Automated Reasoning for Reliable Software

Viktor Kuncak

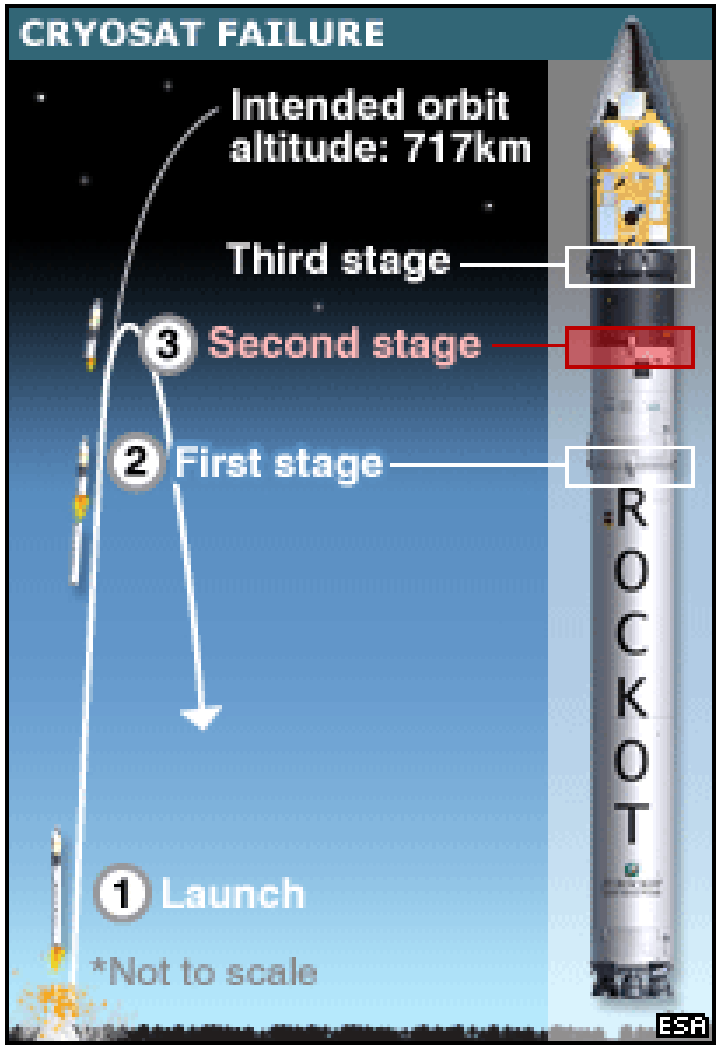
École Polytechnique Fédérale de Lausanne

Importance of Software Reliability

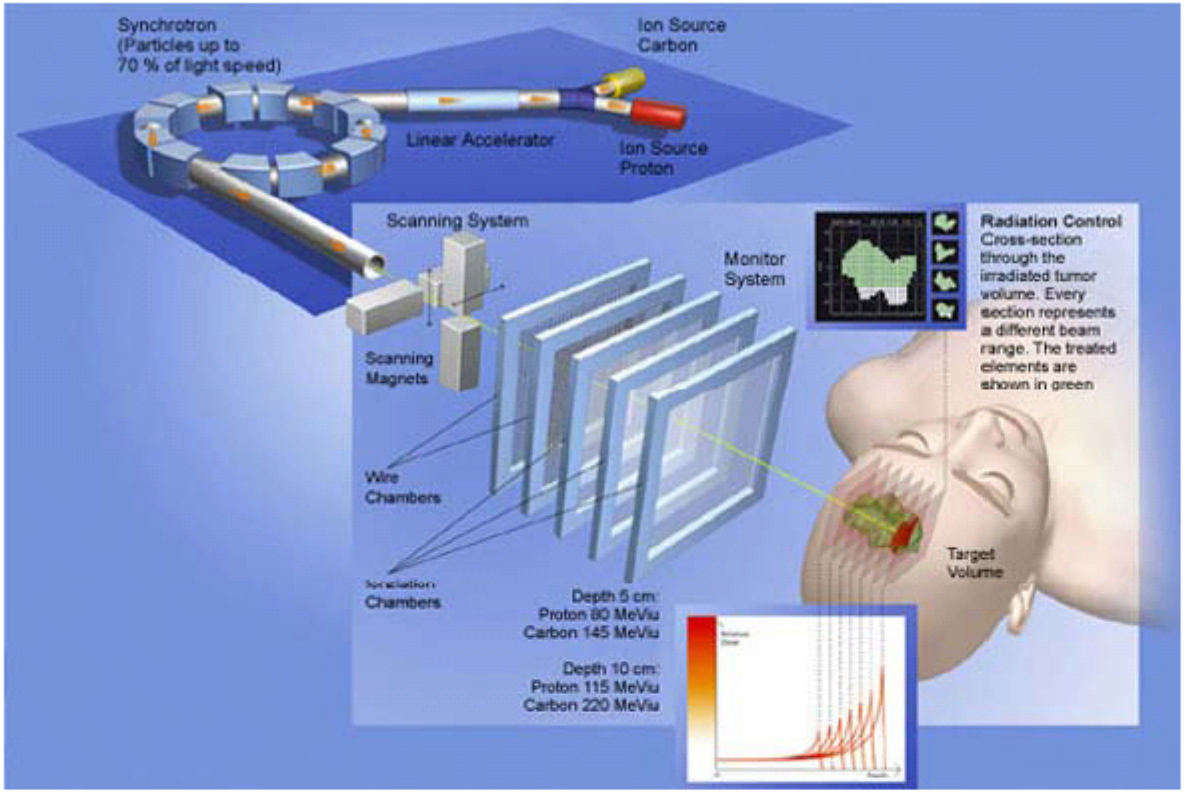
October 2007

Cryosat, a satelite worth

135’000’000 €



sketch from BBC and ESA



Radio Therapy

Between June 1985 and January 1987, a computer-controlled radiation therapy machine, called the Therac-25, massively overdosed six people. These accidents have been described as the worst in the 35-year history of medical accelerators [6].

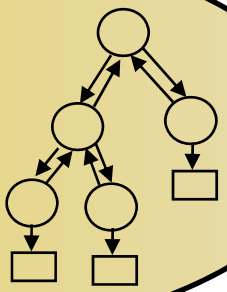
Nancy Leveson
Safeware: System Safety and Computers
Addison-Wesley, 1995

Life-Critical Medical Devices

Program Verification for Reliability

Java source code

```
. . .
void remove(x : Node) {
  Node prev = null;
  Node current = root;
  while (current != null) {
    if (current = x) ...
    else if (current.data < ...
    else if (current.data > ...
      ,,,,
```



x.next.prev = x

tree is sorted

program properties

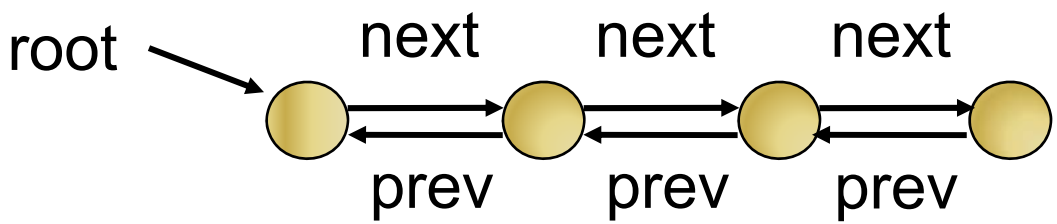
Program Verifier **Jahob**

program satisfies the properties ✓

error in program (or property) !

Data Structure Properties: Examples

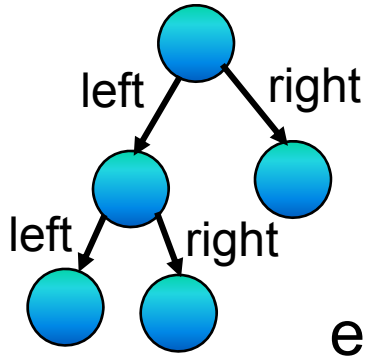
unbounded number of objects, dynamically allocated



x.next.prev == x

acyclicity: ~next^{+}(x,x)

shape not given by types,
may change over time

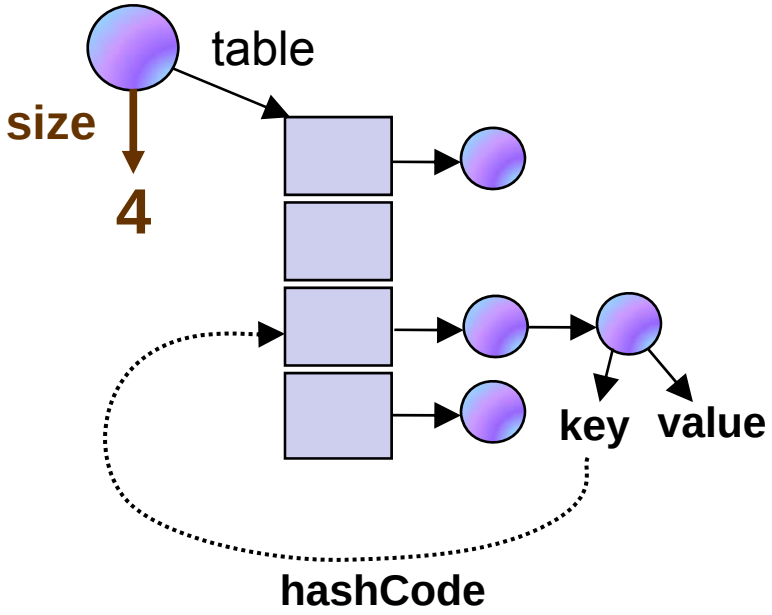


graph is a tree

elements are sorted

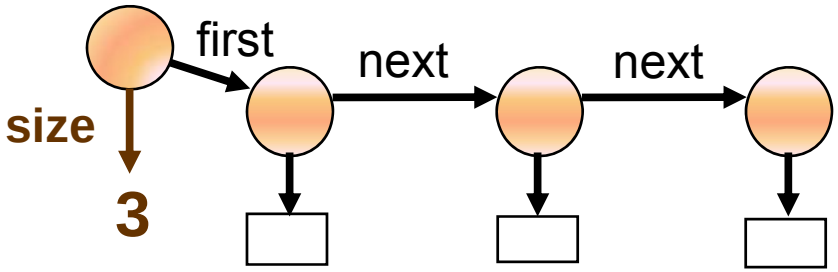
```
class Node {
    Node f1, f2;
}
```

Data Structure Properties: Examples



hash table properties:

node is stored in the bucket given by the hash of node's key



numerical constraints:

value of size field is number of stored objects

size = |{x. next*(first,x)}|

Data Structure Verification using Jahob

Verified properties of

- hash table
- space subdivision tree
- linked list
- array-based list
- priority heap

More information:

http://JavaVerification.org

Karen Zee, MIT

Martin C. Rinard, MIT

Full Functional Verification of Linked Data Structures

*ACM Conf. **P**rog. **L**anguage **D**esign and **I**mplementation*'08

Jahob Statically Enforces Contracts

ArrayList documentation from http://java.sun.com :

Method Summary	
void	**add**(int index, Object element) **Inserts the specified element at the specified position in this list.**
boolean	**add**(Object element) **Appends the specified element to the end of this list.**

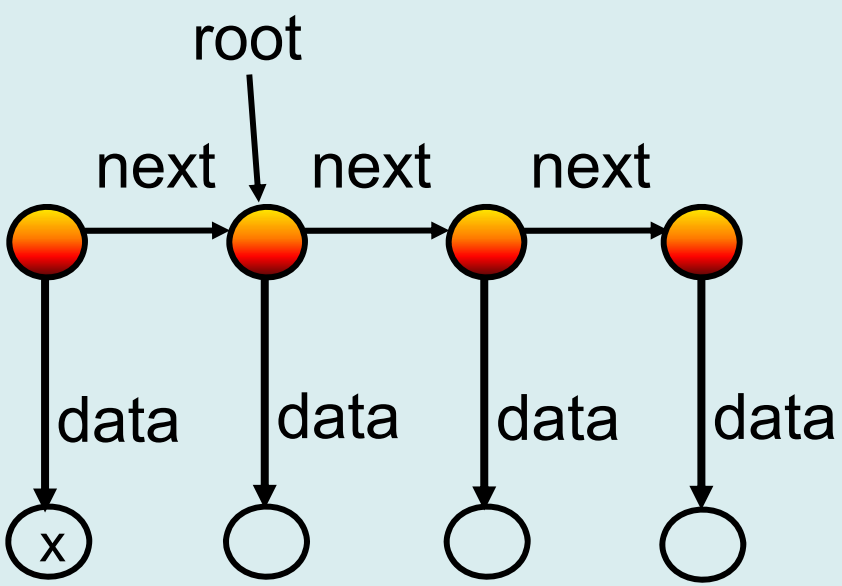
ArrayList verified contract from http://JavaVerification.org :

void	**add**(int index, Object element) **content = {(i,e). (i,e) : old content ∧ i < index} ∪ {(index,element)}** **∪ {(i,e). (i-1,e) : old content ∧ index < i}**
boolean	**add**(Object element) **content = old content ∪ {(size,element)}**

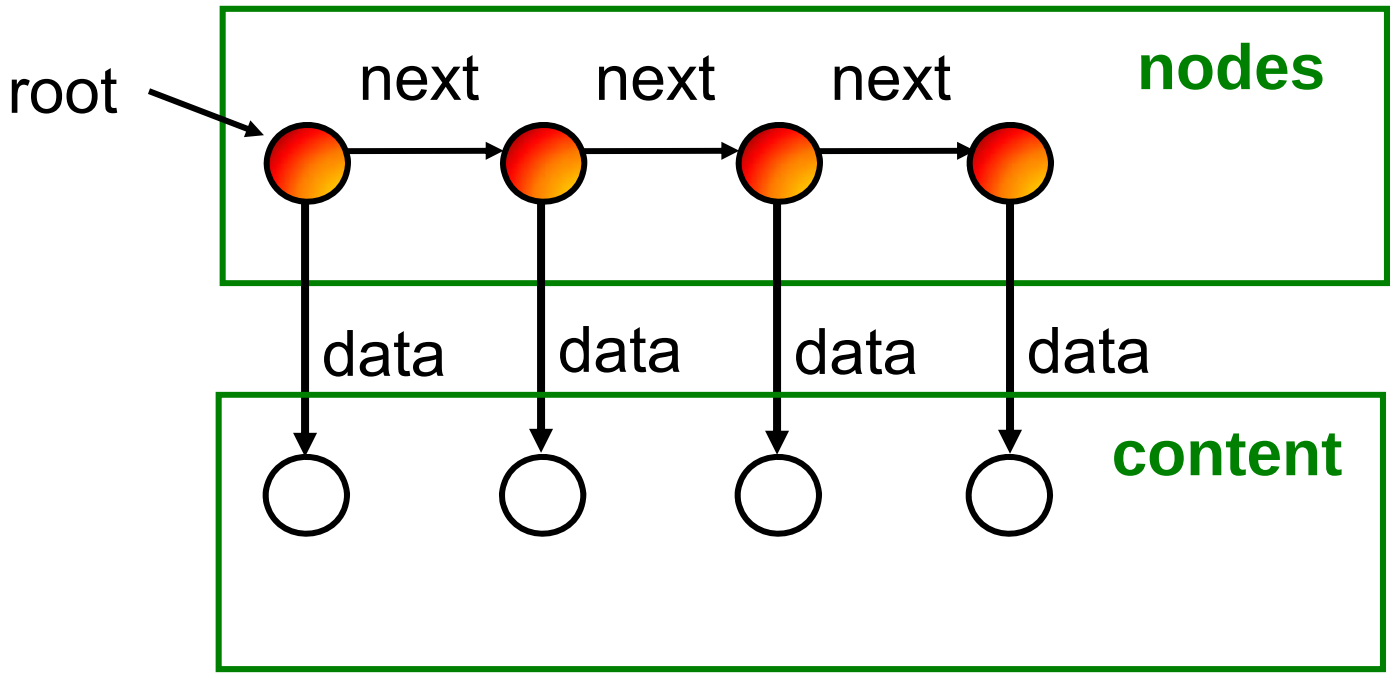
Linked List Implementation

```
class List {
  private List next;
  private Object data;
  private static List root;
  private static int size;

  public static void addNew(Object x) {
     List n1 = new List();
     n1.next = root;
     n1.data = x;
     root = n1;
     size = size + 1;
  }
}
```



Specifying Linked List in Jahob



Abstract the list with its content (data abstraction)

List.java Screenshot

specs as verified comments

public interface is simple

```
class List {
  private List next;
  private Object data;

  private static List root;
  private static int size;
  /*:
    private static ghost specvar nodes :: objset;
    public static ghost specvar content :: objset;

    invariant nodesDef: "nodes = {n. n ≠ null ∧ (root,n) ∈ {(u,
    invariant contentDef: "content = {x. ∃ n. x = List.data n ∧

    invariant sizeInv: "size = cardinality content";
    invariant treeInv: "tree [List.next]";
    invariant rootInv: "root ≠ null ⟶ (∀ n. List.next n ≠ root
    invariant nodesAlloc: "nodes ⊆ Object.alloc";
    invariant contentAlloc: "content ⊆ Object.alloc";
   */

  public static void addNew(Object x)
  /*: requires "(x ∉ content)"
      modifies content
      ensures "content = old content ∪ {x}"
  */
  {
    List n1 = new List();
    n1.next = root;
    n1.data = x;
```

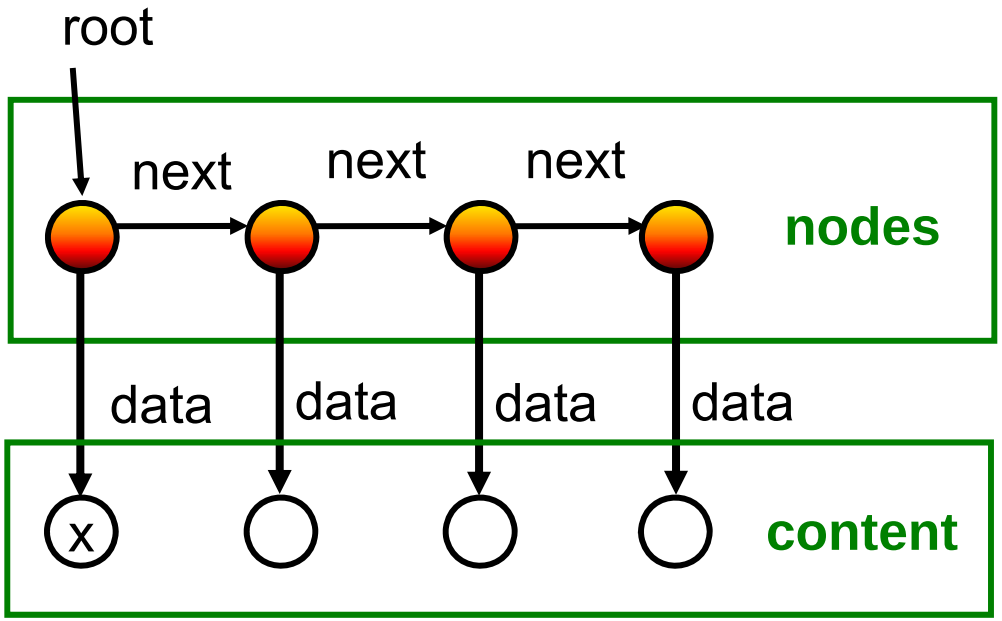
Isabelle/HOL as formula language

`addNew` Verification Condition for size

next0, data0, size0, nodes0, content0

```
List n1 = new List();
n1.next = root;
n1.data = x;
root = n1;
size = size + 1;
```

//: nodes = nodes ∪ {n1}



next, data, size, nodes, content

next=next0[n1:=root0] ∧ data=data0[n1:=x] ∧ … →

MONA: nodes = nodes0 ∪ {n1}

SPASS: content = content0 ∪ {x}

BAPA: |content| = |content0| + 1

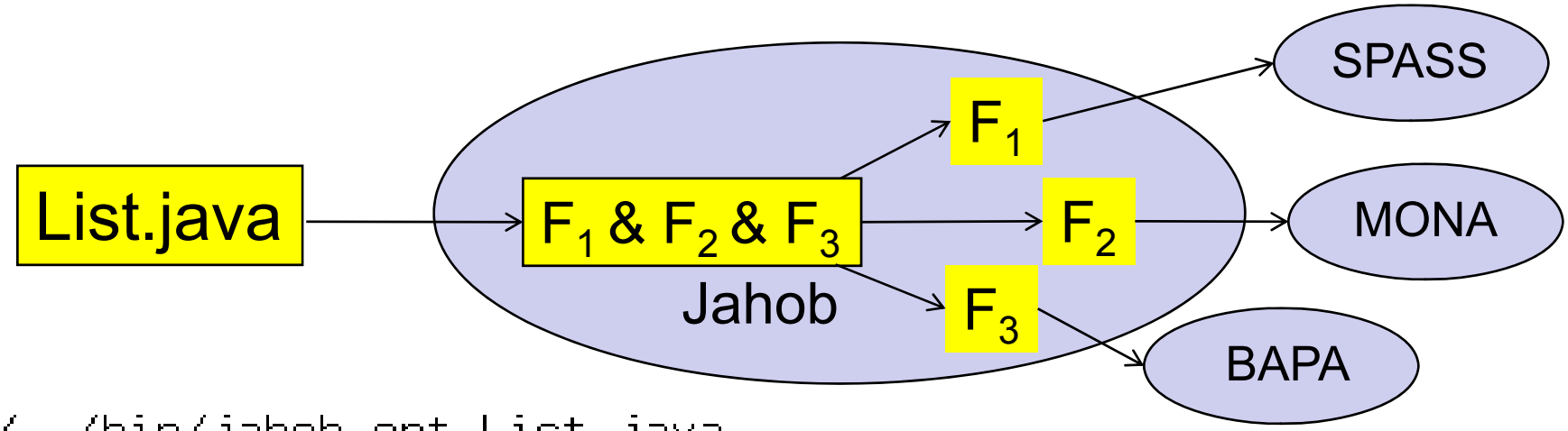
set vars generalize purification in SMT

∴ |{data(n) | (n1,n) ∈ next*} | =

|{data0(n) | (root0,n) ∈ next0*}| + 1

recently: decidability of combination

Verifying the addNew Method



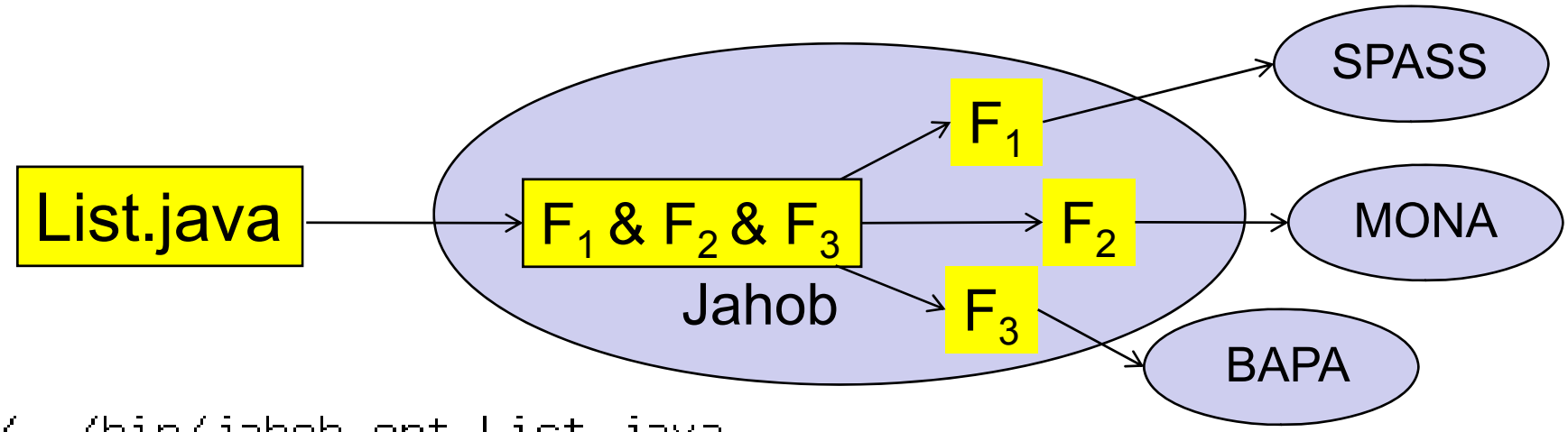

```
../../bin/jahob.opt List.java
-method List.addNew -usedp spass mona bapa
```

Verification steps

- generate verification condition (VC) in HOL stating "The program satisfies its specification"
- split VC into a conjunction of smaller formulas F_i
- *approximate* each F_i with stronger F'_i in HOL subset

prove each F'_i conjunct w/ SPASS,MONA,BAPA

Verifying the addNew Method




```
../../bin/jahob.opt List.java
-method List.addNew -usedp spass mona bapa

[List.addNew:..s!.....xx...sx!....xx...s!....xx.......
==========================================================
Built-in validity checker proved 2 sequents during splitting.
SPASS proved 5 out of 8 sequents. Total time : 0.2 s
MONA proved 2 out of 3 sequents. Total time : 0.7 s
BAPA proved 1 out of 1 sequents. Total time : 0.0 s
==========================================================
A total of 10 sequents out of 10 proved.
:List.addNew]
O=== Verification SUCCEEDED.
```

Jahob Verifier

Verifies programs in Java subset (input - valid Java)

Specifications written in subset of Isabelle/HOL

Jahob proves

- data structure preconditions, postconditions (can relate to ‘old’ values in pre)
- data structure invariants
- absence of run-time errors

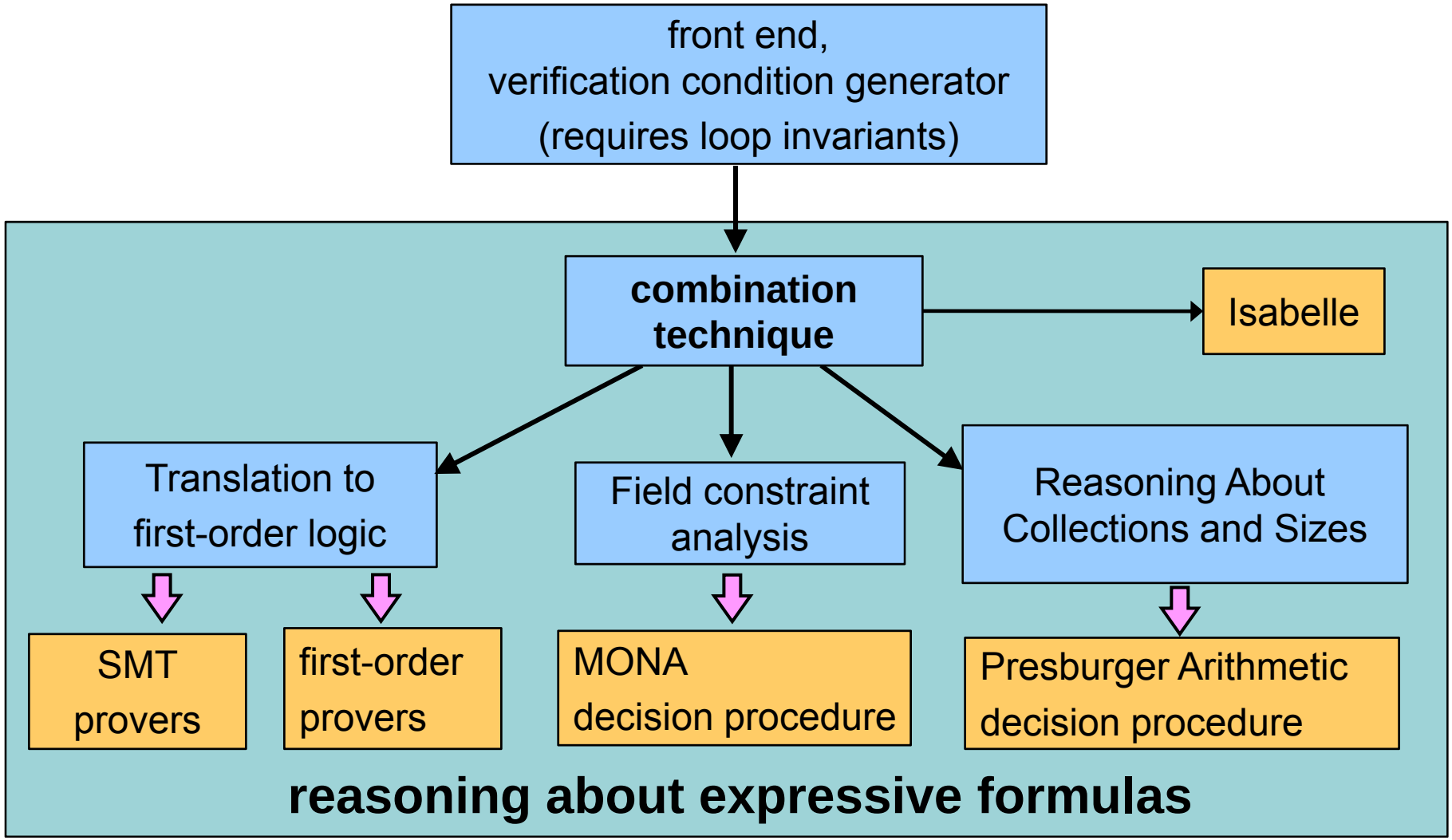
Observation:

automation in such verification is limited by

automated reasoning techniques

for proving verification-condition formulas

Jahob Verifier Overview



Jahob's combination method

1) Split verification condition(vc) into conjuncts

vc is an implication, e.g. pre & code → post

Transform implication into conjunction of implications

$A \rightarrow G1 \,\&\, G2 \quad \Rightarrow \quad A \rightarrow G1,\ A \rightarrow G2$

$A \rightarrow (B \rightarrow G) \quad \Rightarrow \quad (A \,\&\, B) \rightarrow G$

$A \rightarrow \forall\, x.G \quad \Rightarrow \quad A \rightarrow G[x := x_{fresh}]$

2) Approximinate each conjunct w/ stronger formula in more tractable logic (recursive walk)

3) Prove each conjunct using a prover (try all provers in sequence or in parallel)

Range of sound approximations

Worst: $a(F) = \text{False}$ **(useless)**

General idea of our approximations:

$a(F) = a^1(\text{simplify}(F))$

$a^p(F_1 \wedge F_2) = a^p(F_1) \wedge a^p(F_2)$

$a^p(F_1 \vee F_2) = a^p(F_1) \vee a^p(F_2)$

$a^p(\neg F) = \neg\, a^{\neg p}(F)$

$a^p(\text{goodF}) =$ *translation of* goodF

$a^1(\text{badF}) = \text{False}$

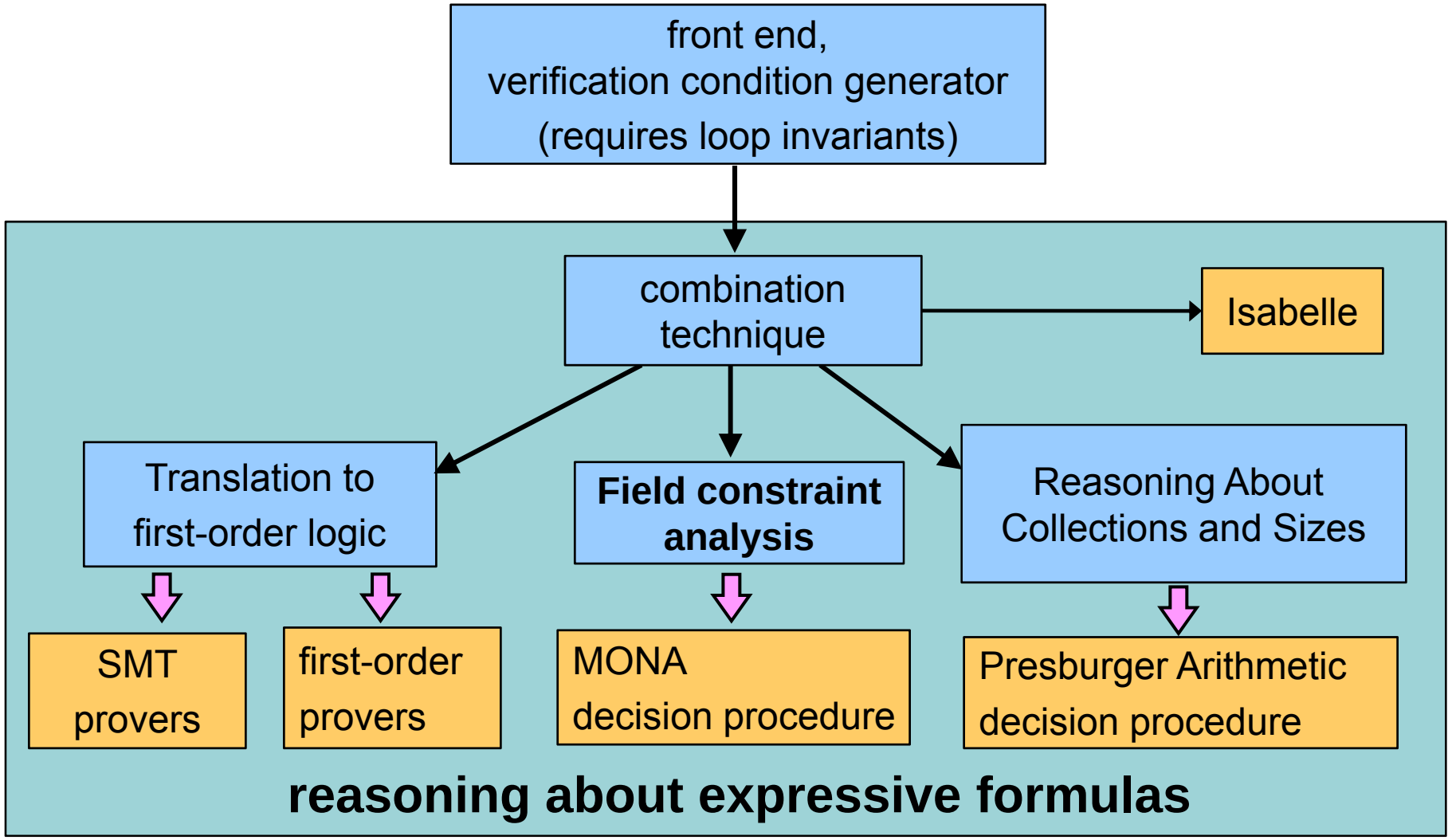
$a^0(\text{badF}) = \text{True}$

Best: $a(F) =$ if "F is valid" then True else False **(impossible)**

Properties of this method

- Properties of splitting
  - introduces only quadratic blowup
  - exploits vc structure: many paths, invariants
  - parallelization opportunities
- Property of approximation
  - unsupported assumptions thrown away
  - sound, incomplete
- Lemmas about sets enable combination
  - user supplied or generated by static analysis

Jahob Verifier Overview



Field Constraint Analysis

Approximation of HOL with WS2S formulas

Automates reasoning about reachability

Effective on formulas of the form:

(tree[f1,f2] &

ALL x. h1(x)=y →F1(x,y) &

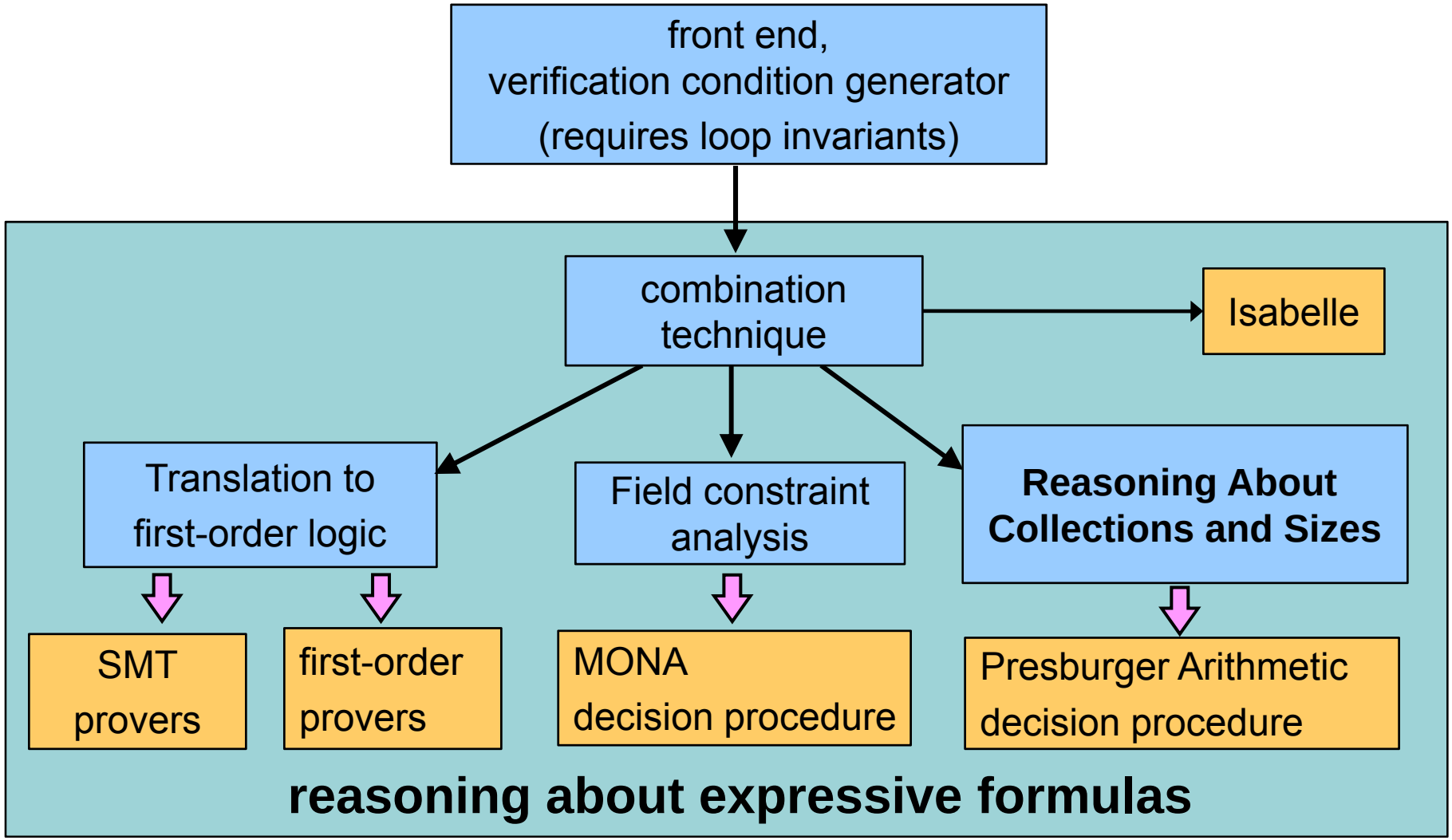
ALL x. h2(x)=y →F2(x,y)) → G(f1,f2,h1,h2)

f1,f2 – backbone *h1,h2 – derived*

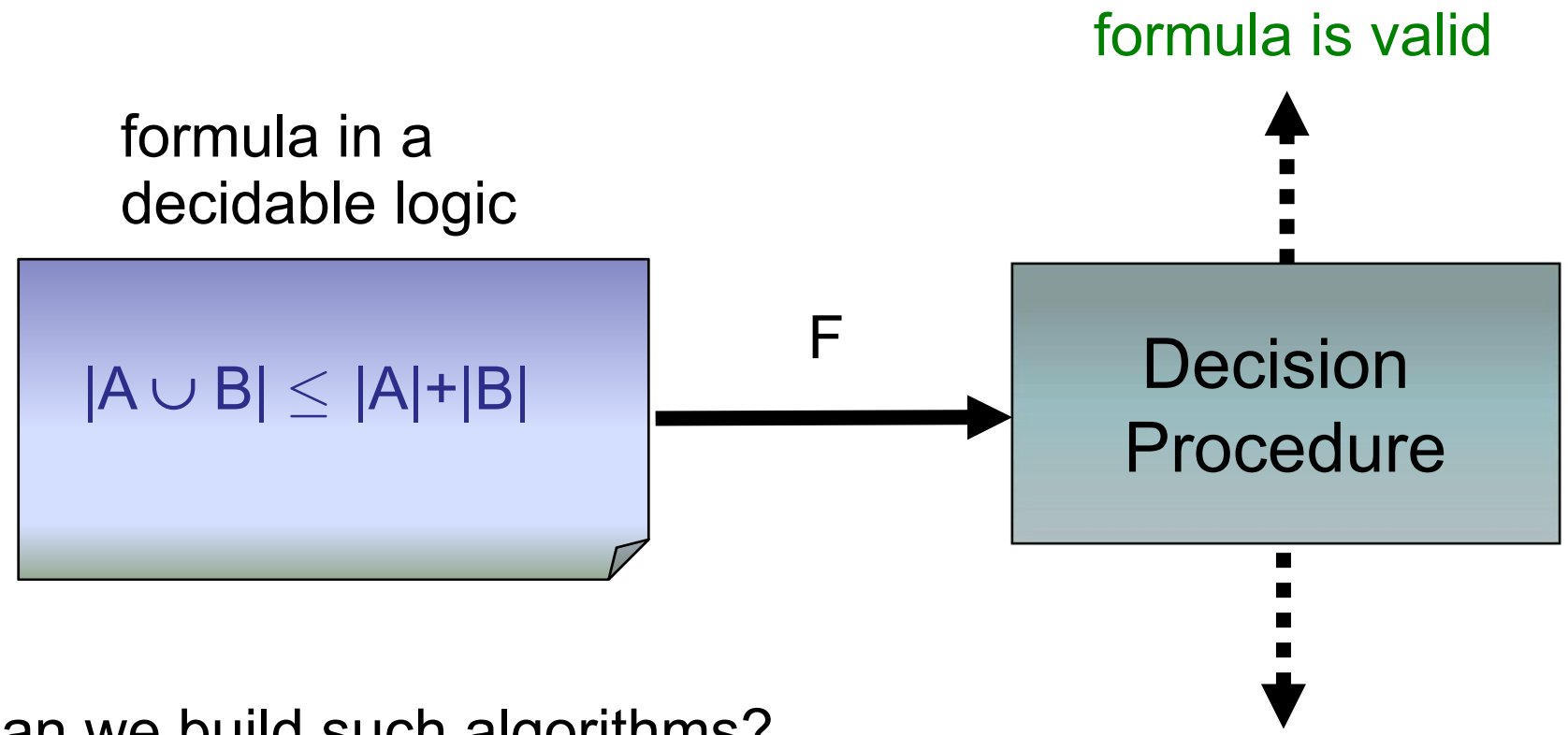
Such form enforced by class invariants

Soundness and (often) completeness

Jahob Verifier Overview



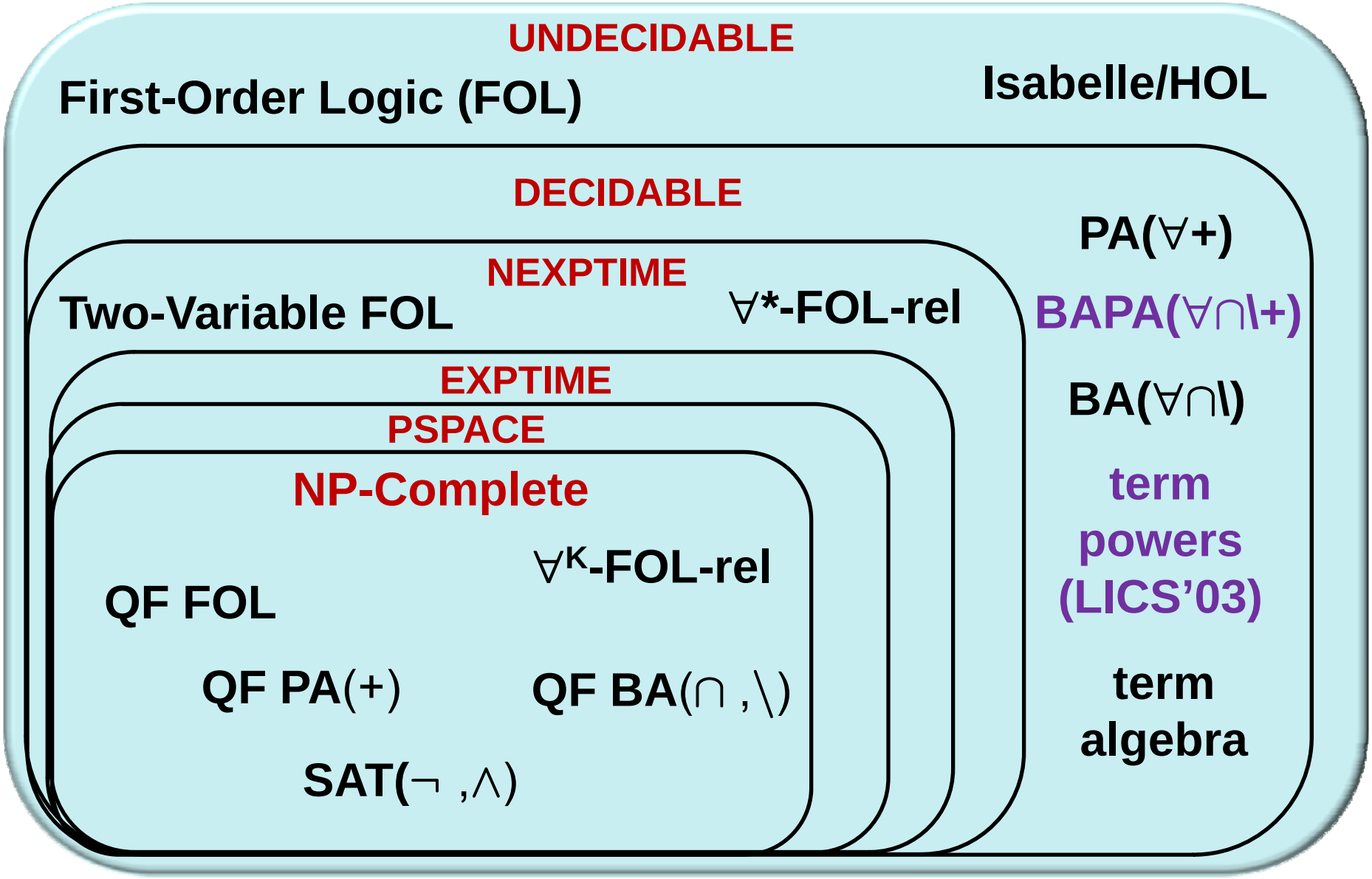
New Decision Procedures



Can we build such algorithms?

- useful implementations
  (used within larger systems)
- worst-case complexity limitations

Complexity Map of Some Logics



Boolean Algebra with Presburger Arithmetic

$$S ::= V \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S_1 \setminus S_2$$
$$T ::= k \mid C \mid T_1 + T_2 \mid T_1 - T_2 \mid C \cdot T \mid \mathbf{card}(S)$$
$$A ::= S_1 = S_2 \mid S_1 \subseteq S_2 \mid T_1 = T_2 \mid T_1 < T_2$$
$$F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists S.F \mid \exists k.F$$

Essence of decidability: Feferman, Vaught 1959

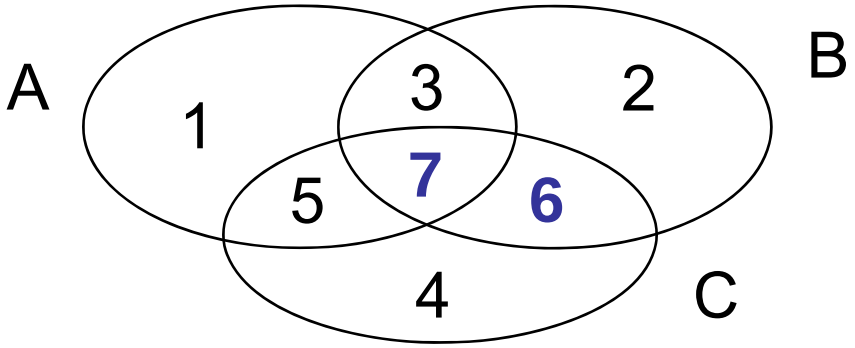
Our results

- first implementation for BAPA (CADE'05)
- first, exact, complexity for full BAPA (JAR'06)
- polynomial-time fragments of QFBAPA (FOSSACS'07)
- first, exact, complexity for QFBAPA (CADE'07)
- generalize to multisets (VMCAI'08,CAV'08,CSL'08)

From BAPA to Linear Arithmetic

$\text{card}(A \cup B) = p \quad \wedge \quad \text{card}(B \cap C) = q$

$x_1 + x_2 + x_3 + x_5 + x_6 + x_7 = p \quad \wedge \quad \mathbf{x_6 + x_7 = q}$

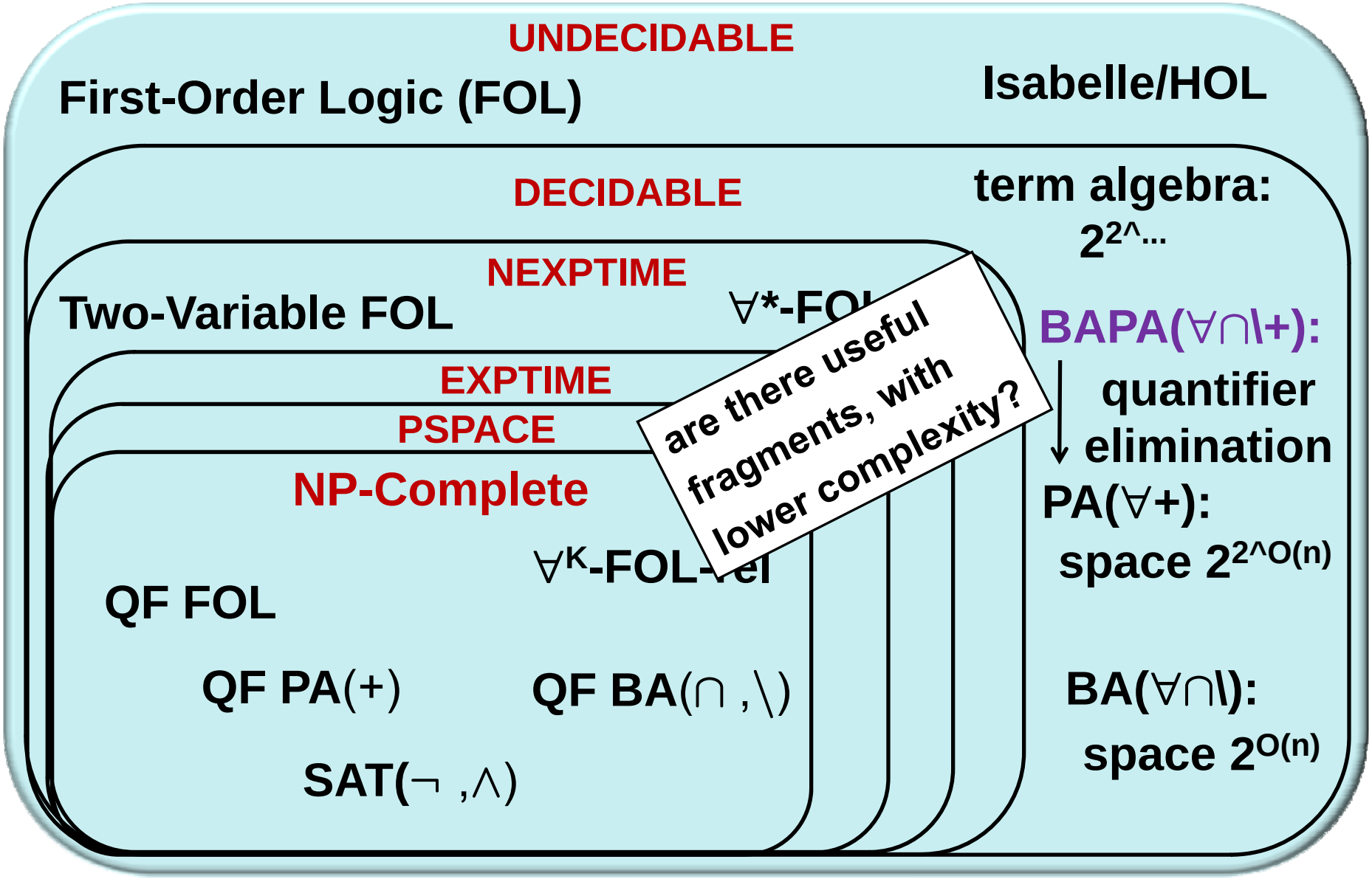


Deciding BAPA: set vars ➔ int vars

- generate equisatisfiable PA formula
- exponentially many variables (for each Venn region)

Works for quantified and quantifier free case

Exact Complexity of BAPA (JAR'06)



Quantifier-Free BAPA

$$S ::= V \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S_1 \setminus S_2$$
$$T ::= k \mid C \mid T_1 + T_2 \mid T_1 - T_2 \mid C \cdot T \mid \mathbf{card}(S)$$
$$A ::= S_1 = S_2 \mid S_1 \subseteq S_2 \mid T_1 = T_2 \mid T_1 < T_2$$
$$F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F$$

Why is quantifier-free fragment useful?

BAPA has quantifier elimination
(algorithm: formula → equivalent Q.F. formula)

We can express same relations using quantifier-free formulas

Verification conditions are often quantifier-free

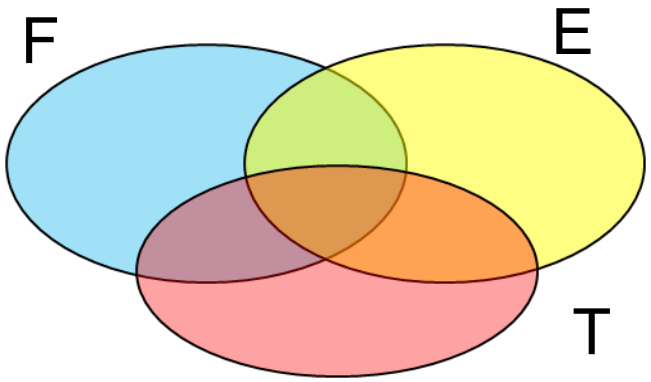
$$x \notin content0 \wedge content = content0 \cup \{x\} \rightarrow$$
$$cardinality(content) = cardinality(content0) + 1$$

Puzzles with Sets and Cardinalities

8 students applied to a PhD program. Each of them speaks French or English and 3/4 of them speak both French and English. Among those speaking French, there are twice as many of those not in top 5% of their master's studies as those that do not speak English.

The number of students who speak French but not English is the same as number of those who speak English but not French.

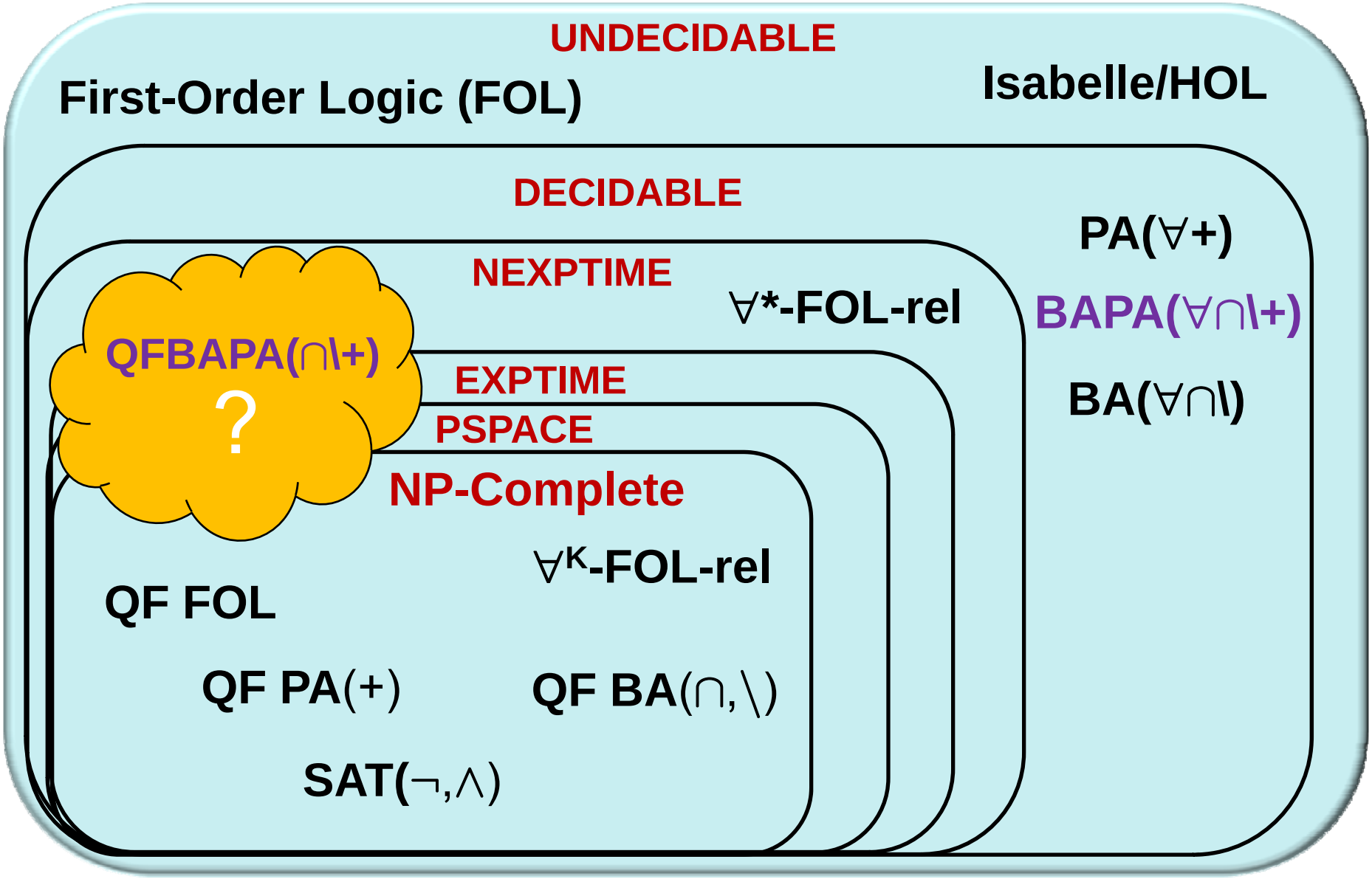
Is this situation possible? If so, how many students speak French, English and are in top 5%?



card(F∩E∩T)>0?

$\text{card}(\mathcal{U})=8 \wedge F \cup E=\mathcal{U} \wedge 4\text{card}(F \cap T)=3\text{card}(\mathcal{U}) \wedge$
$2\text{card}(F \setminus T) = \text{card}(F \setminus E) \wedge \text{card}(F \setminus E)=\text{card}(E \setminus F)$

Exact Complexity of QFBAPA ?



Underlying Integer Programming Problem

Integer linear programming problem: for non-negative x_i

$$x_1 + x_2 + x_3 + x_5 + x_6 + x_7 = p$$

$$\ldots$$

$$x_6 + x_7 = q$$

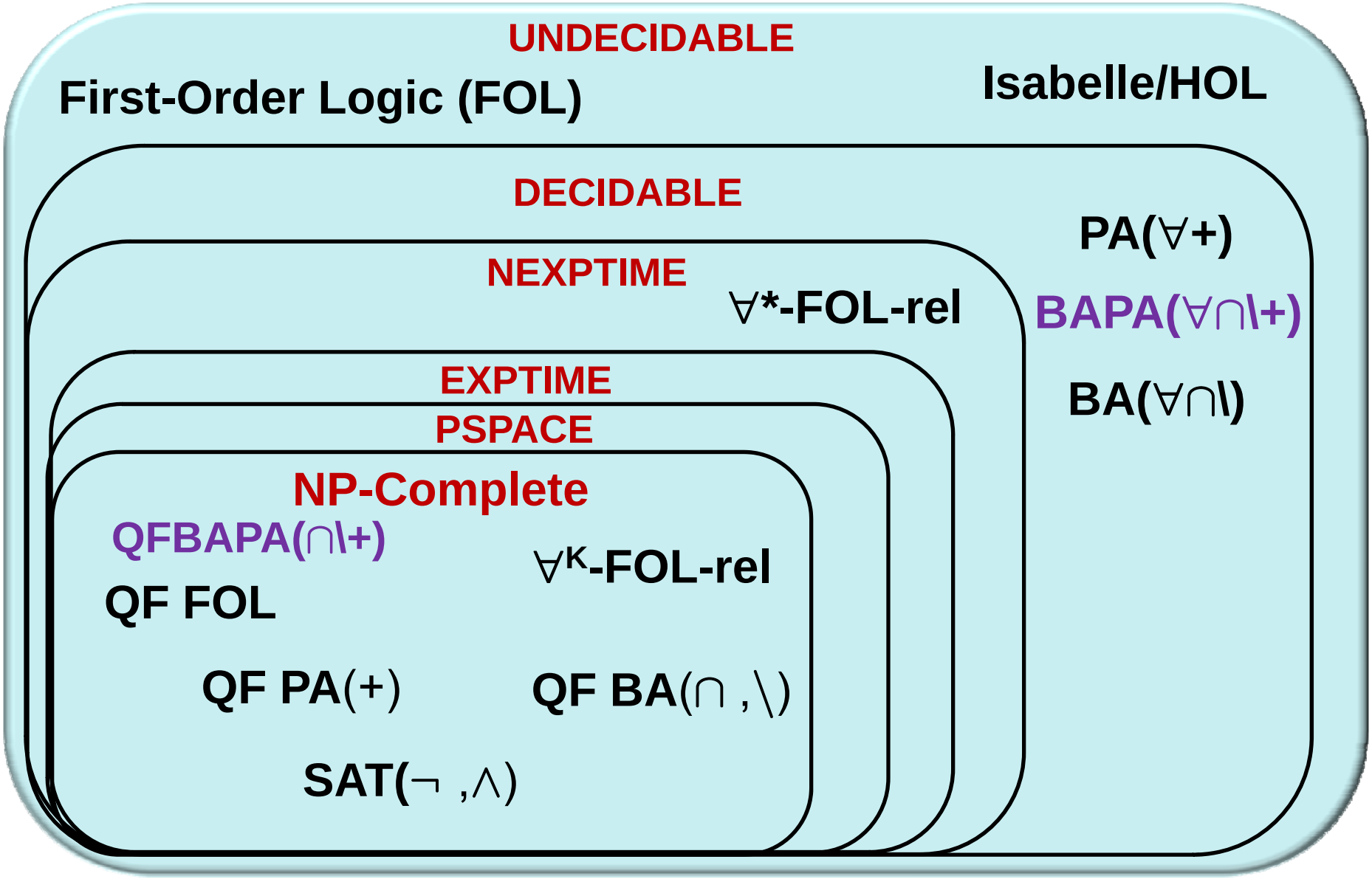
n equations

2^n variables

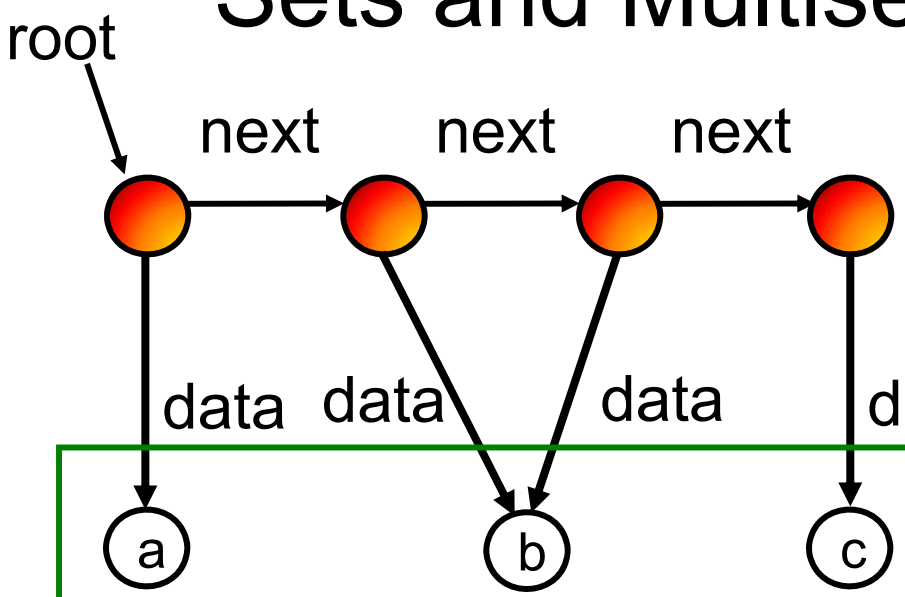
Are there **sparse** solutions where $O(n^k)$ variables are non-zero?

for reals	- yes, matrix rank is O(n)
for non-negative reals	- yes, theory of LP
for non-negative integers	**- Eisenbrand, Shmonin'06**

Exact Complexity of QFBAPA (CADE'07)



Sets and Multisets in Abstraction



content as set

content = {a,b,c}

U → {0,1}

card(content)=3

a b b c

mcontent as multiset

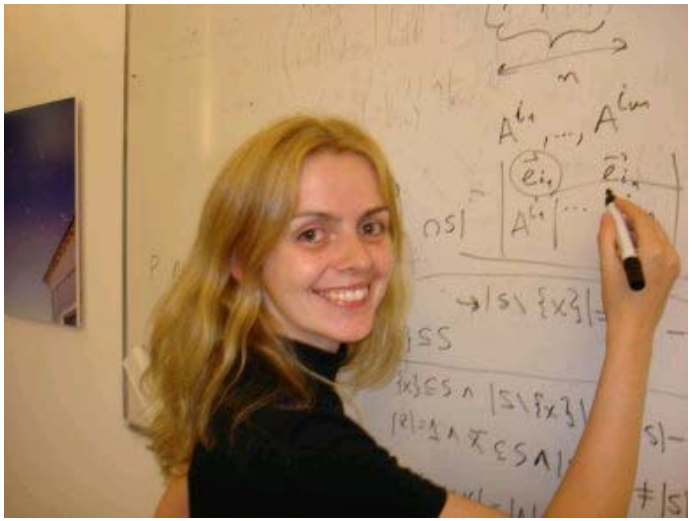
mcontent = {|a,b,b,c|}

U → {0,1,2,...}

card(mcontent)=4

C = C0 ⊎ {x} → card(C)=card(C0) + 1

Reasoning about Collections and Sizes



work with

Ruzica Piskac, 2nd year
PhD student in my group

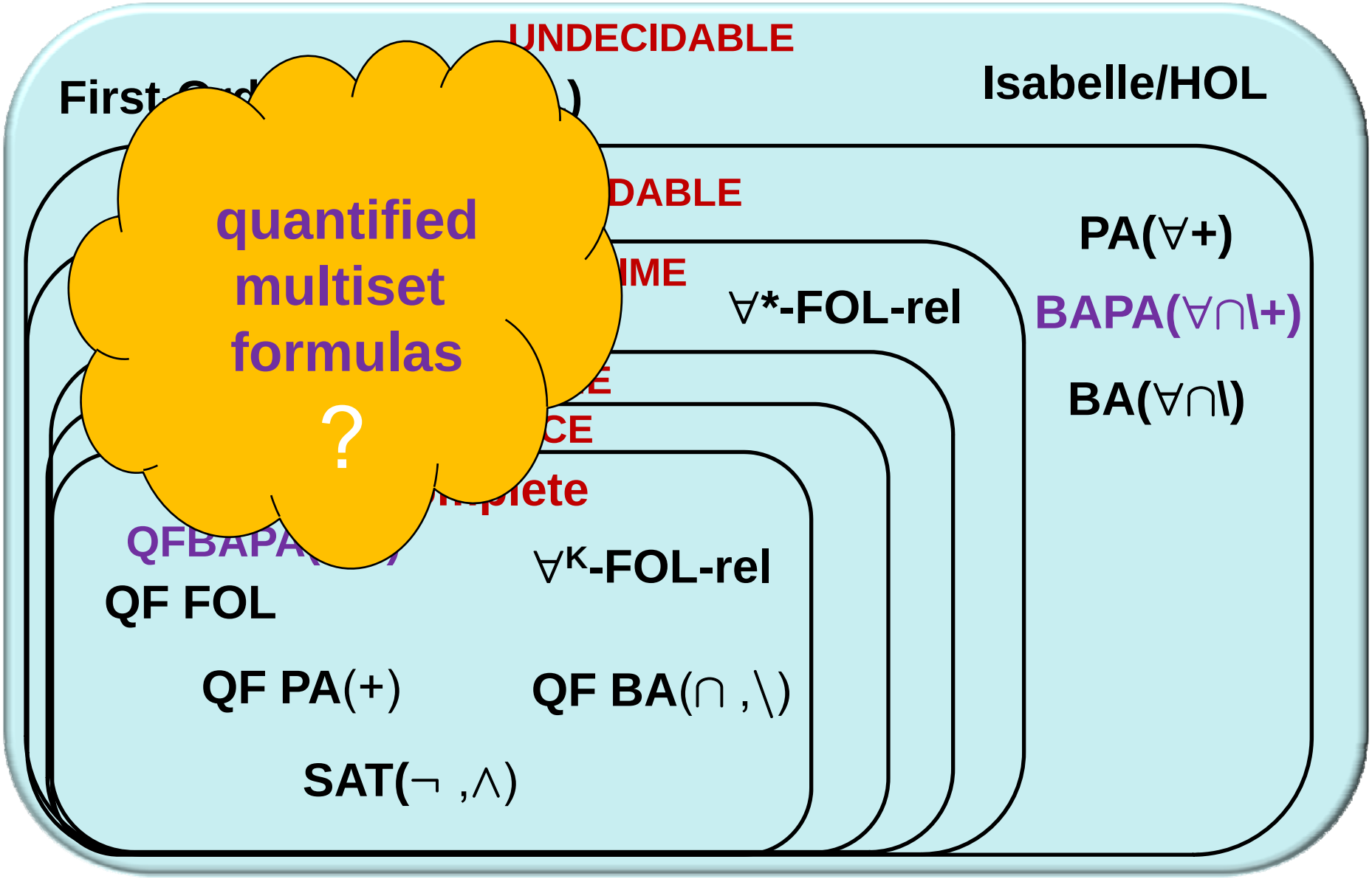
generalized from sets to multisets

Decision Procedures for Multisets with Cardinality Constraints,
Verification, Model Checking, and Abstract Interpretation, 2008

Linear Arithmetic with Stars, *Computer Aided Verification*, 2008

Fractional Collections with Cardinality Bounds and Mixed Integer Linear Arithmetic with Stars,
Computer Science Logic, 2008

Complexity of Quantified Multisets?



Complexity of Quantified Multisets

- Addition

$$x + y = z \iff \exists a, b.\ |a| = x \ \wedge\ |b| = y \ \wedge\ |a \uplus b| = z$$

- Multiplication

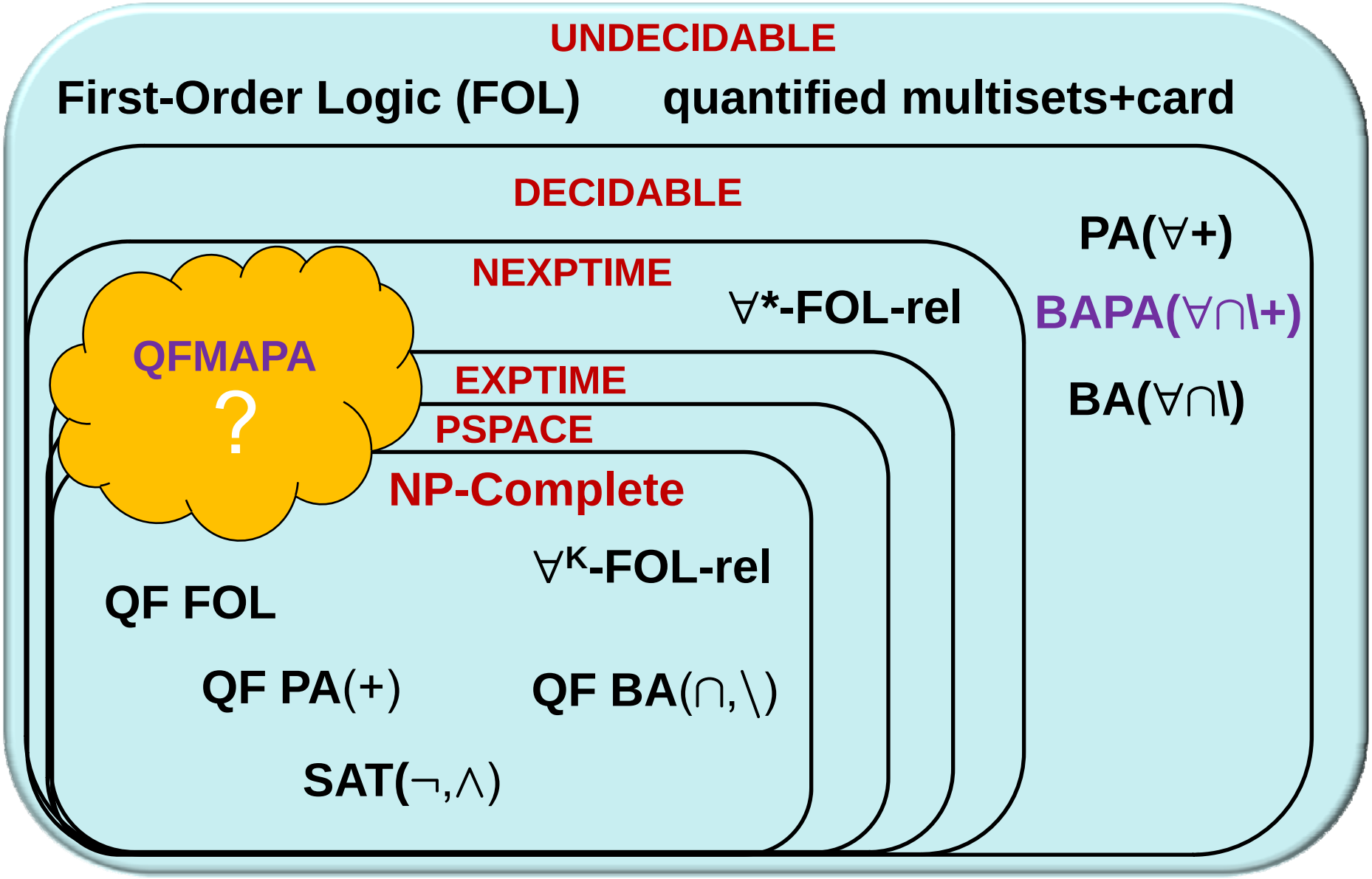
$$x \cdot y = z \iff \exists p.\ z = |p| \ \wedge\ x = |\mathsf{setof}(p)| \ \wedge\ (\forall m.\ |m| = z \wedge |\mathsf{setof}(m)| = 1 \wedge \mathsf{setof}(m) \subseteq p \implies |m \cap p| = y)$$

Hilbert's Tenth Problem

Given a polynomial Diophantine equation with integer coefficient, is there a general algorithm for deciding whether the equation has a solution in integers.

UNDECIDABLE

Exact Complexity of QFMAPA ?



From Collections to Stars

Check satisfiability of

$|x|=1 \wedge L_1=L \uplus x \wedge |L_1| \neq |L|+|x|$

(negation of a verification condition)

Normal form:

$(1,k,k_1) = \sum\{(x(e),L(e),L_1(e))| e \in E\} \wedge$

$\forall e.\ L_1(e)=L(e)+x(e) \wedge \ k_1 \neq k+1$

This is *equisatisfiable* with

$(1,k,k_1) \in \{(x,L,L_1)| L_1=L+x\}^* \wedge \ k_1 \neq k+1$

(Such transformation works in general.)

Integer Linear Arithmetic with Star

Decidability: each formula describes

semilinear set, i.e. union of sets of the form

$\{a\} + \{b_1,\ldots,b_n\}^*$

Semilinear sets are closed under *

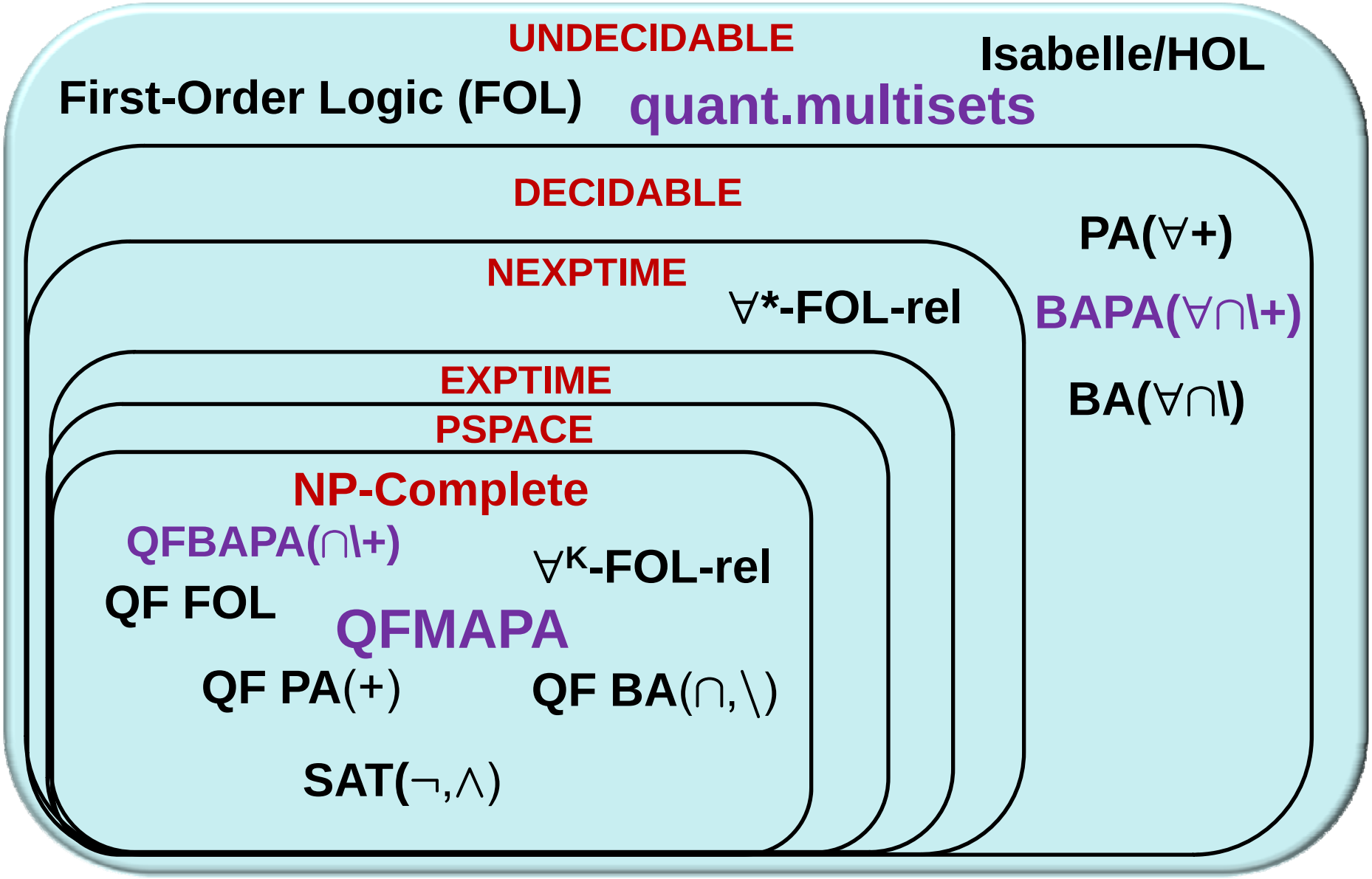
Computing semilinear sets: NEXPTIME

Using sparseness, and bounds on a,b

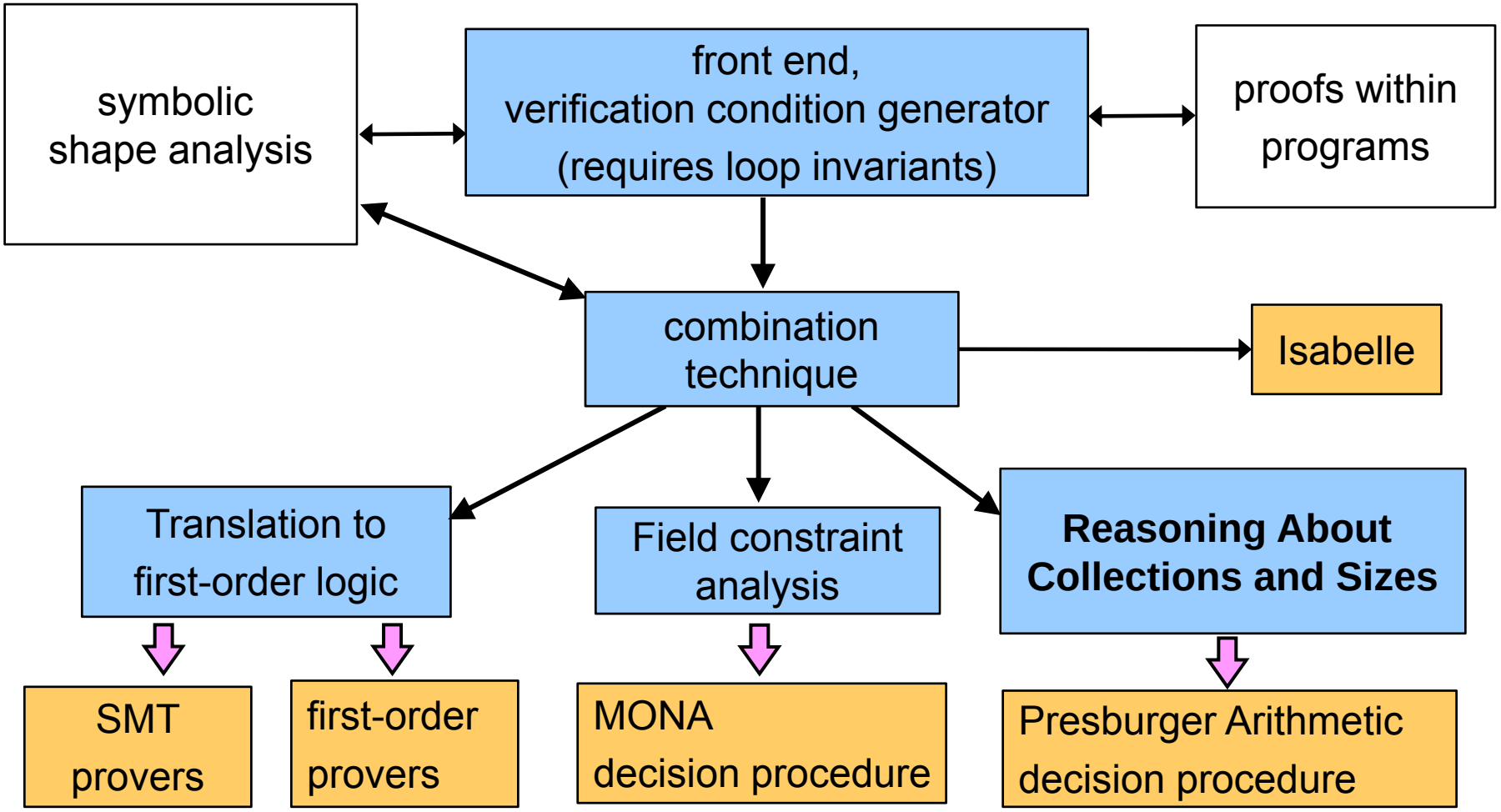
→ NP (CAV' 2008)

Application to transition system reachability

Summary of Decision Procedure Results



Jahob Verifier Overview

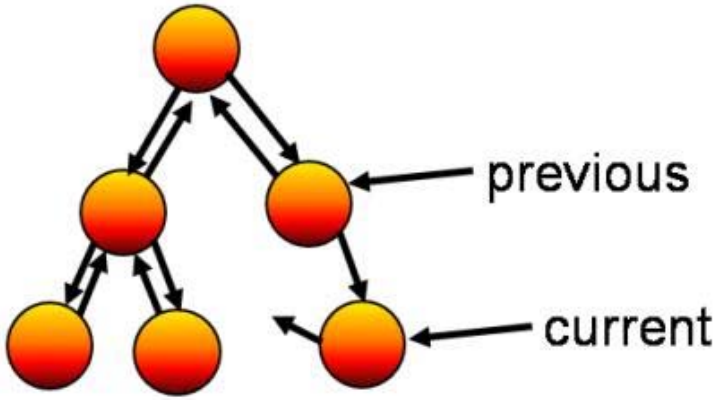


Symbolic Shape Analysis

Automatically infers
quantified loop Invariants for
data structure implementations

Thomas Wies

Freiburg
(now EPFL)

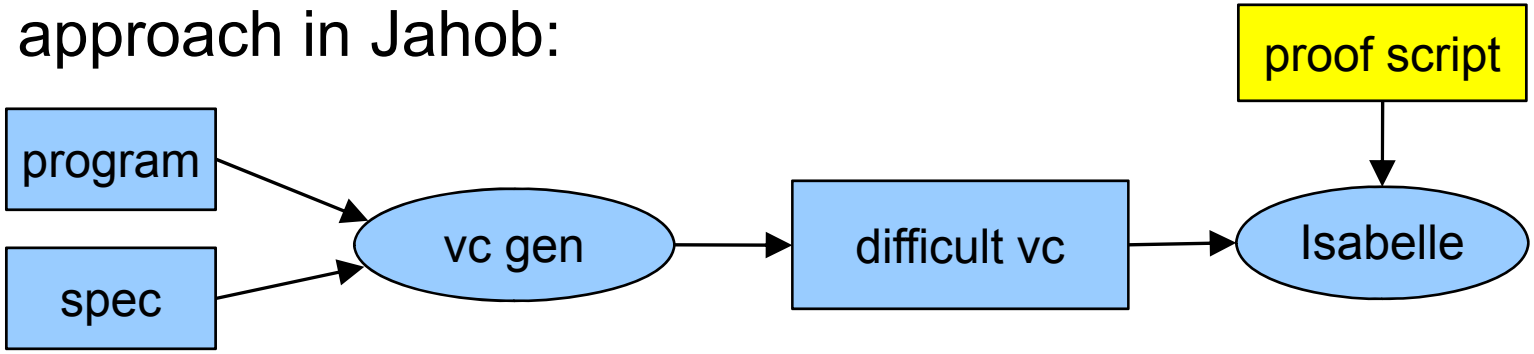


Andreas Podelski

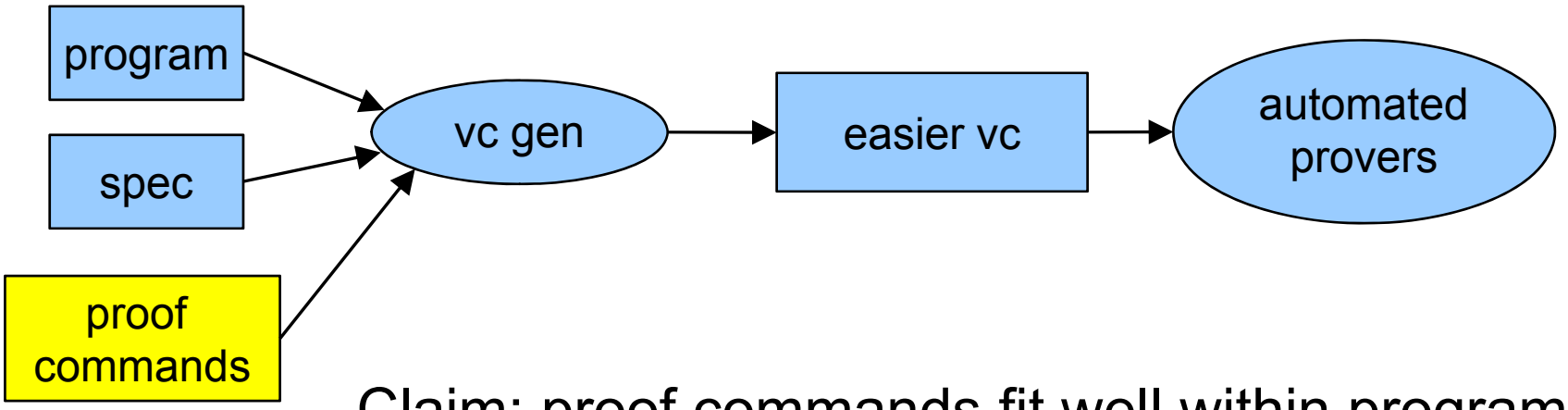
Freiburg

Proofs within Programs in Jahob

one approach in Jahob:



alternative approach:



Claim: proof commands fit well within programs

Guarded Commands and wp

Basis for verification condition generation

programs + spec → guarded commands

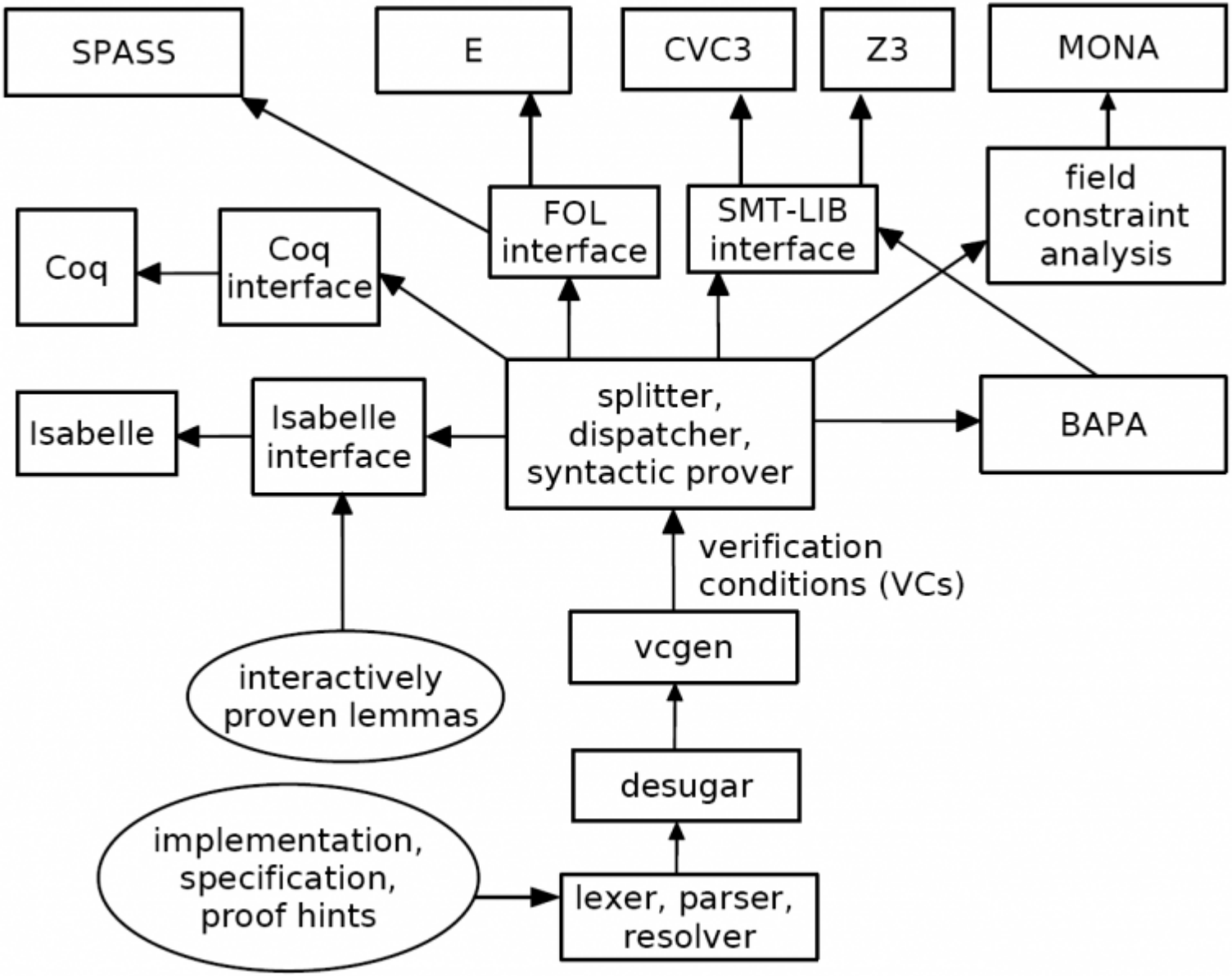
Command c:	wp(c,G):
assume F	F → G
assert F	F & G
havoc x	ALL x. G
[]	&

Proof commands corresponding to FOL

(intro and elimin for *binders, and,or,not*)

Soundness, completeness for FOL

Jahob Verifier



Summary: http://javaverification.org

Jahob: a (data structure) verification system

Verified: lists, trees, hash tables, queues, clients

Proving validity of expressive formulas (HOL subset) by combining multiple reasoning techniques

Reasoning about collections with cardinality

Proofs within programs

Laboratory for Automated Reasoning

http://lara.epfl.ch

EPFL School of Computer and Communication Sciences

http://ic.epfl.ch

EPFL PhD Program : http://phd.epfl.ch/edic