Constraint Programming for Formal Methods and using Formal Methods

Michael Leuschel
University of Düsseldorf
Logical Foundations

• Typed first-order predicate logic
  • Well-Definedness Conditions to stay in two-valued logic

• Arithmetic over mathematical integers and implementable integers

• Set theory
  • Sets, Relations, Functions, Sequences
  • including higher-order functions

\[ p \in \text{dom}(a) \rightarrow \text{dom}(a) \wedge \forall i \cdot (i \in 1..(\text{size}(a)-1) \Rightarrow p(a(i)) < p(a(i+1))) \]
Constraint Programming for B
• **Validation** Tool for High-level Specifications

• Multiple Languages: B, Z, Event-B, TLA, CSP, B||CSP

• Multiple Validation Technologies: Animation, Model Checking, Refinement Checking, Disproving, …

• Most rely on ProB **Constraint Solving** Kernel for B Datatypes and Operators

http://www3.hhu.de/stups/prob/
Side Note: ProB can do CSP
Constraint Solving for B

• **Execution:**
  - \{2,3,5\} \cap (4..6) \rightarrow \{4,5\}

• **Proof:**
  - \(x \geq 0 \land n > 0 \vdash x + n > 0 \rightarrow YES\)

• **Constraint Solving:**
  - \(x \geq 0 \land n > 0 \land (x+n) \in \{2,3\} \rightarrow x=0, n=2\)
Overview of Talk

1. Applications of constraint solving for typical validation tasks for B (or other formal methods)

2. Ways to solve constraints involving B (or related formal methods)

3. Expressing typical constraint solving problems in B (and highlighting new Latex package)

4. Applications of expressing constraint problems in B

5. New integrated technique illustrated on an example
Applications of Constraint Programming for B

- Animate implicit / high-level specifications
  B4MSecure [Ledru et al. CAiSe’11]
iUML [Snook et al.], Alstom Interlocking Simulator [Mejia et al.]
CODA [Butler, Colley, Edmunds, Snook, Evans, Gran, Marshall]

- Constraint-based invariant, deadlock, refinement checks (ICLP’11)

- Model-Based Testing [SEFM’15], Beta [de Matos, Moreira, Neto 2012], [Moreira et al. TAP’15], BTestBox (https://github.com/ValerioMedeiros/BTestBox), [Dinca,Ipate,Stefanescu ABZ’12]

- Disprover/Prover for B and Event-B (SEFM’15), BEval [Medeiros,Deharbe]

- Enabling Analysis (ABZ’16)

- Symbolic Model Checking (BMC,k-Induction, IC3) (ABZ’16),
  Genesis [Radhouani, Idani, Ledru, Rajeb 15] [Ledru et al. CAiSe’11]

Event-B model(s) used in teaching

one event of third refinement shown:

... event add_to_stack
  any current_proc next_proc
  where
    @next_proc1 next_proc ∈ procs
    @next_proc2 current_command = next_proc
    @next_proc3 programs(next_proc) ≠ ∅
    @stack_size card(program_stack) < max_stack_size
    @current_proc current_proc ∈ procs ∧ current_proc ∈ ran(program_stack) ∧ current_proc = program_stack(card(program_stack))
  then
    @act2 current_command = null
    @current_proc2 program_stack = program_stack u {card(program_stack)+1→ next_proc}
    @act3 index(next_proc) = 0
  end

..
ProB Animator in Rodin for Lightbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Previous value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctx1_Field.field</td>
<td>{1-1, 1-2, 1-3}</td>
<td></td>
</tr>
<tr>
<td>Ctx1_Field.lights</td>
<td>{1-1, 1-2, 1-3}</td>
<td></td>
</tr>
<tr>
<td>main_queue_i</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>start_direction</td>
<td>left_direction</td>
<td>left_direction</td>
</tr>
<tr>
<td>start_x</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>start_y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>turn_r</td>
<td>{up_direction, right_d_}</td>
<td>{up_direction, right_d_}</td>
</tr>
<tr>
<td>Ctx2_Layer.level</td>
<td>{1-1, 2, 1-2, 2-2}</td>
<td></td>
</tr>
<tr>
<td>Ctx2_Layer.start_z</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Ctx3_Commands</td>
<td>{move, turn_right, turn_}</td>
<td>{move, turn_right, turn_}</td>
</tr>
<tr>
<td>max_size_stack</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>program_size</td>
<td>{main, 12}</td>
<td>{main, 12}</td>
</tr>
<tr>
<td>LEVELS</td>
<td>{level_1_2 ...}</td>
<td></td>
</tr>
<tr>
<td>LIGHTS</td>
<td>{level_1_2 ...}</td>
<td></td>
</tr>
<tr>
<td>PROGRAM_SIZE</td>
<td>{level_1_2 ...}</td>
<td></td>
</tr>
<tr>
<td>START_DIRECTIONS</td>
<td>{level_1_2 ...}</td>
<td></td>
</tr>
<tr>
<td>START_POSITIONS</td>
<td>{level_1_2 ...}</td>
<td></td>
</tr>
<tr>
<td>selection</td>
<td>level_1_5</td>
<td>level_1_1</td>
</tr>
<tr>
<td>Bot_direction</td>
<td>left_direction</td>
<td>left_direction</td>
</tr>
<tr>
<td>bot_x</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>bot_y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>light_state</td>
<td>{(1-2) = FALSE, (3-3) = FALSE}</td>
<td>(3-3) = FALSE</td>
</tr>
</tbody>
</table>

No event errors detected.
ProB with BMotionWeb running Lightbot
ProB with BMotionWeb running Lightbot
ProB with BMotionWeb for Pac Man
Animation: Need for Constraint Solving
Test-Case Generation and Bounded Model Checking

• model checking and test-case generation can be done on explicit state space

• problem when high-branching factor:
  • constant valuation,
  • operation parameters, …

• simple example:

MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INVARINANT c>=0 & c<=m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN c := c+i END END
MACHINE Counter
CONSTANTS m
PROPERTIES m : {127,255}
VARIABLES c
INVARIANT c>=0 & c<=m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN
    c := c+i END
END

state space at the
time ProB finds error
(default mixed bf/df)
with breadth-first even worse part of explored state space:

MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INVARIANT c\geq0 & c\leq m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN
    c := c+i END
END
Idea

- do not enumerate all possible constant values (m) and all parameters (i)
- let constraint solver instantiate them depending on target:
  - test-case generation: execute all/one event leading to a target predicate (valid end state of a test)
  - bounded-model checking (BMC): execute a sequence of events leading to invariant violation
MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INVARIANT c >= 0 & c <= m
INITIALISATION c := 0
OPERATIONS
    incby(i) = PRE i:1..64 THEN
        c := c + i
    END
END
BMC Algorithm: Steps

INITIALISATION

not(INV)

Call to solver:

\[ m \in \{127, 255\} \land c = 0 \land \neg(c \geq 0 \land c \leq m) \]
BMC Algorithm: Steps

MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INVARIANT c>=0 & c<=m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i: 1..64 THEN
  c := c + i END
END

Call to solver:
m \in \{127, 255\} \land c = 0
BMC Algorithm: Steps

Call to solver:
\[ m \in \{127, 255\} \land c = 0 \land (c \geq 0 \land c \leq m) \land \
i \in 1..64 \land c' = c + i \land \neg(c' \geq 0 \land c' \leq m) \]
BMC Algorithm: Steps

MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INARIANT c\geq0 & c\leq m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN
    c := c+i END
END

Call to solver:
m\in\{127,255\} \land c=0 \land (c\geq0 \land c\leq m) \land
i\in1..64 \land c'=c+i
BMC Algorithm: Steps

MACHINE Counter
CONSTANTS m
PROPERTIES m : \{127,255\}
VARIABLES c
INVARIANT c>=0 & c<=m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN
    c := c+i END
END

Call to solver:

m\in\{127,255\} \land (c\geq0 \land c\leq m) \land
i\in1..64 \land c'=c+i \land
i'\in1..64

INVARIANT
〜 incby
〜 guard of incby ?

not(INV)
INV
not(INV)
INV
BMC Algorithm: Steps

INITIALISATION

not(INV)  INV  ✓

not(INV)  INV  ✓

Call to solver:

\[ m \in \{127, 255\} \land c = 0 \land (c \geq 0 \land c \leq m) \land i \in 1..64 \land c' = c + i \land (c' \geq 0 \land c' \leq m) \land i' \in 1..64 \land c'' = c' + i' \land \neg(c'' \geq 0 \land c'' \leq m) \]
BMC Algorithm: Steps

MACHINE Counter
CONSTANTS m
PROPERTIES m : {127,255}
VARIABLES c
INVARIANT c>=0 & c<=m
INITIALISATION c:=0
OPERATIONS
  incby(i) = PRE i:1..64 THEN
c := c+i END
END

Counterexample found!
Counterexample

Call to solver:

\[
\begin{align*}
&m \in \{127, 255\} \land c = 0 \land (c \geq 0 \land c \leq m) \land \\
&i \in 1..64 \land c' = c + i \land (c' \geq 0 \land c' \leq m) \land \\
&i' \in 1..64 \land c'' = c' + i' \land \neg(c'' \geq 0 \land c'' \leq m)
\end{align*}
\]

Solution:

\[
\begin{align*}
&m = 127 \\
&c = 0 \\
&i = 64, \quad c' = 64 \\
&i' = 64, \quad c'' = 128
\end{align*}
\]
BMC/Test-Case Algorithm

1. **Input:** set of events \( \text{target} \subseteq \text{Events} \) and a set of events \( \text{final} \subseteq \text{Events} \)
2. Initially we set \( \text{paths} := \{[]\} \) where \([[]\) is the path of length 0
3. \( \text{depth} := 0 \)
4. \( \text{final} \) is the set of events which have to be final; in our case study: \( \text{target} \)
5. \( \text{target}' := \text{target} \)
6. \( \text{for every } p \in \text{paths} \text{ of length } \text{depth} \text{ do:} \)
7. \( \text{for every } t \in \text{target} \cap \text{enable}(p) \text{ do:} \)
8. \( \text{if solve constraints of path } p \leftarrow t \text{ then} \)
9. \( \text{target} := \text{target} \setminus \{t\} \), store solution as test for \( t \)
10. \( \text{path} := \text{path} \cup \{p \leftarrow t\} \)
11. \( \text{fi} \)
12. \( \text{od} \)
13. \( \text{od} \)
14. \( \text{if } \text{target} = \emptyset \text{ or maximum depth reached then return fi} \)
15. \( \text{for every } p \in \text{paths} \text{ of length } \text{depth} \text{ do:} \)
16. \( \text{for every } e \in \text{feasibleAfter}(p) \setminus \text{target}' \setminus \text{final} \text{ do:} \)
17. \( \text{if solve constraints of path } p \leftarrow e \text{ then path} := \text{path} \cup \{p \leftarrow e\} \text{ fi} \)
18. \( \text{od} \)
19. \( \text{od} \)
20. \( \text{depth} := \text{depth} + 1; \text{ goto 5} \)
Application

• More details about an application for Java Card security in [SEFM’15] paper

• runtime reduced from estimated 2 years to 45 minutes for creating all relevant mutants
Demo:
Example from yesterday

MACHINE EmailTestCaseGenExample
SETS User = {u1,u2,u3,u4,u5}
VARIABLES Wifi, Bluetooth, Airplane, EmailLoggedIn, to_deliver
INVARIANT
Wifi:BOOL &
Bluetooth:BOOL &
Airplane:BOOL & (Airplane=TRUE => (Wifi=FALSE & Bluetooth=FALSE)) &
EmailLoggedIn <: User &
(EmailLoggedIn /= {} => Wifi=TRUE) &
to_deliver: User <-> User
INITIALISATION
Wifi,Bluetooth,Airplane := FALSE,FALSE,FALSE || EmailLoggedIn := {} || to_deliver := {} 
OPERATIONS
TurnWifiOn = PRE Wifi=FALSE THEN Wifi:=TRUE || Airplane :=FALSE END;
TunrAirplaneModeOn = PRE Airplane=FALSE THEN
   Airplane := TRUE || Wifi,Bluetooth := FALSE,FALSE
   || EmailLoggedIn := {}
END;
LogIntoEmail(u) = PRE u:User & Wifi=TRUE & u /: EmailLoggedIn THEN
    EmailLoggedIn := EmailLoggedIn / {u} END;
LogOutOfEmail(u) = PRE u:EmailLoggedIn THEN
    EmailLoggedIn := EmailLoggedIn \ {u} END;
SendEmail(u,to) = PRE u:EmailLoggedIn & to:User THEN to_deliver := to_deliver \ {u}->to} END;
ReceiveEmail(u,from) = PRE u:EmailLoggedIn & from|->u:to_deliver & from/=u THEN
   to_deliver := to_deliver \ {from|->u} END;
SynContacts(u) = PRE u:EmailLoggedIn THEN skip END
END
Recent ProB developments for explicit state model checking

- **parB**: uses ZeroMQ for parallel model checking

- **TLC Backend**: uses TLC model checker as alternate backend; very fast for low-level B models, can be parallelised (AWS), can deal with very large state spaces (on disk)

- **LTSMin Bridge**: links ProB with LTSMin model checking engine:
  - BDD-style symbolic model checking
  - Partial Order Reduction
Conclusion Part 1

• Constraint programming for B enables many important validation and verification activities

• enables animation (and thus model checking) of high level specifications

• many validation tasks can be encoded as finding the solution of a B predicate: bounded model checking, test-case generation

• complements proof
Part 2: Overview of Constraint Solvers for B
Challenges of Constraint Programming for B

- **undecidable** formal method, with nested **quantifiers** and negation: \( \forall n. (n > 2 \Rightarrow \neg \exists a, b, c . a^n + b^n = c^n) \)

- **higher-order** functions and relations, **unbounded** variables

- we want **reliable** solving, e.g., satisfy CENELEC EN50128 T2 or T3 tool classification

- we want to find **all** solutions to predicates

- we want to be able to deal with real data: **large** relations, large integers
Default ProB Solver

- written in SICStus Prolog with CLP(FD) finite domain library

- ?- X in 1..3, Y #= X+X. \implies Y in 2..6

- ProB uses CLP(FD) for integers and deferred set / enumerated set elements

- >>> x:1..3 & y=x+x \implies x = ?:1..3 & y = ?:2..6

- Booleans are encoded separately using pred_false,pred_true values

- Set have three representations: Prolog lists, AVL-trees, \textbf{symbolic} closures and custom solver using Prolog co-routines which trigger upon set-instantiations

- Main Idea: \textbf{determinism} first, efficient set representation first
Demo with -repl

• ! deterministic mode

```python
>>> x:1..3 & y=x+x
  ~ x = ?:1..3 & y = ?:2..6

>>> r = {1|->11, 2|->22, 3|->33, 4|->44} & x|->y:r & y>22
  ~ x = ?:3..4 & y = ?:{33}\{44}

>>> cube = {x,y|y=x*x*x} & cube(v)=512
```
ProB Architecture

ProB Waitflags Store

ProB Pairs, Sets, Relations, Records Solver

ProB Base Sets Solver

ProB Integer Solver

ProB Boolean Predicate Solver

ProB Kernel

SICStus CLP(FD)

Enumeration

Formula
Kodkod Library

- Java library to translate first-order relational calculus expressions into propositional logic for SAT solving
  - can make use of various SAT solvers
  - core of the Alloy tool
Overview of ProB Kodkod Backend

1. ProB Solver
2. Kodkod Call
3. ProB Solver

- B Properties
- Interval Analysis
- Deterministic Part
- First-Order B
- First-Order Relational Calculus
- Kodkod Translation
- Rest

- Kodkod Java Library
- SAT Solver

Solution
Demo with -repl

>>> :kodkod r: 1..5 <-> 1..5 & (r;r) = r & r /= {} & dom(r)=1..5 & r[2..3]=4..5

kodkod ok:   r : 1 .. 5 <-> 1 .. 5 & (r ; r) = r & r /= {} & do... ints: irange(0,5), intatoms: irange(0,5)

Kodkod module started up successfully (SAT solver SAT4J with timeout of 1500 ms).

Times for computing solutions: [4,3,4,2,1,5,1,2,2,1,2,1,1,1,1,1,1,1,1,1,1,1]

Existentially Quantified Predicate over r is TRUE

Solution:

   r = {(1|->4),(2|->4),(3|->4),(3|->5),(4|->4),(5|->4),(5|->5)}

>>> $

% Type: pred

% Eval Time: 10 ms (400 ms walltime)
ProB Z3 Backend

- Prior work: Deharbe et al.: SMT solvers for proof in Rodin
- Here use Z3’s native set support
- Translation more tuned for solving than for proof
- Two possible integrations:
  - 1) translate entire predicate to Z3
  - 2) interleave calls to Z3 in ProB interpreter
Unsatisfiability

• $x < y \& y < x$

• Time-out with CLP(FD)

• Kodkod backend not applicable (no finite bound)

• Z3: quickly finds inconsistency

• Z3 often very good at detecting inconsistencies; useful in ProB disprover or in interleaved interpreter to prune branches
Satsifiable (Model Finding)

- KISS*KISS=PASSION puzzle

- \{K,P\} <: 1..9 & \{I,S,A,O,N\} <: 0..9 &
  \((1000*K+100*I+10*S+S) * (1000*K+100*I+10*S+S) = 1000000*P+100000*A+10000*S+1000*S+100*I+10*O+N\) & \card(\{K, I, S, P, A, O, N\}) = 7

- 0 ms with ProB

- 800 ms with Kodkod backend

- 4570 ms with Z3 backend
Anecdotal Evidence

• ProB standard Prolog solver: good for integers, disequalities, finding models, can deal with symbolic set representations
  • not good for unbounded inconsistent predicates, complicated relational constraints with image, closure,…

• Kodkod/SAT: good for enumerated sets, relational operators (composition, closure, image)
  • not good for large integers, large sets; cannot deal with sets of sets, unbounded values

• Z3/SMT: good for detecting inconsistencies, particular unbounded domains
  • unreliable for finding models, finite problems often get translated into unbounded SMTLib problems
Part 3: Constraint Programming in B
Why do CP in B?

- Teaching B, Mathematics, Theoretical Computer Science
- More expressive language (will be shown on examples)
- Proof Support (from B)
- Stress test our solver for B
- Experiment with an alternate approach of using B:
  - formal models as artefacts at runtime
  - no code generation, or much more limited code generation
Latex Slides generated with B and ProB
Part 4: Applications of Constraint Programming within B
Practical Use?

- Solving Puzzles with B is nice for teaching
- Produces test cases for ProB
- But does it have any practical significance or application?
Railway Applications

Use of the B Method developed by CLEARSY SYSTEM ENGINEERING

Metros and Trains equipped with B SIL4 software

Use of ProB to validate topology and deployment configuration
Data Validation

http://www.data-validation.fr/

Railways Data Validation
Formal Methods in action

Data Validation & Reverse Engineering
6 June 2013 13:57 min · Leave a Comment · Clear5y

Data validation principles have been applied recently to a railways reverse-engineering project with great success. B and ProB have demonstrated again how efficient they are when used in combination.

This project was aimed at redeveloping an IDE for Embedded Diagnosis Systems (EDS), mostly from scratch as source code and development documentation are lost. Obsolete hardware (with no direct access to the filesystem) and original IDE were used for black box testing.
Data Validation for Thales RBC

- Raw Input Data
- Transformation to B
- B Model
- Generated Data
- Validation Rules
- Formalisation of rules
- Validation Rules in Natural Language
- Interactive Validation Tool
- Results
- Instructions
- Application
  - UI, Reports
Example B Rule

"Two signals valid for the same direction should not be placed at the same location"

∀signal1, signal2 :: (signal1 ∈ Signals ∧ signal2 ∈ Signals ∧ signal1 ≠ signal2 ∧ Signal_Direction(signal1) = Signal_Direction(signal2) ⇒ ¬(Signal_TrackSegment(signal1) = Signal_TrackSegment(signal2) ∧ Signal_OperationalKM(signal1) = Signal_OperationalKM(signal2)))
Benefits

• declarative notation of the rules is well suited for reviews and eases the certification process

• correctness of complex computations can be ensured by formal assertions

• elimination of communication problems and quick turn-around

• product line management reflected in formal model
Reverse Engineering of Application Binary with ProB

bij: G7 --> ADR &

!xx.(xx: G7 => bij[next[{xx}]] = suiv[bij[{xx}]]])

162! =
1229694218739449434110178928491750176572300599427169306620762521167814540117728965860988098467051531783599
50744299047097082734018078243654115928975695099566042246320538220924308010459938381430588227927174194100982
18920470961529319832639077341092590387200000000000000000000000000000000000000
ProB Constraint Solver in Action

Detect and Fix University Timetabling Conflicts [FM’15]
Alternate Encodings (of simpler course model)

• ProB model: milliseconds to seconds to solve courses or detect impossibility

• Alloy: not successful; not able to solve timetabling for size of courses we require

• Z3: initial models very slow (hours to find solutions); latest version faster (good for inconsistency; still relatively slow for finding models: minutes vs seconds in ProB)
Part 5: A small case study: highlight expressivity of B and showcase new solving technique (Latex Slides)
Conclusions

• B/Event-B encodings often very compact and elegant

• ProB good at arithmetic & \(\neq\) (Alphametic, Queens)
  
  • success for industrial case studies (data validation for Siemens, Alstom, General Electric, RATP, …)
  
  • extensive validation, double chain available

• Can deal with large data and (in some cases) with higher-order datatypes and unbounded domains and infinite sets symbolically

• Kodkod backend very good for relational image, closure, composition

• Z3 backend very good for detecting inconsistent, unbounded predicates

• Combined ProB CLP(FD) and Kodkod backend shows promise
One encouraging result: SMTLib Competition 2016

**NIA (Main Track)**

Competition results for the NIA division as of Mon Jun 27 20:09:59 GMT

Benchmarks in this division: 9

Winners:

<table>
<thead>
<tr>
<th>Sequential Performances</th>
<th>Parallel Performances</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProB</td>
<td>ProB</td>
</tr>
</tbody>
</table>

**Result table**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Sequential performance</th>
<th>Parallel performance</th>
<th>Other information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error Score</td>
<td>Correctly Solved Score</td>
<td>avg. CPU time</td>
</tr>
<tr>
<td>CVC4</td>
<td>0.000</td>
<td>5.000</td>
<td>0.019</td>
</tr>
<tr>
<td>ProB</td>
<td>0.000</td>
<td>8.000</td>
<td>0.888</td>
</tr>
<tr>
<td>vampire_smt_4.1</td>
<td>0.000</td>
<td>6.000</td>
<td>335.043</td>
</tr>
<tr>
<td>vampire_smt_4.1_parallel</td>
<td>0.000</td>
<td>7.000</td>
<td>537.677</td>
</tr>
<tr>
<td>z3n</td>
<td>0.000</td>
<td>9.000</td>
<td>0.038</td>
</tr>
</tbody>
</table>
The Goal

- Dealing with integers, large data like CLP (SAT/SMT not enough to automatically detect simple high-level propagation rules)

- Learning like SAT, SMT (CLP propagation not enough in presence of complicated constraints)

- Symbolic functions with good performance (partial evaluation ?)

- (Executable) Models as Runtime Artifacts
  Executable Mathematics
Thanks!

Jens Bendisposto
Carl Friedrich Bolz
Ivo Dobrikov
Nadine Elbeshausen
Fabian Fritz
Marc Fontaine
Stefan Hallerstede
Dominik Hansen
Christoph Heinzen
Michael Jastram
Philip Körner
Sebastian Krings
Lukas Ladenberger
Li Luo
Daniel Plagge
Mireille Samia
David Schneider
Corinna Spermann
Dennis Winter

Michael Butler
Thierry Massart
Edd Turner
Theory Plugin support

context Demo

constants cube isqrt sol1 sol2

axioms
@axm0 cube = \( \lambda x \cdot x \in \mathbb{Z} \mid x \cdot x \cdot x \) // observe: no domain restriction on y

@axm1 isqrt = \{ x \mapsto r \mid x \in \mathbb{N} \land r \in \mathbb{N} \land r \cdot r \leq x \land (r + 1) \cdot (r + 1) > x \} // automatically detected as symbolic in new ProB; observe: no explicit finiteness restriction on x

theorem @thm0 \{ y \mid \text{cube(cube(y))} = 512 \} = \{ 2 \} // observe: no domain restriction on y

theorem @thm1 \text{isqrt}(100) = 10

theorem @thm2 \text{isqrt}(10) = 3

theorem @thm3 \text{isqrt}[1..20] = 1.4

theorem @thm4 \{ 1\mapsto 4, 2\mapsto 9, 3\mapsto 26 \} ; \text{isqrt} = \{ 1\mapsto 2, 2\mapsto 3, 3\mapsto 5 \}

theorem @thm5 \{ x \mid \text{isqrt}(x) = 10 \} = 100..120 // constraint solving with isqrt; observe: no restriction on x

theorem @thm6 \text{cls}({ 1\mapsto 2, 2\mapsto 3 }) = { 1\mapsto 2, 2\mapsto 3, 3\mapsto 2, 4\mapsto 3 }

theorem @thm7 \text{cls}(\text{isqrt})[[16]] = \{ 4, 2, 1 \}

theorem @thm8 \text{cls}(\text{isqrt})[[1024]] = \{ 1, 2, 5 \}

@axm8 sol1 = \text{cls}(\text{isqrt})[[1000000]]

theorem @thm_sol1 sol1 = \{ 1, 2, 4, 17, 316 \}

@axm9 sol2 = \{ x \mid x \in 1..100 \land \text{cls}(\text{isqrt})[\{ x \}] = \{ 1, 2, 4 \} \}

theorem @thm_sol2 sol2 = 16.24

theorem @nested \{ x \mid x \in 1..100 \land \{ y \mid y \in x..x+10 \} = \{ z \mid z \geq x \land z < x+11 \land \exists v \land (v > x) \} \} = 1..100

end