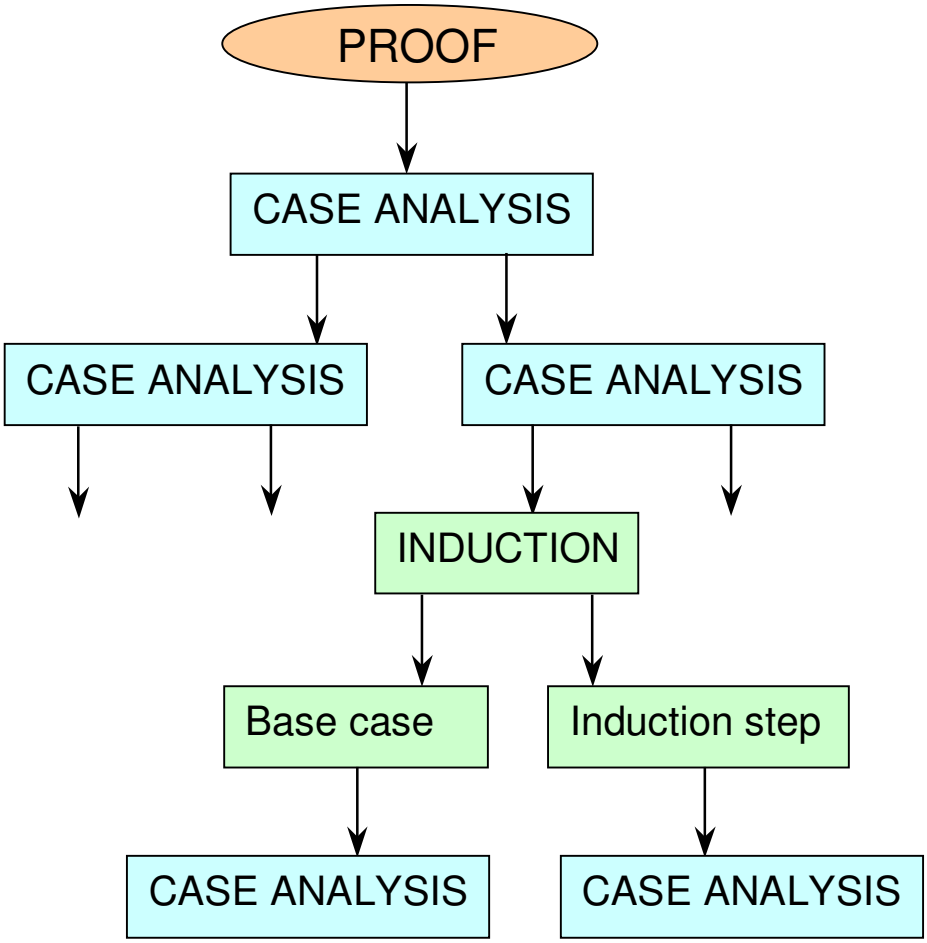
Additional lemma:

```
all x,y:aelem all z:atail
check(x, comba(y, check(x, z))) = check(x, comba(y, z));

all x,y:aelem all z:atail
comb(x, comba(y, z)) = comba(y, comb(x, z));

all n:nat all z:list
alphatail(append(n, z)) = comba(alphaelem(n),
alphatail(z));
```

Tuning-up INKA

Concrete level:
Recursively defined functions.
Need induction to explore their nature.

Abstract level:
Non-recursively defined functions.
Need speculations of nested case analysis.



Results and conclusions

- Though the abstractions seems to be simple, defining even relatively simple functions on abstracted data is error-prone
- After tuning-up, INKA could prove more safety obligations automatically
- Style of defining functions significantly influences the proving process

Future work

- What is a “good” style of function definitions?
- How to connect INKA to Spin?
  (input language problem)