

# Jahob: A System for Verifying Data Structure Consistency

Thomas Wies

Max-Planck-Institut für Informatik, Saarbrücken, Germany  
wies@mpi-inf.mpg.de

Joint work with

Viktor Kuncak  
Martin Rinard  
Andreas Podelski

Karen Zee  
Charles Bouillaguet  
Huu Hai Nguyen

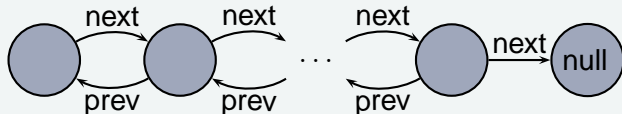
Peter Schmitt  
Bruno Marnette  
Suhabe Bugrara

# Motivation

Statically verify data structure consistency properties.

## Example

### Internal Data Structure Consistency



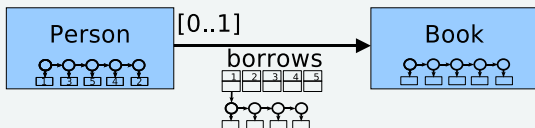
- field **prev** is inverse of field **next**
- field **next** is acyclic

→ inconsistency can cause program crashes.

# External Consistency Properties

## Example

### Library



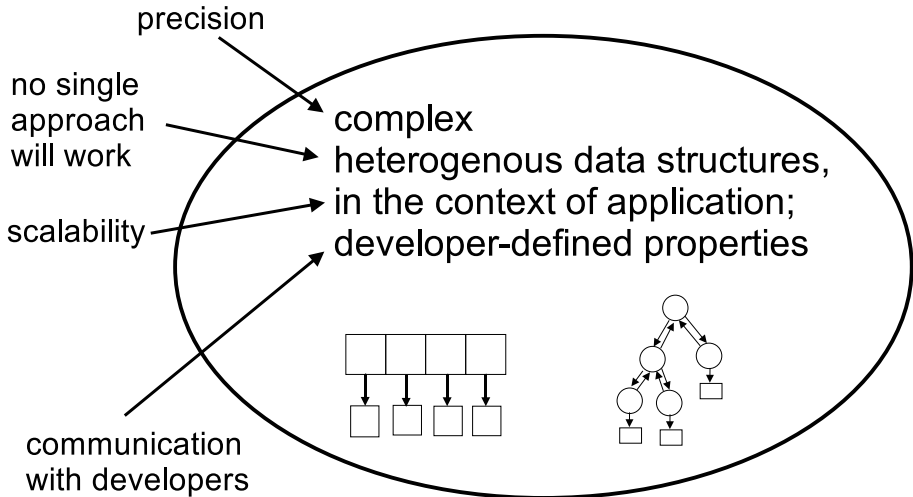
- if a person has borrowed a book, then
    - the person is registered with the library, and
    - the book is in the catalog
  - two persons cannot borrow the same book
- 
- correlate multiple data structures
  - depend on internal consistency
  - capture design constraints (object models)
- inconsistency can cause policy violations.

# Goal

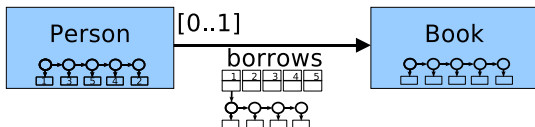
## Proof data structure consistency properties

- for all program executions (**sound**)
- with high level of **automation**
- both **internal** and **external** consistency properties
- both **implementation** and **use** of data structures.

# Challenges



# The Jahob Approach through an Example



```

class Library {
    public static Set persons;
    public static Set books;
    public static Relation borrows;
    ...
}
class Relation {
    private Set[] a;
    private int size;
    ...
    public void add(int i, Object o1){
        ...
    }
}

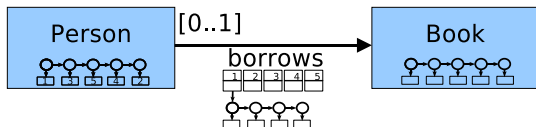
```

```

class Set {
    private Node first;
    ...
    public void add(Object o1){
        Node n = new Node();
        n.data = o1;
        n.next = first;
        first = n;
    }
}

```

# Factoring Out Complexity



if a person has borrowed a book, then

- the person is registered with the library, and
- the book is in the catalog

$$\forall p \ b. (a[p] \neq \text{null} \wedge \exists n. \text{next}^*(\text{borrows}.a[p], n) \wedge n.\text{data} = b) \rightarrow$$

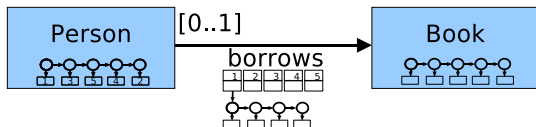
$$(\exists n. \text{next}^*(\text{persons}.first, n) \wedge n.\text{data} = p)$$

$$\wedge (\exists n. \text{next}^*(\text{books}.first, n) \wedge n.\text{data} = b)$$

## Difficulties

- need to track exact reachability properties
- need to correlate multiple data structures

# Factoring Out Complexity



if a person has borrowed a book, then

- the person is registered with the library, and
- the book is in the catalog

$$\forall p \ b. (a[p] \neq \text{null} \wedge \exists n. \text{next}^*(\text{borrows}.a[p], n) \wedge n.\text{data} = b) \rightarrow$$

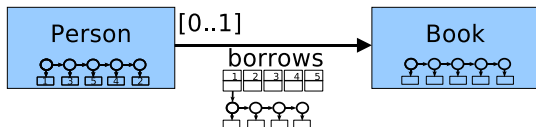
$$(\exists n. \text{next}^*(\text{persons}.first, n) \wedge n.\text{data} = p)$$

$$\wedge (\exists n. \text{next}^*(\text{books}.first, n) \wedge n.\text{data} = b)$$

## Specification Variables

$$\text{Set.content} = \{x \mid \exists n. \text{next}^*(\text{first}, n) \wedge n.\text{data} = x\}$$

# Factoring Out Complexity



if a person has borrowed a book, then

- the person is registered with the library, and
- the book is in the catalog

$$\forall p \ b. \ a[p] \neq \text{null} \wedge b \in \text{borrows.a}[p].\text{content} \rightarrow$$

$$p \in \text{persons.content}$$

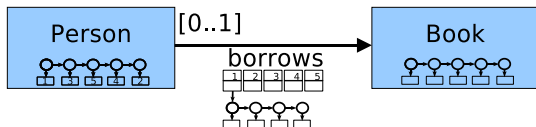
$$\wedge b \in \text{books.content}$$

## Specification Variables

$$\text{Set.content} = \{ x \mid \exists n. \text{next}^*(\text{first}, n) \wedge n.\text{data} = b \}$$

$$\text{Relation.content} = \{ (x, y) \mid a[x] \neq \text{null} \wedge y \in a[x].\text{content} \}$$

# Factoring Out Complexity



if a person has borrowed a book, then

- the person is registered with the library, and
- the book is in the catalog

$$\begin{aligned} \forall p b . (p, b) \in \text{borrows.content} &\rightarrow \\ &p \in \text{persons.content} \\ &\wedge b \in \text{books.content} \end{aligned}$$

## Specification Variables

$$\text{Set.content} = \{ x \mid \exists n . \text{next}^*(\text{first}, n) \wedge n.\text{data} = b \}$$

$$\text{Relation.content} = \{ (x, y) \mid a[x] \neq \text{null} \wedge y \in a[x].\text{content} \}$$

# Defining Interfaces using Specification Variables

```

class Set {
  public Node first;
  ...
  /*: public specvar content :: objset;
  vardefs "content == {x. EX n. n..Node.data = x &
              rtrancl_pt (% v1 v2. v1..Node.next = v2) first n}";
  ...
  invariant "tree [Node.next]";
  */
  public void add(Object o1)
    /*: requires "o1 ~= content"
       modifies "content"
       ensures "content = old content Un {o1}"
    */
    { ... }
}

```

# Use Interfaces to Verify Data Structure Clients

```

class Library {
  public static Set persons;
  ...
  /*: invariant "ALL p b. (p,b) : borrows..Relation.content →
      p : persons..Set.content & b : books..Set.content" */

  public static void checkOutBook(Person p, Book b)
  /*: requires "p ~= null & b ~= null &
      b : books..Set.content & p : persons..Set.content"
  modifies "borrows..Relation.content"
  ensures "(ALL p1. (p1,b) ~: old borrows..Relation.content) →
      borrows..Relation.content = old (borrows..Relation.content) Un (p,b)
  & (EX p1. (p1,b) : old borrows..Relation.content →
      borrows..Relation.content = old borrows..Relation.content)"
  */
  { ... }
}

```

# Overview of the Jahob Approach

Reasoning about program in terms of simpler interfaces

- uses of interfaces
- global consistency

**scalable analyses**

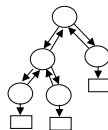
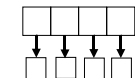
Application  
(Data Structure Client)

A interface

B interface

A implementation

B implementation



Checking that interfaces reflect implementations  
and internal consistency is preserved - **precise analyses**

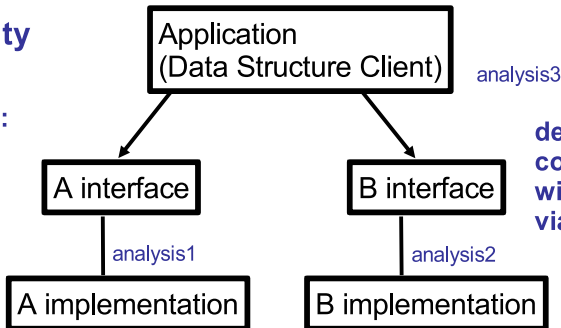
# Overview of the Jahob Approach

Used in manual verification, VDM, ESC/Java as **data abstraction**

Reasoning about program in terms of simpler structures

**scalability**

**heterogeneity:  
multiple  
analyses**



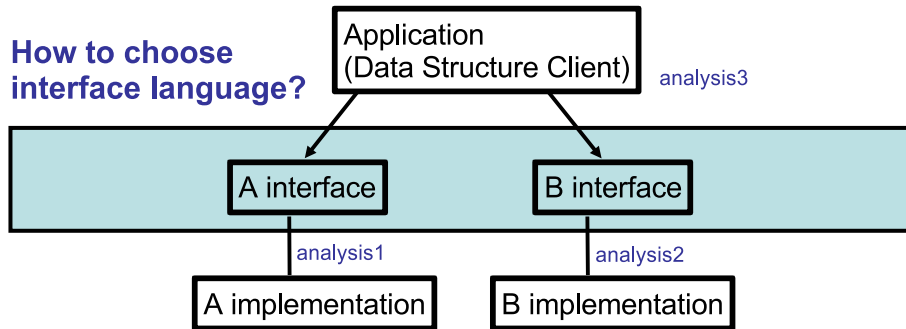
**developers  
communicate  
with system  
via interfaces**

**precision:  
within data  
structures**

# Overview of the Jahob Approach

Key question in automating approach (while keeping it useful)

**How to choose interface language?**



# Jahob's specification languages

## Interface specification language

### sets and relations

- express key properties of data structures
- verifiable on both sides of interfaces

## Internal specification language

### subset of Isabelle/HOL formulas

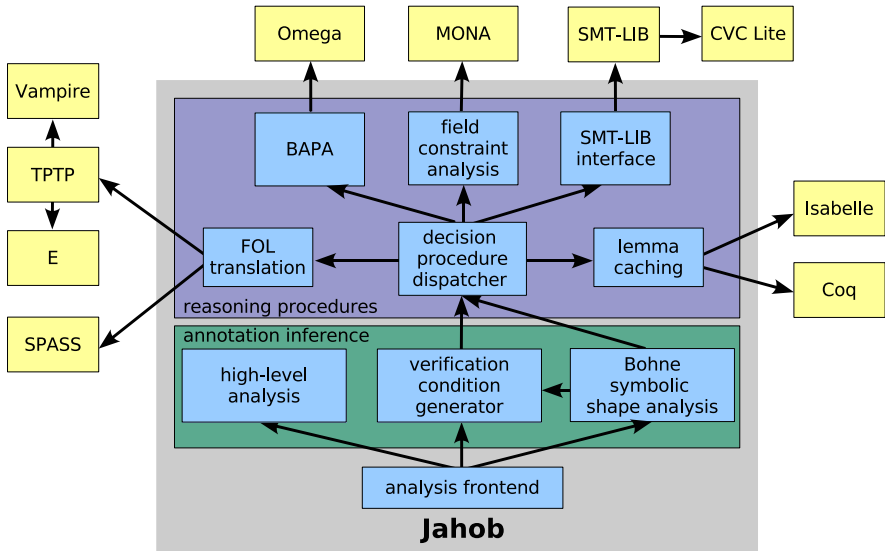
- natural syntax
- unifying semantic foundation for all specification constructs
- no artificial limitations regarding expressiveness
- decision procedures can be used to automate reasoning
- interactive theorem provers can be used for
  - debugging the system
  - proving the most difficult theorems interactively

# Interactive Theorem Proving

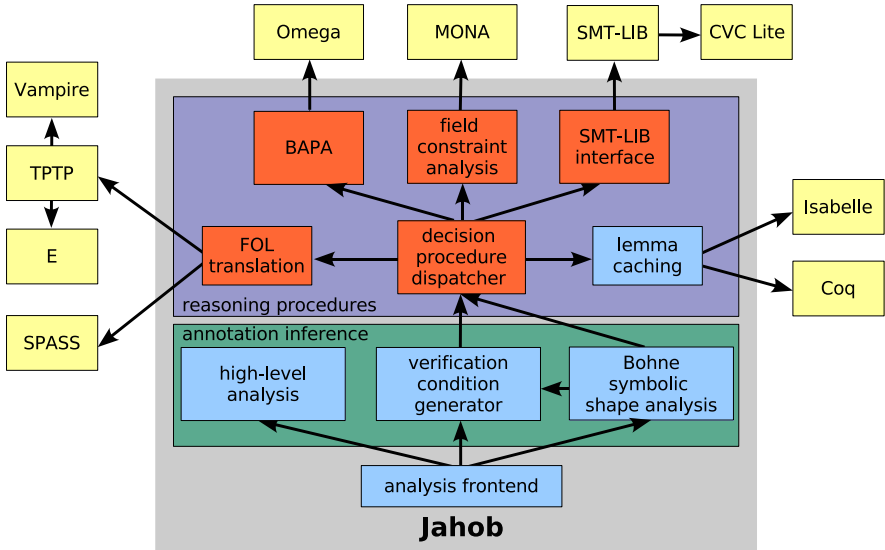
## Jahob provides

- **Isabelle interface**: Jahob generates proof obligations in Isabelle format and prints them in the right syntax.
  - **Lemma caching**: externally proved Isabelle obligations can be used as lemmas. Matching of lemmas tolerates:
    - additional assumptions
    - some simple strengthening
  - **Automated reasoning**: external tools are used to automate Isabelle formulas.
- Reasoning back end is directly accessible through **stand-alone executable**.

# Jahob System Architecture



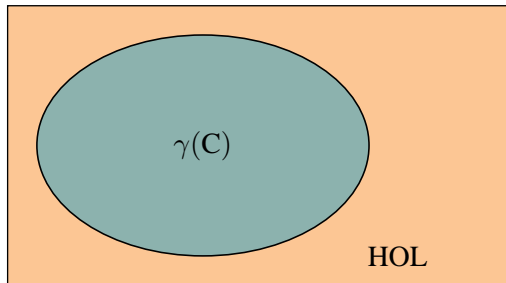
# Automating HOL formulas



# Abstracting HOL Formulas

Given class of **tractable formulas**  $C$  and embedding function

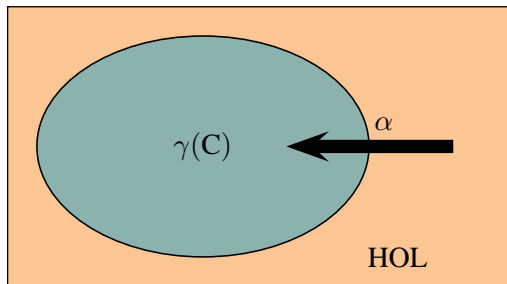
$$\gamma \in C \rightarrow \text{HOL}$$



# Abstracting HOL Formulas

Given class of **tractable formulas**  $C$  and embedding function

$$\gamma \in C \rightarrow \text{HOL}$$



Define abstraction  $\alpha$  such that

$$\gamma(\alpha(F)) \models F$$

Simple example for  $\gamma = id$

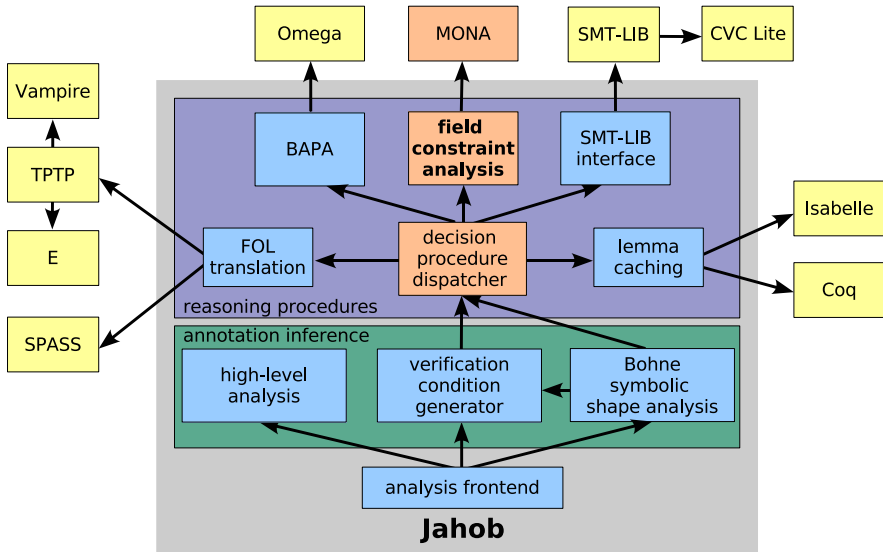
$$\alpha(F) = \begin{cases} F, & F \in C \\ \text{false}, & \text{otherwise} \end{cases}$$

# Abstracting HOL Formulas

## Improvements

- apply equivalence transformations to eliminate HOL constructs
  - eliminate definitions using one-point rule, e.g.  
 $f = (\lambda x. 2x) \rightarrow F$  rewrites to  $F[f := \lambda x. 2x]$
  - apply  $\beta$ -reduction
- apply approximation recursively on subformulas
  - replace intractable subformulas with fresh Boolean variables
  - replace common subformulas with the same variable  
 $(x > 0 \rightarrow F) \wedge (x \leq 0 \rightarrow G)$  rewrites to  $(b \rightarrow F) \wedge (\neg b \rightarrow G)$
- flatten terms to minimize effect of approximation
- apply multiple approximations to the same formula

# Field Constraint Analysis



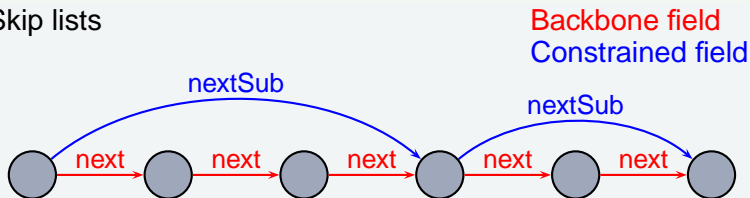
# Monadic Second-Order Logic (MSOL)

## MSOL

- allows complete reasoning about reachability in trees
- but **only** trees.

## Example

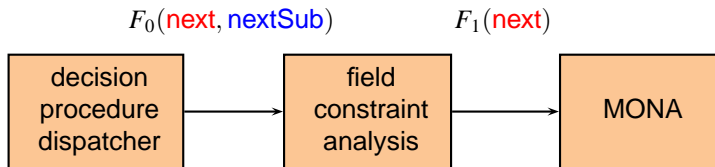
### Skip lists



Derived field satisfies **field constraint**

$$\forall x y . \text{nextSub}(x) = y \rightarrow \text{next}^+(x, y)$$

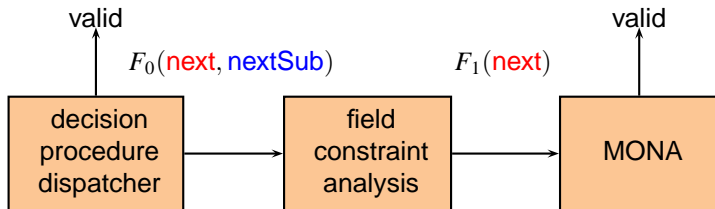
# Eliminating Constrained Fields



Derived field satisfies **field constraint**

$$\forall x y. \text{nextSub}(x) = y \rightarrow \text{next}^+(x, y)$$

# Eliminating Constrained Fields

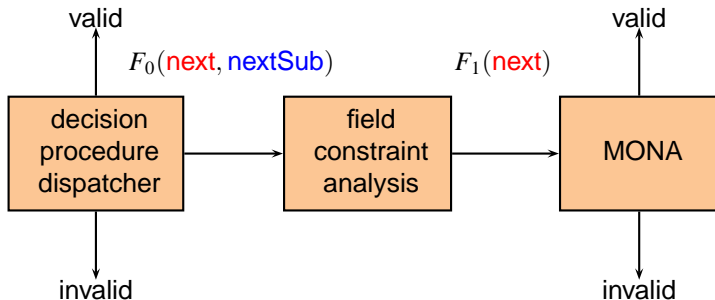


Sound

Derived field satisfies **field constraint**

$$\forall x y. \text{nextSub}(x) = y \rightarrow \text{next}^+(x, y)$$

# Eliminating Constrained Fields



Sound

Complete (for interesting class of formulas)

→ previous approaches limited to deterministic field constraints

Derived field satisfies **field constraint**

$$\forall x y. \text{nextSub}(x) = y \rightarrow \text{next}^+(x, y)$$

# Proving a Sequent with formDecider

## Example

### Preservation of field constraint

```
$ cat skiplist-sequent
```

```
[|(Skiplist.reach = (% a b. rtranc1_pt (% (x::obj) (y::obj). x..Node.next = y) a b));
tree [Node.next];
(ALL x y. Node.nextSub x = y -->
(x = null --> y = null) & (x ~= null --> Skiplist.reach (x..Node.next) y));
~(Skiplist.reach Skiplist.root e);
Skiplist.reach Skiplist.root sprev;
(ALL x. x ~= null & ~(Skiplist.reach Skiplist.root x) -->
~(EX e. e ~= null & e..Node.next = x) & (x..Node.next) = null);
Skiplist.reach sprev prev;
Skiplist_.reach current scurrent;
(scurrent = sprev..Node.nextSub);
(current = prev..Node.next);
(Skiplist.reach_1 = (% a b. rtranc1_pt (% (x::obj) (y::obj).
(fieldWrite (fieldWrite Node.next e current) prev e) x = y) a b));
(0 < e..Node.v);
(x ~= null);
(fieldWrite (fieldWrite Node.nextSub sprev e) e scurrent) x = y
|] ==>
Skiplist.reach_1 ((fieldWrite (fieldWrite Node.next e current) prev e) x) y
```

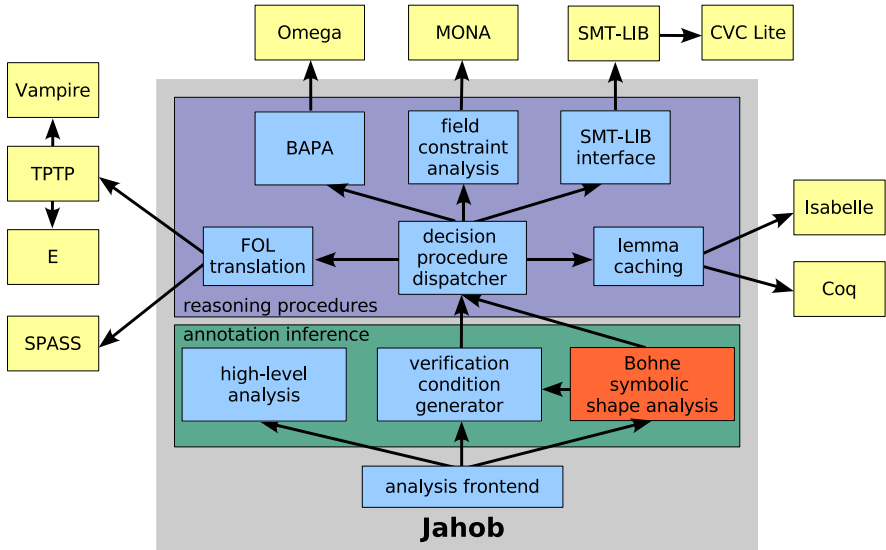
```
$ formDecider skiplist-sequent -usedp mona
```

```
Begining proof process...
```

```
...input formula (skiplist-sequent) is VALID. Yahoooo !
```

```
$
```

# Symbolic Shape Analysis



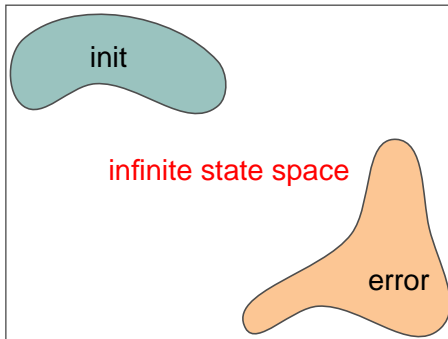
# Symbolic Shape Analysis

## Advantages of symbolic approach

- ① **high degree of automation**: software model-checking
- ② **predictability**: based on decision procedures
- ③ **generality**: a priori no limitation to specific data structures
- ④ **scalability**: assume/guarantee reasoning

# Software Model-Checking

## Predicate Abstraction

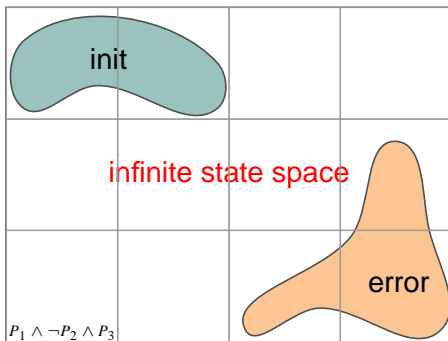


**Goal:** compute states reachable from **init**.

- may take infinitely many computation steps
- use abstraction

# Software Model-Checking

## Predicate Abstraction



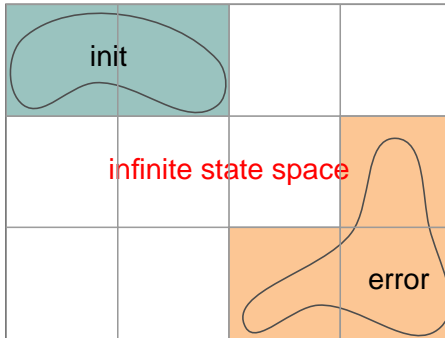
Partition state space according to **state predicates**:

$$P = x + y \geq 0$$

Abstract states are **bit-vectors**.

# Software Model-Checking

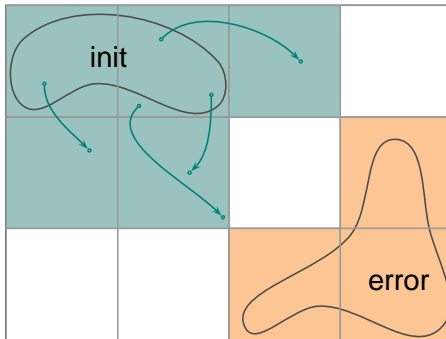
## Predicate Abstraction



Abstract the concrete states.

# Software Model-Checking

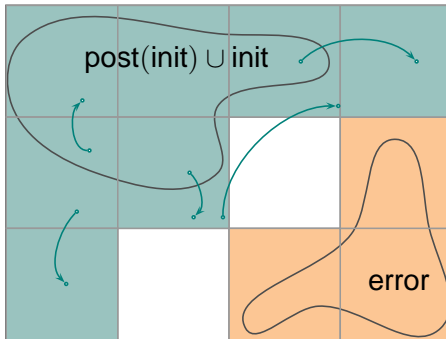
## Predicate Abstraction



Compute reachable abstract states...

# Software Model-Checking

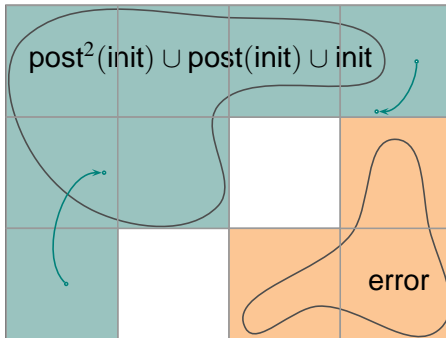
## Predicate Abstraction



... until fixpoint reached.

# Software Model-Checking

## Predicate Abstraction



... until fixpoint reached.



# Predicate Abstraction

## Benefits of Predicate Abstraction

- generic framework
- useful even if predicates are manually supplied
- combination with abstraction refinement results in fully automated verification technique

Software model-checker: SLAM, BLAST, ARMC, MAGIC, ...

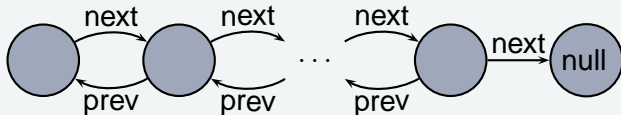
Successfully applied to control-intensive software, e.g. device drivers.

What about data structure consistency?

# Quantified Invariants

## Example

### Doubly-linked lists



field **prev** is inverse of field **next**:

$$\forall v. \text{next}(v) \neq \text{null} \rightarrow \text{prev}(\text{next}(v)) = v$$

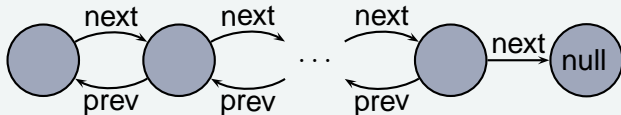
field **next** is acyclic:

$$\forall v. \text{next}^*(v, \text{null})$$

# Quantified Invariants

## Example

### Doubly-linked lists



field **prev** is inverse of field **next**:

$$\forall v. \text{next}(v) \neq \text{null} \rightarrow \text{prev}(\text{next}(v)) = v$$

field **next** is acyclic:

$$\forall v. \text{next}^*(v, \text{null})$$

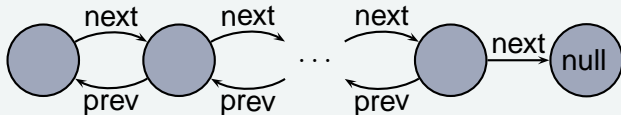
## Potential solutions with standard predicate abstraction

- **quantified predicates**
  - decision procedures might not be able to handle quantifiers
  - finding the right predicates is as hard as finding the invariant

# Quantified Invariants

## Example

### Doubly-linked lists



field **prev** is inverse of field **next**:

$$\forall v. \text{next}(v) \neq \text{null} \rightarrow \text{prev}(\text{next}(v)) = v$$

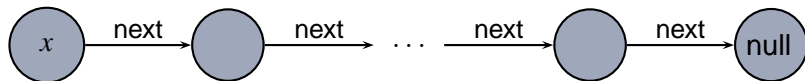
field **next** is acyclic:

$$\forall v. \text{next}^*(v, \text{null})$$

## Potential solutions with standard predicate abstraction

- **quantified predicates**
  - decision procedures might not be able to handle quantifiers
  - finding the right predicates is as hard as finding the invariant
- **Skolemization**
  - does not work for non-local properties such as reachability

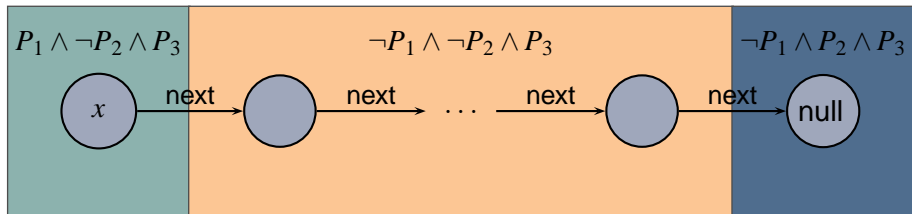
# Three-valued shape analysis [Sagiv, Reps, Wilhelm]



# Three-valued shape analysis [Sagiv, Reps, Wilhelm]

Partition heap according to finitely many **predicates on heap objects**.

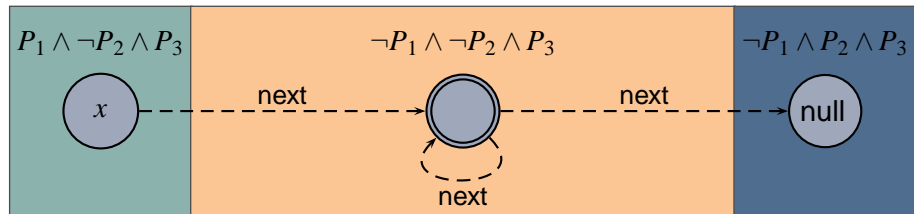
$$P_1 = \{v \mid v = x\} \quad P_2 = \{v \mid v = \text{null}\} \quad P_3 = \{v \mid \text{next}^*(x, v)\}$$



# Three-valued shape analysis [Sagiv, Reps, Wilhelm]

Partition heap according to finitely many **predicates on heap objects**.

$$P_1 = \{v \mid v = x\} \quad P_2 = \{v \mid v = \text{null}\} \quad P_3 = \{v \mid \text{next}^*(x, v)\}$$



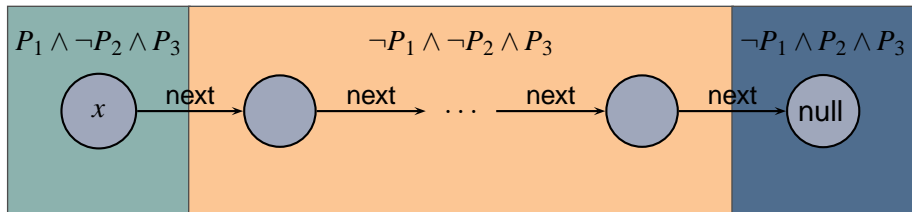
→ shape graph

Trend is to move to symbolic approaches.

# Boolean Heaps

Partition heap according to finitely many **predicates on heap objects**.

$$P_1 = \{v \mid v = x\} \quad P_2 = \{v \mid v = \text{null}\} \quad P_3 = \{v \mid \text{next}^*(x, v)\}$$



Describe partitioning as a universally quantified formula

$$\forall v. P_1 \wedge \neg P_2 \wedge P_3 \vee \neg P_1 \wedge \neg P_2 \wedge P_3 \vee \neg P_1 \wedge P_2 \wedge P_3$$

→ Boolean heaps [Podelski, Wies]

# Boolean Heaps

## Abstract domain

$$\underbrace{\bigvee_i \bigvee v . \bigvee_j \underbrace{\bigwedge_k P_{i,j,k}(v)}_{\text{abstract object}}}_{\text{Boolean heap}}$$

sets of Boolean heaps

→ sets of sets of bitvectors

## Compared to predicate abstraction

$$\bigvee_i \underbrace{\bigwedge_j P_{i,j}}_{\text{abstract state}}$$

sets of abstract states

→ sets of bitvectors

→ Boolean heaps provide extra precision needed for shape analysis.

# Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\text{post}^\#(H) = ?$$

# Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\text{post}^\#(H) = \alpha \circ \text{post} \circ \gamma(H)$$

Answer is given by **abstract interpretation** framework.

**But:** naive implementation requires  $2^{2^n}$  decision procedure calls.

# Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\begin{aligned} \text{post}^\#(H) &= \alpha \circ \text{post} \circ \gamma(H) \\ &\models \text{CartesianPost}(H) \end{aligned}$$

→ use additional **Cartesian abstraction**.

# Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\begin{aligned}
 \text{post}^\#(H) &= \alpha \circ \text{post} \circ \gamma(H) \\
 &\models \text{CartesianPost}(\forall v. \bigvee_i C_i \text{ as } H) \\
 &= \forall v. \bigvee_i \bigwedge \{ P \mid \Gamma \wedge C_i \models \text{wlp}(P) \}
 \end{aligned}$$

## Context-sensitive Cartesian Post

Compute effect of heap updates locally

- for each abstract object  $C_i$
- and independently for each heap predicate  $P$
- but take into account some global information  $H \models \Gamma$

# Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\begin{aligned}
 \text{post}^\#(H) &= \alpha \circ \text{post} \circ \gamma(H) \\
 &\models \text{CartesianPost}(\forall v. \bigvee_i C_i \text{ as } H) \\
 &= \forall v. \bigvee_i \bigwedge \{ P \mid \Gamma \wedge C_i \models \text{wlp}(P) \}
 \end{aligned}$$

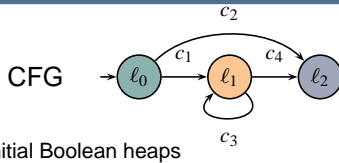
## Context-sensitive Cartesian Post

What is gained?

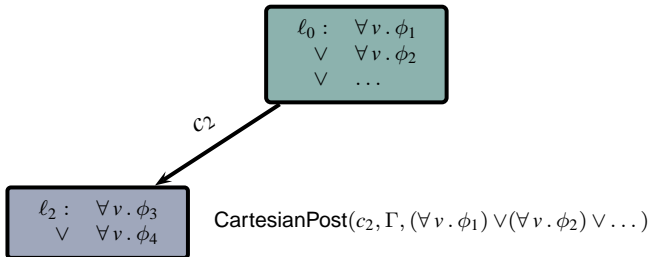
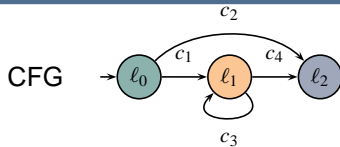
- abstraction reduced to checking verification conditions
- decision procedure queries are quantifier free
- requires  $\mathcal{O}(n^k)$  decision procedure calls (in practice)
- $\text{post}^\#$  can be computed from Cartesian post.

# On-Demand Abstraction

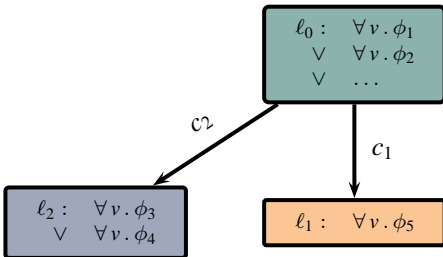
$l_0 : \forall v. \phi_1$   
 $\vee \forall v. \phi_2$   
 $\vee \dots$



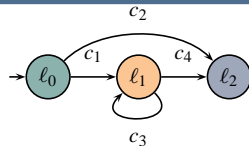
# On-Demand Abstraction



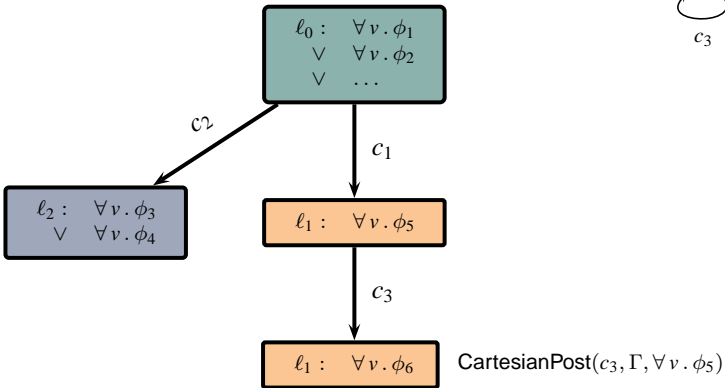
# On-Demand Abstraction



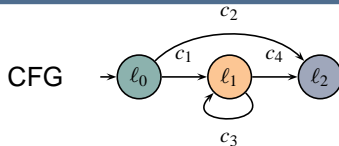
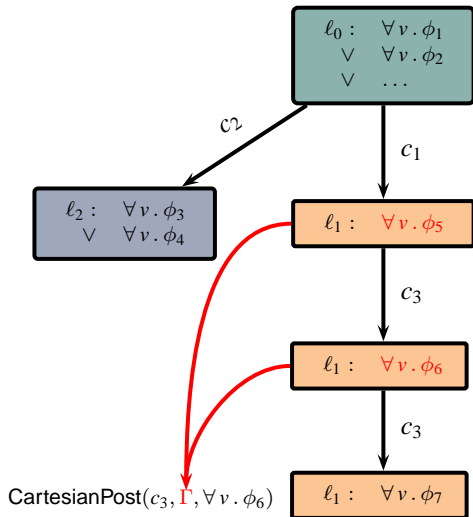
CFG


 $\text{CartesianPost}(c_3, \Gamma, (\forall v. \phi_1) \vee (\forall v. \phi_2) \vee \dots)$

## On-Demand Abstraction



# On-Demand Abstraction



- abstraction takes into account already discovered Boolean heaps.
- abstract transformers are **precomputed** and recomputed on-demand only if  $\Gamma$  changes.
- decision procedure calls are cached '**semantically**' to make recomputation fast.

# Some Experimental Results

## Bohne benchmarks (without abstraction refinement)

benchmark	used DP	# predicates total (manually supplied)	# validity checker calls total (cache hits)	running time total (DP)
DLL.addLast	MONA	7 (0)	118 (19%)	2s (69%)
List.reverse	MONA	7 (2)	371 (22%)	4s (72%)
SortedList.add	MONA, CVC lite	16 (1)	368 (40%)	11s (65%)
Skiplist.add	MONA	20 (0)	787 (44%)	26s (57%)
Tree.add	MONA	13 (0)	358 (31%)	31s (92%)
ParentTree.add	MONA	13 (0)	362 (32%)	33s (91%)
Linear.arrayInv	CVC lite	7 (5)	882 (52%)	57s (97%)

# Some Experimental Results

## Further programs analyzed with Hob/Jahob

- **Water particle simulation:** ordering of computation phases
- **Web server:** initialization, ordering, data structures  
serving `http://hob.csail.mit.edu`
- **Ongoing work:**
  - turn-based strategy game, collection classes
  - operating system data structures (Verisoft project)

## Publications Related to Jahob

- Field Constraint Analysis, T. Wies, V. Kuncak, P. Lam, A. Podelski, and M. Rinard, VMCAI 2006.
- Relational Analysis of Algebraic Datatypes, V.Kuncak and D. Jackson, ESEC/FSE 2005.
- Boolean Heaps, A. Podelski, T. Wies, SAS 2005.
- Decision Procedures for Set-Valued Fields, V.Kuncak and M. Rinard, AIOOL 2005.
- Boolean Algebra with Presburger Arithmetic, V. Kuncak and M. Rinard, CADE-20.
- Generalized Typestate Checking for Data Structure Consistency, P. Lam, V. Kuncak, and M. Rinard, VMCAI 2005.
- Combining Theorem proving with Static Analysis for Data Structure Consistency, K. Zee, P. Lam, V. Kuncak, M. Rinard, SVV 2004.

# Conclusion

## Credo

Abstraction is the key for making verification feasible.

Jahob supports abstraction on multiple levels via

- specification variables and
- abstract interpretation.

## Contributions of Jahob

- a framework for verifying data structure consistency in larger systems
- new analyses for verifying data structures
- automated reasoning techniques which are applicable in the more general setting of (interactive) theorem proving.

# Future Work

## Jahob

- incorporate more specialized decision procedures
- incorporate high-level analyses
- deploy within software development environments
- address the construction of models
  - counterexamples for models via model finders (Alloy, Paradox, . . .)
  - testing, run-time checking of specifications

## Bohne

- automated abstraction refinement
- analysis of arrays
- interprocedural analysis

# Related Work

## Shape analysis

- Jones, Muchnik '79: memory optimizations
- Larus, Hilfinger'88: detecting conflicts in memory accesses
- Hendren, Nicolau '90: parallelization, connection analysis
- Chase, Wegman, Zadeck'90: allocation-site model
- Klarlund, Schwartzbach'93: graph types
- Deutsch '94: symbolic bounds on paths
- Fradet, Metayer '97: graph-grammars
- Sagiv, Reps, Wilhelm '99: 3-valued framework
- Lev-Ami, Sagiv '00: TVLA implementation
- Moeller, Schwartzbach '01: PALE based on MONA
- Yorsh, Reps, Sagiv '04: assume/guarantee reasoning for 3VL
- McPeak, Necula '05: local pointer properties
- Rugina, Hackett'05: region-based

# Related Work

## Predicate abstraction

- Graf, Saidi '97: using PVS
- Ball, Rajamani, Podelski '01: Cartesian abstraction
- Ball, Majumdar, Millstein, Rajamani '01: SLAM
- Henzinger, Jhala, Sutre '02: BLAST
- Flanagan, Qadeer '02: use of Skolem constants
- Lahiri, Seshia, Bryant '04: UCLID, indexed predicates
- Balaban, Pnueli, Zuck '05: small models for lists
- Bingham, Rakamaric '06: abstraction of lists
- Beyer, Henzinger, Théoduloz '06: lazy shape analysis

## Related Work

### Decision procedures and theorem provers

- Barrett, Berezin '04: CVC Lite
- Detlef, Nelson, Saxe '03: Simplify
- Ball, Lahiri, Musuvathi '05: Zap
- Thatcher, Wright '68: MSOL over finite trees
- Klarlund, Moeller, Schwartzbach '00: MONA
- Yorsh, Rabinovich, Sagiv, Meyer, Bouajjani'06: reachability logic
- BAPA: Feferman, Vaught '59; Zarba'04,'05
- Voronkov'95: Vampire, Weidenbach'01: Spass
- Gordon'85: HOL, Pfenning '91: LF, Coquand, Huet'85: Coq
- Constable, Allen, Bromley, Cleaveland, Cremer, Harper, Howe, Knoblock, Mendler, Panangaden, Sasaki, Smith '86: NuPRL
- Gray, Hickey, Nogin, Tapus: MetaPRL
- Kaufmann, Manolios, Moore '00: ACL2
- Nipkow, Paulson, Wenzel '02: Isabelle