http://richmodels.org

Towards a Rich Model Toolkit

An Infrastructure for Reliable Computer Systems

The objective of the Action is making automated reasoning techniques and tools applicable to a wider range of problems, as well as making them easier to use by researchers, software developers, hardware designers, and information system users and developers.

Viktor Kuncak

Lab for Automated Reasoning and Analysis

http://lara.epfl.ch

EPFL

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

COST Action IC0901

Application area: reliable computer systems

Technique: **automated reasoning** (broadly)

- e.g. theorem proving, verification, synthesis

Nature of activities

- collaboration on existing national research
- framework to obtain further national and international funds
- intrinsic results, *e.g. common formats*

Forms of activities

1) meetings 2) mutual visits of researchers

Activities in 2010

1. This meeting, 28-29 January 2010
2. Synthesis, Verification and Analysis of Rich Models
   http://richmodels.org/svarm
   – at FLOC, **Edinburgh** July 20-21 2010, collocated with IJCAR(CADE+) and CAV (also there: LICS, ITP,RTA,SAT,CSF,ICLP)
   – invited speaker: Natarajan Shankar
3. Meeting in **Lugano** (CH), with FMCAD
   – Significant hardware verification audience
   – Analysis and Synthesis

Europe-wide initiative

Country	MC Member
Austria (MC Member)	Professor Roderick BLOEM
Austria (MC Member)	Professor Armin BIERE
Czech Republic (MC Member)	Dr Stefan RATSCHAN
Czech Republic (MC Member)	Dr Tomas VOJNAR
Denmark (MC Member)	Professor Peter SESTOFT
Denmark (MC Member)	Professor Lars BIRKEDAL
Denmark (MC Substitute Member)	Professor Peter SCHNEIDER-KAMP
Estonia (MC Member)	Dr Jaan RAIK
Finland (MC Member)	Professor Ilkka NIEMELA
Finland (MC Member)	Professor Ivan PORRES
Finland (MC Substitute Member)	Professor Keijo HELJANKO
France (MC Member)	Dr Tayssir TOUILI
France (MC Member)	Dr Barbara JOBSTMANN
Germany (MC Member)	Professor Tobias NIPKOW
Germany (MC Member)	Professor Rupak MAJUMDAR
Germany (MC Substitute Member)	Dr Andrey RYBALCHENKO
Israel (MC Member)	Professor Alexander RABINOVICH
Israel (MC Member)	Dr Eran YAHAV
Italy (MC Member)	Professor Maria Paola BONACINA
Norway (MC Member)	Professor Marc BEZEM
Poland (MC Member)	Professor Leszek PACHOLSKI
Romania (MC Member)	Dr Gabriel ISTRATE
Romania (MC Member)	Dr Marius MINEA
Serbia (MC Member)	Professor Silvia GHILEZAN
Serbia (MC Member)	Dr Predrag JANICIC
Slovenia (MC Member)	Professor Denis TRCEK
Slovenia (MC Substitute Member)	Mr Iztok STARC (Pending)
Spain (MC Member)	Dr Enric RODRIGUEZ CARBONELL
Spain (MC Member)	Dr Cesar SANCHEZ
Sweden (MC Member)	Professor Reiner HAHNLE
Switzerland (MC Member)	Professor Natasha SHARYGINA
United Kingdom (MC Member)	Dr Paul JACKSON
United Kingdom (MC Member)	Professor Ian HORROCKS
United Kingdom (MC Substitute Member)	Dr Philipp RUEMMER
United Kingdom (MC Substitute Member)	Dr Radu CALINESCU

Work Groups

1. **Rich Model Language**
   Design, Benchmarks (a unifying activity)
   Chair: Tobias Nipkow; Vice Chair: Paul Jackson
2. **Decision Procedures** for
   Rich Model Language Fragments (key technique)
   Chair: Maria Paola Bonacina; V.Chair: Armin Biere
3. **Analysis of Executable Rich Models**
   large potential for practical impact
   Chair: Natasha Sharygina
4. **Synthesis from Rich Models**
   Chair: Barbara Jobstmann;V.Chair: Roderick Bloem

Rich *Model* Language (RML)

mathematical model ≈ specification (formula)

RML is a specification language

– rich ≈ great expressive power (higher-order logic)

– precise syntax (abstract and concrete)

– precise (and natural) semantics – agree, not invent

– a set of more tractable fragments

Rich Model Toolkit (RMT)

– set of tools that manipulate models in RML

– tools interoperate thanks to the common language

– benchmark suite drives further development

Example of verification of linked list

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

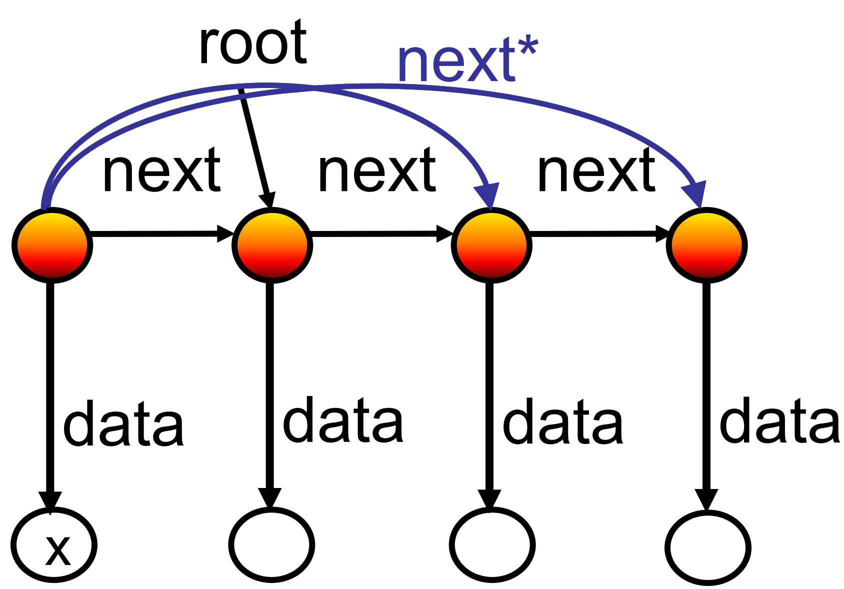
Set of stored objects:

{data(n). next*(root,n)}

ensure |{data(n). next*(root,n)}| = |old({data(n). next*(root,n)})| + 1

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

size: 4



Example Rich Constructs in Formulas

Sets and relations

- represent data structures in programs
- the language of mathematics

Transitive closure

- of un-interpreted relations: regions of program heap
- of transition systems: reachable states of system

Cardinality

- generalize quantifiers, e.g. card{x|P(x)}=1
- |A|=|B| - shows up naturally in many examples

Recursive definitions as part of language of formulas

- capture computable functions
- natural for both specification and constraint solving

Benefits of RML for Tools

- Tools that cover a wider range of problems
  - solve problems that combine multiple aspects
- Easier interfacing of tools
  - avoid differences that hamper interoperability
- Tools are more likely to be correct
  - semantics (though embedding into formulas) is explicit part of representation

Methodological Benefits of RMT

Some of current approaches to reasoning

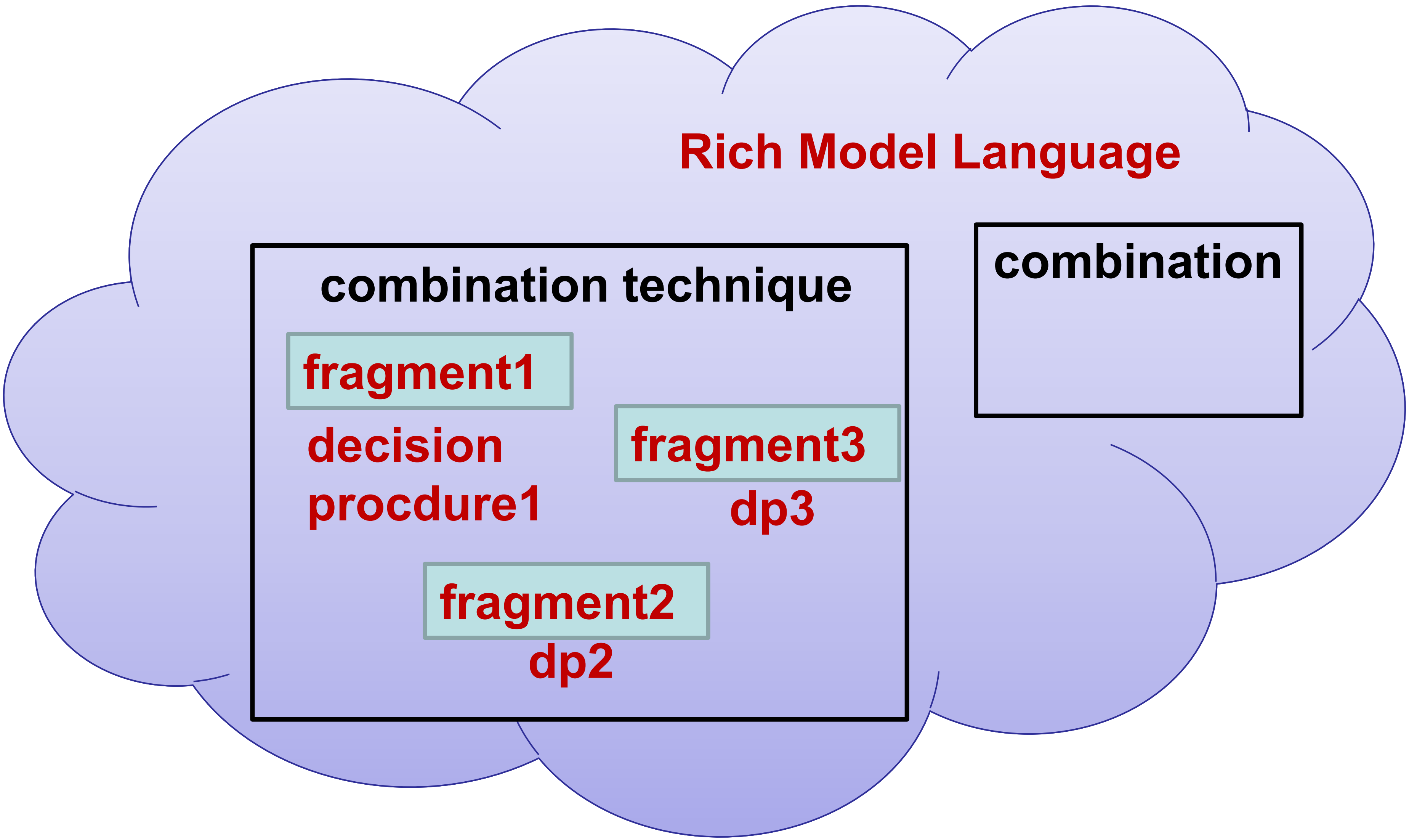
- provers for pure logic (FOL, pure HOL)
- decision procedures for individual theories

Current combinations of theories

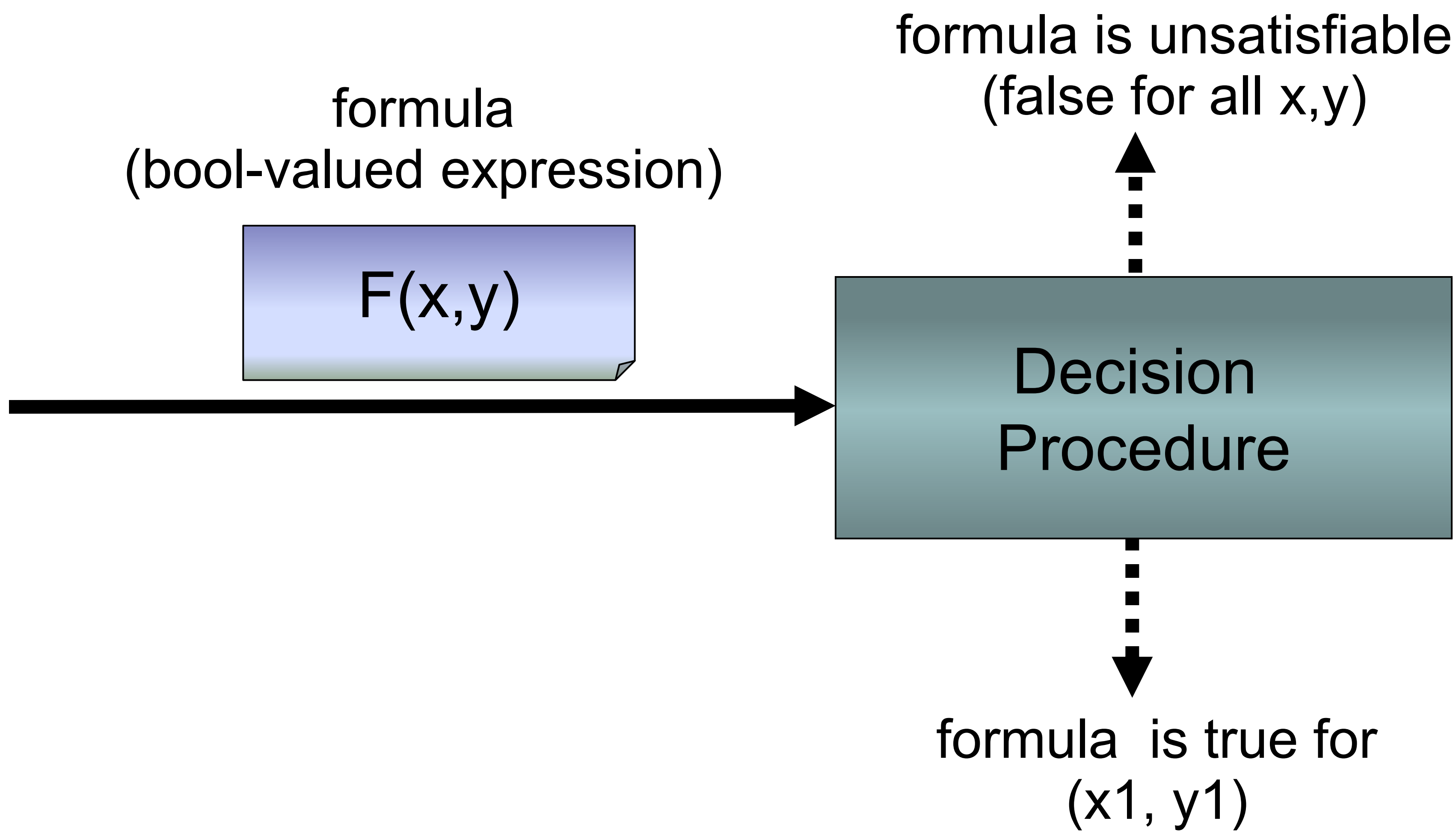
- specific **traditional theories** dominate (int, UF)
- almost exclusively **disjoint** combinations
- many sophisticated decidable logics left out, they do not fit the framework

Opportunity: consider richer language, combine sophisticated decision procedures

How to reason about rich models?



Decision Procedures for Fragments



Ways of defining RML fragments

Syntactic restriction examples – on grammar
- – no relations/functions/quantifier alt. / not / or
- – use only two variable names, guarded fragment

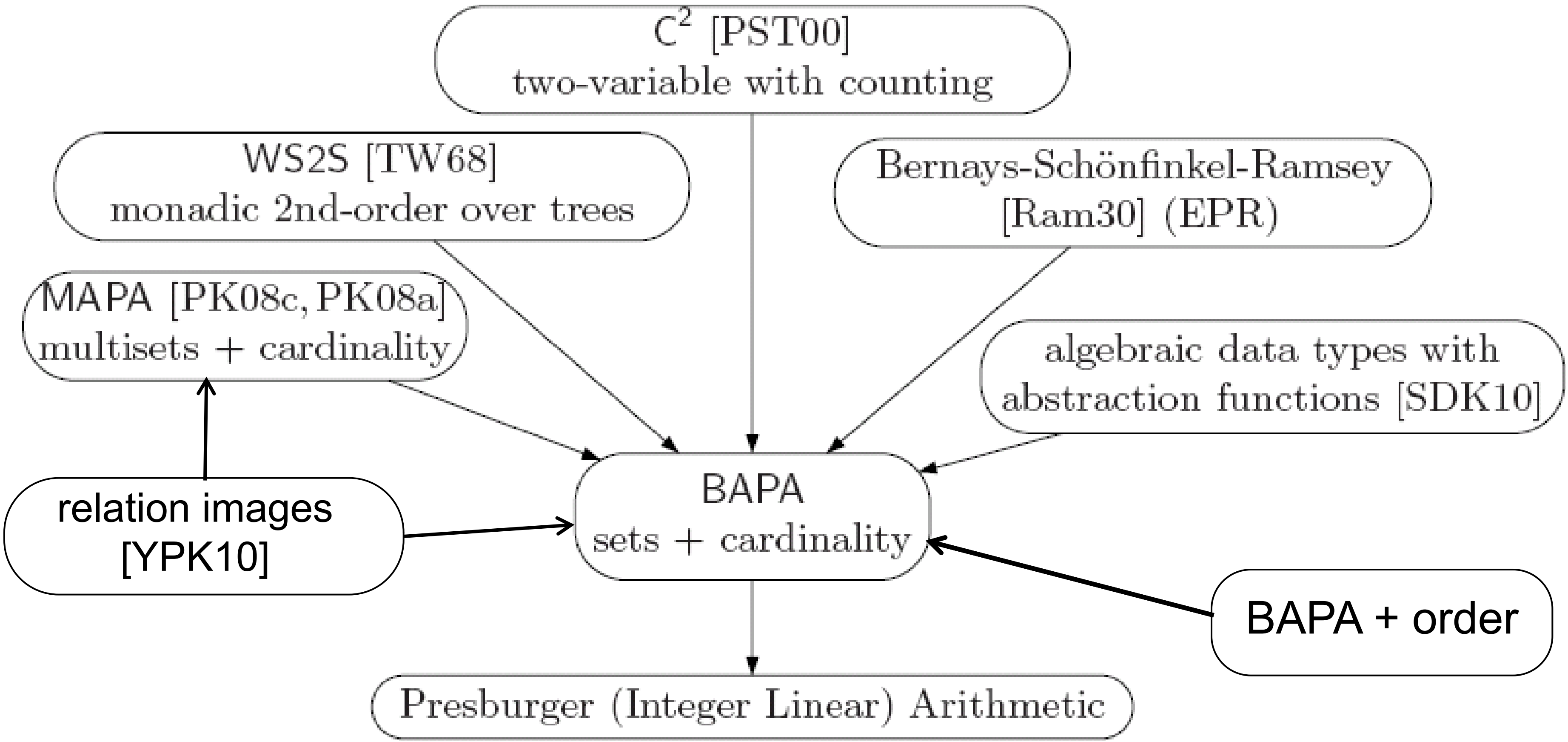
Symbols satisfy FO axioms – FO theories
- – in HOL finite formulas often suffice, (Ax /\ F)
- – up to system which part of formula are axioms

Program representation: complex structure
- – concurrency? recursion? mutation?

Executable.   Finitely bounded

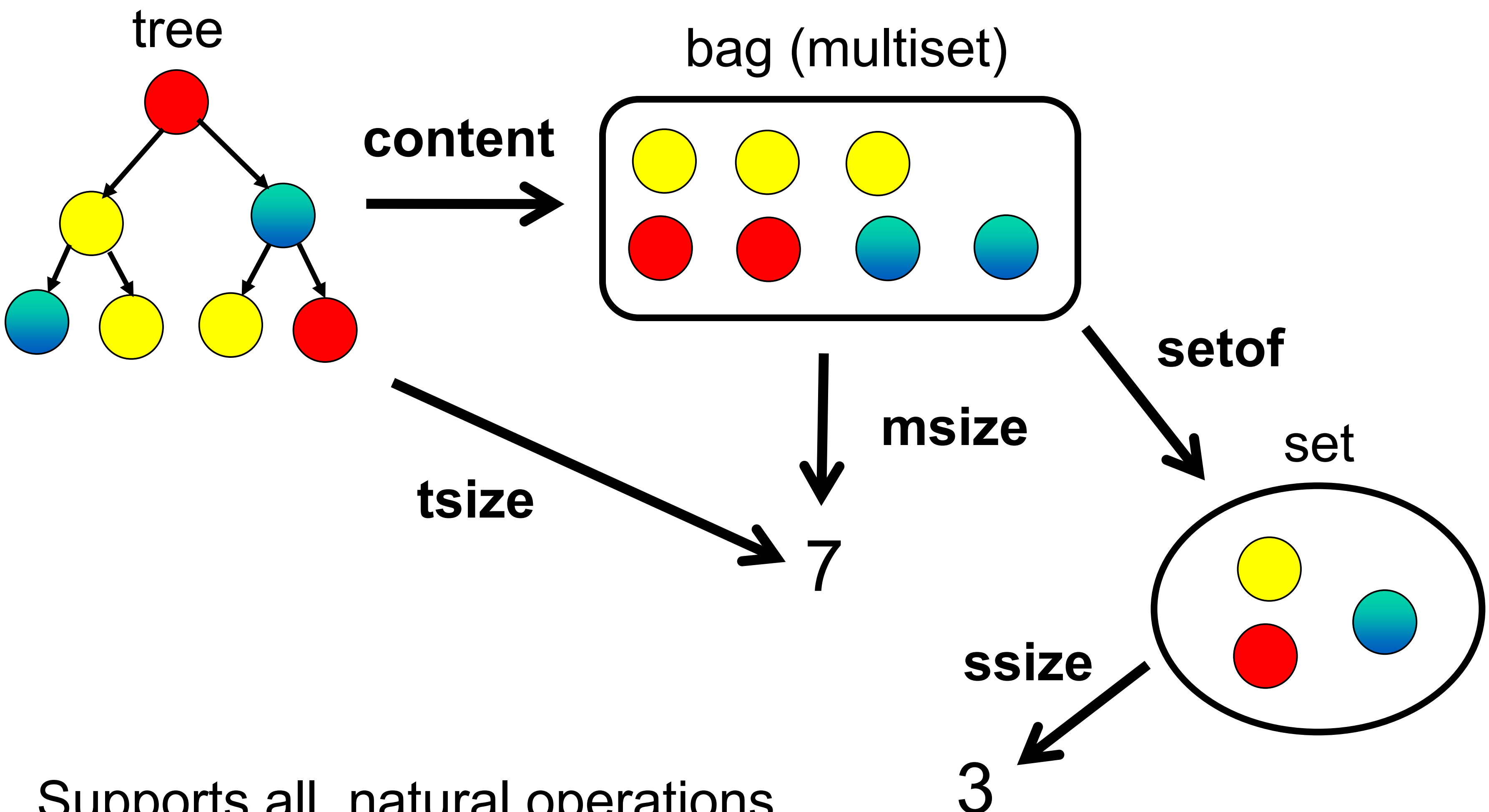
Procedure answers: 1) in fragment? 2) valid?

Our non-disjoint combination result



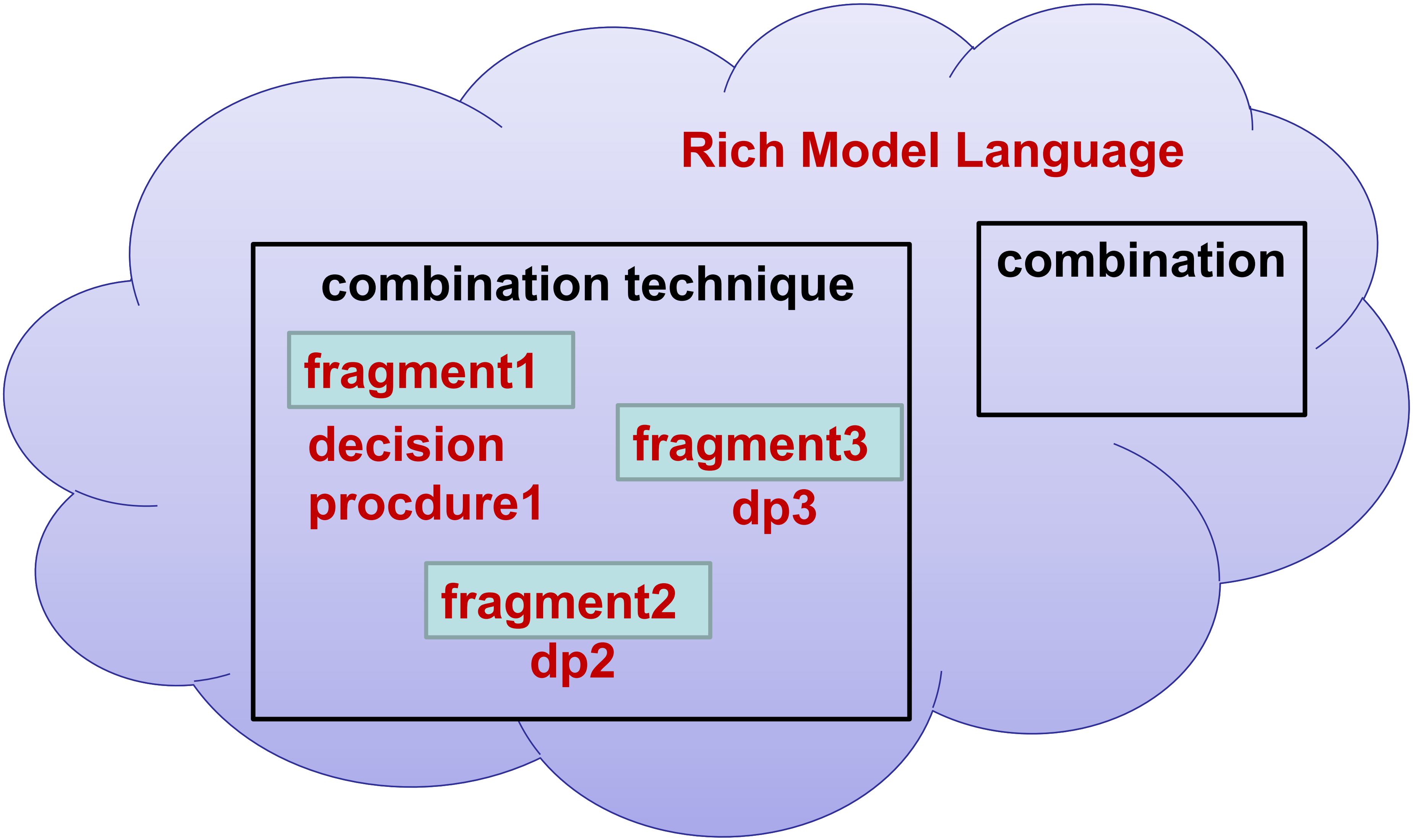
So far, using axiomatization with FOL provers, SMT provers, and HOL prover LEO II suggest that these general approaches do not work for these problems out of box

One Consequence
Calculus of Data Structures



Supports all natural operations
on trees, multisets, sets, and homomorphisms between them

This is one combination technique



Work Groups

1. **Rich Model Language**
   Design, Benchmarks (a unifying activity)
   Chair: Tobias Nipkow; V.Chair: Paul Jackson
2. **Decision Procedures** for
   Rich Model Language Fragments (key technique)
   Chair: Maria Paola Bonacina; V.Chair: Armin Biere
3. **Analysis of Executable Rich Models**
   large potential for practical impact
   Chair: Natasha Sharygina
4. **Synthesis from Rich Models**
   Chair: *Barbara Jobstmann*;V.Chair: Roderick Bloem

Formula-Based Analyses

Bounded reachability question as a formula

Interpolation-based analysis

- get invariants from absence of short error paths

Predicate abstraction

- propositional combinations of “given” formulas
- recently: add universal quantifiers (heap)

Template-based analyses

- invariants are polynomials (find coefficients)
- set constraints: invariants are sets of terms

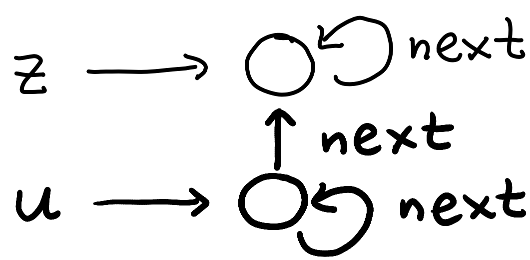
Candidate tools to incorporate into RMT

Rich Models for Static Analysis

Data-flow analysis $\xrightarrow{\gamma}$ Formula semantics

$x \to [0,+\infty)$

$y \to [1,255]$



$0 \le x \quad \land$

$1 \le y \le 255 \quad \land$

$\exists A,B.\ \ A \cap B = \emptyset \quad \land$

$z \in A \land next[A] \subseteq A \quad \land$

$u \in B \land$

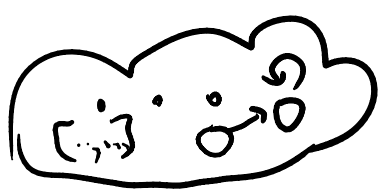
$next[B] \subseteq A \cup B$

data-flow transfer function

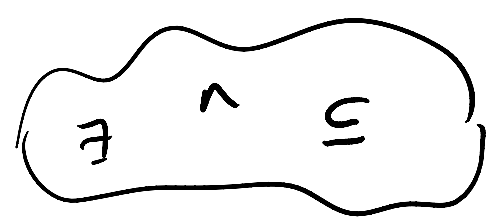
statement

$\boxed{u.next := y}$

postcondition



$\xrightarrow{\gamma}$



New requirements from analysis

Approximate a given formula by a formula in a given fragment

- extract information from user annotations
- eliminate quantifiers (intermediate states)
- approximate disjunction (join in lattice)
- approximate strongest postcondition (post#)

Avoid non-terminating sequence of formulas

- widening

Find a missing coefficient in a formula

- template based analysis of polynomials

Executing Specifications

Why

- execution is efficient constraint propagation
- debug specifications
- make programming languages higher level

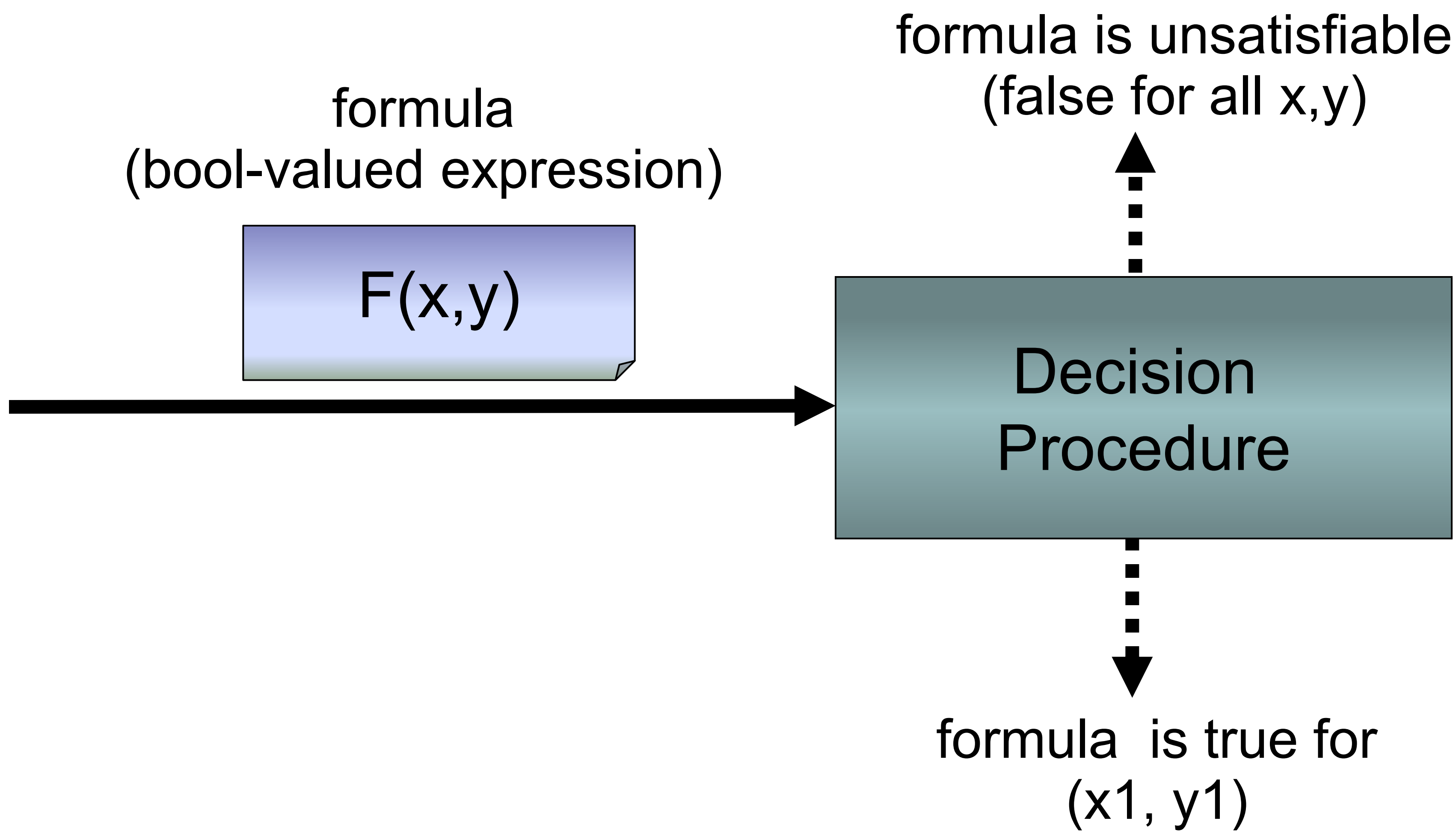
Approaches

- solve constraints at run-time (CLP)
- mode analysis (recent workshop in Belgrade)
- our recent work: delayed execution – ICSE'10
- compile constraints **synthesis** – PLDI'10

Work Groups

1. **Rich Model Language**
   Design, Benchmarks (a unifying activity)
   Chair: Tobias Nipkow; V.Chair: Paul Jackson
2. **Decision Procedures** for
   Rich Model Language Fragments (key technique)
   Chair: Maria Paola Bonacina; V.Chair: Armin Biere
3. **Analysis of Executable Rich Models**
   large potential for practical impact
   Chair: Natasha Sharygina
4. **Synthesis from Rich Models**
   Chair: *Barbara Jobstmann*;V.Chair: Roderick Bloem

Starting point: counterexample-generating decision procedures (satisfiability)



Example: integer linear arithmetic

formula F with integer variables

$10 < y \wedge x < 6 \wedge y < 3*x$

Decision Procedure

true for $x=4$, $y=11$

No a-priori bounds on integers
(add e.g. $0 <= y < 2^{64}$ if needed)

Synthesis procedure for integers

formula F with integer variables

$10 < y \wedge \mathbf{x} < 6 \wedge y < 3*x$

Synthesis Procedure

function g on integers
$g_x(y)=(y+1) \text{ div } 3$

precondition P on y
$10 < y < 14$

Two kinds of variables:
- inputs – here y
- outputs – here x

- P describes precisely when solution exists.
- $(g_x(y),y)$ is solution whenever P(y)

How does it work?

Quantifier elimination

Take formula of the form

$\exists$ x. F(x,y)

replace it with an **equivalent** formula

G(y)

without introducing new variables

Repeat this process to eliminate all variables

Algorithms for quantifier elimination (QE) exist for:

- Presburger arithmetic (integer linear arithmetic)
- set algebra
- algebraic data types (term algebras)
- polynomials over real/complex numbers
- sequences of elements from structures with QE

Example: test-set method for QE (e.g. Weispfenning'97)

Take formula of the form
$\exists x.\ F(x,y)$

replace it with an **equivalent** formula

$$\bigvee_{i=1}^{n} F_i(t_i(y),y)$$

We can use it to generate a program:

x = **if** $F_1(t_1(y),y)$ **then** $t_1(y)$
else if $F_2(t_2(y),y)$ **then** $t_2(y)$
…
else if $F_n(t_n(y),y)$ **then** $t_n(y)$
else throw new Exception("No solution exists")

Can do it more efficiently – generalizing decision procedures and quantifier-elimination algorithms (use **div**, **%**, …)

Example: Omega-test for Presburger arithmetic – Pugh'92

Presburger Arithmetic

$$T ::= k \mid C \mid T_1 + T_2 \mid T_1 - T_2 \mid C \cdot T$$
$$A ::= T_1 = T_2 \mid T_1 < T_2$$
$$F ::= A \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \mid \exists k.F$$

Presburger showed quantifier elimination for PA in 1929

- requires introducing divisibility predicates
- Tarski said this was not enough for a PhD thesis

Normal form for quantifier elimination step:

$$\bigwedge_{i=1}^{L} a_i < x \wedge \bigwedge_{j=1}^{U} x < b_j \wedge \bigwedge_{i=1}^{D} K_i \mid (x + t_i)$$

Parameterized Presburger arithmetic

Given a base, and number convert a number into this base

```
val base = read(…)
val x = read(…)
val (d2,d1,d0) = choose((x2,x1,x0) =>
  x0 + base * (x1 + base * x2) == x &&
  0 <= x0 < base &&
  0 <= x1 < base)
```

This also works, using a similar algorithm

- This time essential to have '**for'** loops

'for' loops are useful even for simple PA case

- reduce code size, preserve efficiency

Synthesis as Scala-compiler plugin

Given number of seconds, break it into hours, minutes, leftover

$$
\begin{array}{l}
\textbf{val}\ (\text{hours, minutes, seconds}) = \text{choose}((\text{h: Int, m: Int, s: Int}) \Rightarrow (\\
\quad ?\text{h} * 3600 + ?\text{m} * 60 + ?\text{s} == \underline{\text{totsec}}\\
\ \&\&\ 0 \leq ?\text{m}\ \&\&\ ?\text{m} \leq 60\\
\ \&\&\ 0 \leq ?\text{s}\ \&\&\ ?\text{s} \leq 60))
\end{array}
$$

parameter - variable in scope

our synthesis procedure

$$
\begin{array}{l}
\textbf{val}\ (\text{hours, minutes, seconds}) = \{\\
\quad \textbf{val}\ \text{loc1} = \text{totsec div } 3600\\
\quad \textbf{val}\ \text{num2} = \text{totsec} + ((-3600) * \text{loc1})\\
\quad \textbf{val}\ \text{loc2} = \text{min}(\text{num2 div } 60, 59)\\
\quad \textbf{val}\ \text{loc3} = \text{totsec} + ((-3600) * \text{loc1}) + (-60 * \text{loc2})\\
\quad (\text{loc1, loc2, loc3})\\
\}
\end{array}
$$

`Warning: solution not unique for: totsec=60`

Synthesis for Pattern Matching

```
def pow(base : Int, p : Int) = {
  def fp(m : Int, b : Int, i : Int) = i match {
    case 0 ⇒ m
    case 2∗j ⇒ fp(m, b∗b, j)
    case 2∗j+1 ⇒ fp(m∗b, b∗b, j)
  }
  fp(1,base,p)
}
```

Our Scala compiler plugin:

- generates code that does division and testing of reminder
- checks that all cases are covered
- can use any integer linear arithmetic expressions

Beyond numbers

Boolean Algebra with Presburger Arithmetic

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

Our results related to BAPA

- complexity for full BAPA (like PA, has QE)
- polynomial-time fragments
- complexity for Q.F.BAPA
- generalized to multisets
- combined with function images
- used as a glue to combine expressive logics
- **synthesize sets of objects from specifications**

Synthesizing sets

Partition a set into two parts of almost-equal size

```
val s = …
val (a1,a2) = choose((a1:Set[O],a2:Set[O]) ⇒
  a1 union a2 == s &&
  a1 intersect a2 == empty &&
  abs(a1.size – a2.size) ≤ 1)
```

http://lara.epfl.ch/dokuwiki/comfusy

Complete Functional Synthesis

Scala progrmaming language – developed in Martin Odersky's group at EPFL

Time improvements of synthesis

Example: propositional formula F

```
var p = read(…); var q = read(…)
val (p0,q0) = choose((p,q) => F(p,q,u,v))
```

- SAT is **NP-hard**
- generate BDD circuit over input variables
  - for leaf nodes compute one output, if exists
- running through this BDD is **polynomial**

Reduced NP problem to polynomial one

Also works for linear rational arithmetic (build decision tree with comparisons)

Rich Model Toolkit in LARA Group

Infrastructure for reliable computer systems

- Rich Model Language – unifying activity
  - an initial proposal based on Isabelle/HOL
- Decision Procedures – key enabling technique
  - new decision procedures, their combination
- Analysis of Transition systems – static analysis, abstract interpretation, verification
  - plans to work on constraint-based analyses
- Synthesis of systems correct by construction
  - currently for Presburger arithmetic and sets