Prolog for Verification, Analysis and Transformation Tools

Michael Leuschel
University of Düsseldorf

VPT-2020/21

Prologue

Prolog: First Contact

University of Brussels
Licence en Informatique
1988

```
delete(X, [X|Y], Y).
delete(U, [X|Y], [X|Z]) :- delete(U, Y, Z).
```

Prolog for Programming ?? Really ??

Prolog: Second Contact

- Master of Artificial Intelligence, 1992/93, KU Leuven
- Topic for Master's thesis: partial evaluation of Prolog
  - Prolog, logic programming: very nice mathematical properties
  - yes, let's try and write a partial evaluator for it
  - But I would rather write it in Scheme or ML
  - But then I cannot make it self-applicable
  - Ok, I will try and write the partial evaluator in Prolog itself

30 years later

Portfolio of Prolog Tools

ECCE
Online Specialiser

XTL
CTL Model Checker

Logen
Offline Specialiser

BTA
Size-Change

Lix
Self-Applicable

ProB

Animator

Model Checker

Refinement Checker

VisB

WD-Prover

CSP Interpreter

B, Z, Event-B, TLA+ Interpreter

Object Petri Net Interpreter

Sets, Functions, … Solver

Java ByteCode Interpreter

Promela Interpreter

- **Validation** Tool for **High-level** Specifications
  - Multiple Languages: B, Z, Event-B,TLA,Alloy, CSP, B||CSP, XTL (Prolog)
  - Multiple V&V Technologies: Animation, Model Checking, Refinement Checking, Disproving, …
  - Most rely on Prolog **Constraint Solving** Kernel for B Datatypes and Operators

http://www3.hhu.de/stups/prob/

ProB Jupyter Kernel

KISS PASSION Puzzle

A slightly more complicated puzzle (involving multiplication) is the KISS * KISS = PASSION problem.

In [3]:

```
    {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
```

Out[3]: $TRUE$

Solution:

- $P = 4$
- $A = 1$
- $S = 3$
- $I = 0$
- $K = 2$
- $N = 9$
- $O = 8$

ProB Jupyter Kernel

Mixing Functional Programming with Constraint Programming

In B we can also define (higher-order) functions and mix those with logical predicates.

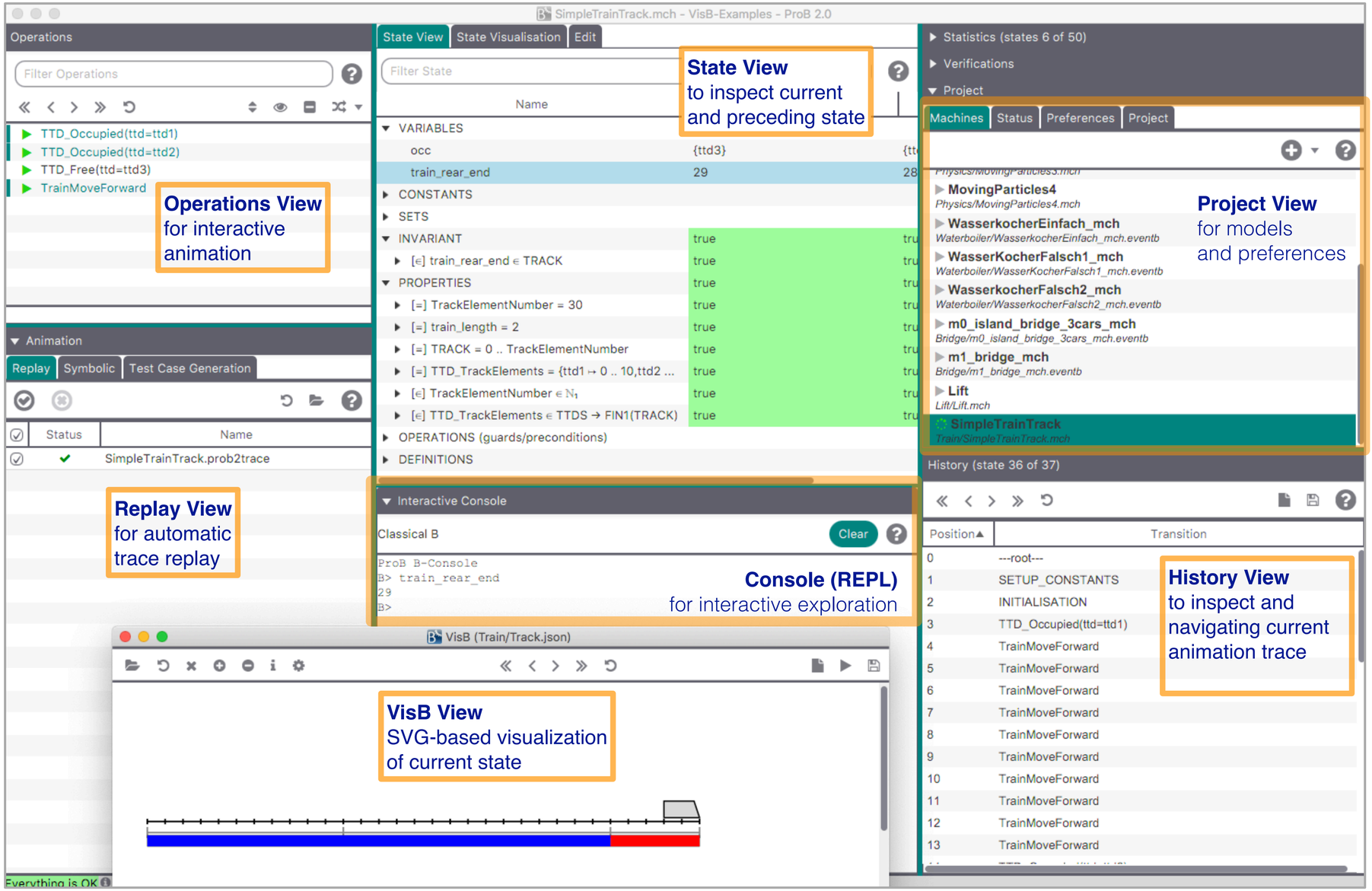
In [12]:
```
1 f = λx.(x∈ℤ|x∗x) ∧ res={x|f(x) = 100}
```

Out[12]: $TRUE$

Solution:

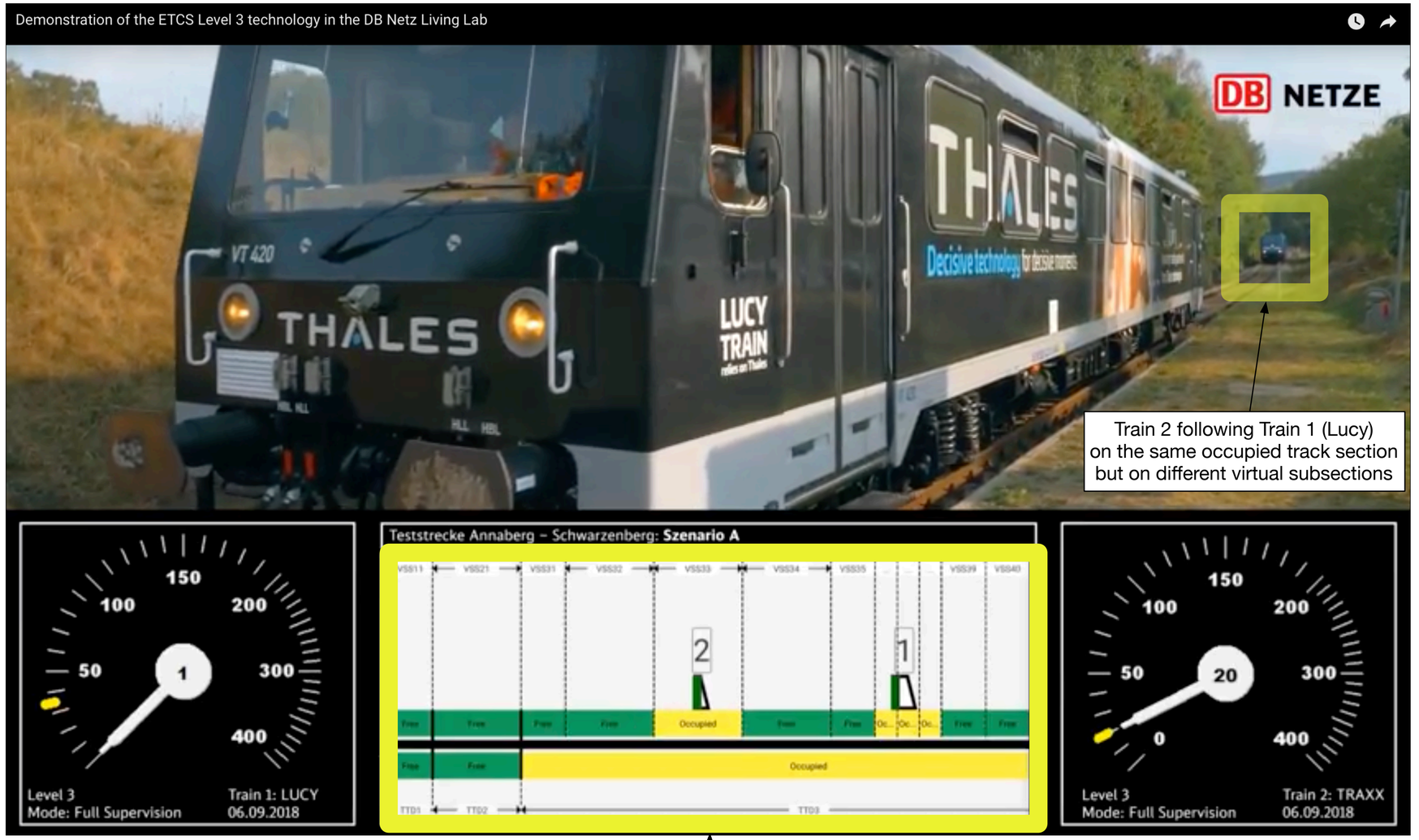
- $res = \{-10, 10\}$
- $f = \lambda x \cdot (x \in INTEGER \mid x * x)$

ProB2-UI


SimpleTrainTrack.mch - VisB-Examples - ProB 2.0
Operations
Filter Operations
TTD_Occupied(ttd=ttd1)
TTD_Occupied(ttd=ttd2)
TTD_Free(ttd=ttd3)
TrainMoveForward
Operations View
for interactive
animation
State View
State Visualisation
Edit
Filter State
Name
State View
to inspect current
and preceding state
VARIABLES
occ
{ttd3}
train_rear_end
29
CONSTANTS
SETS
INVARIANT
[∈] train_rear_end ∈ TRACK
PROPERTIES
[=] TrackElementNumber = 30
[=] train_length = 2
[=] TRACK = 0 .. TrackElementNumber
[=] TTD_TrackElements = {ttd1 ↦ 0 .. 10,ttd2 ...
[∈] TrackElementNumber ∈ ℕ1
[∈] TTD_TrackElements ∈ TTDS → FIN1(TRACK)
OPERATIONS (guards/preconditions)
DEFINITIONS
Statistics (states 6 of 50)
Verifications
Project
Machines
Status
Preferences
Project
MovingParticles4
Physics/MovingParticles4.mch
WasserkocherEinfach_mch
Waterboiler/WasserkocherEinfach_mch.eventb
WasserKocherFalsch1_mch
Waterboiler/WasserKocherFalsch1_mch.eventb
WasserkocherFalsch2_mch
Waterboiler/WasserkocherFalsch2_mch.eventb
m0_island_bridge_3cars_mch
Bridge/m0_island_bridge_3cars_mch.eventb
m1_bridge_mch
Bridge/m1_bridge_mch.eventb
Lift
Lift/Lift.mch
SimpleTrainTrack
Train/SimpleTrainTrack.mch
Project View
for models
and preferences
Animation
Replay
Symbolic
Test Case Generation
Status
Name
SimpleTrainTrack.prob2trace
Replay View
for automatic
trace replay
Interactive Console
Classical B
Clear
ProB B-Console
B> train_rear_end
29
B>
Console (REPL)
for interactive exploration
History (state 36 of 37)
Position
Transition
0 ---root---
1 SETUP_CONSTANTS
2 INITIALISATION
3 TTD_Occupied(ttd=ttd1)
4 TrainMoveForward
5 TrainMoveForward
6 TrainMoveForward
7 TrainMoveForward
8 TrainMoveForward
9 TrainMoveForward
10 TrainMoveForward
11 TrainMoveForward
12 TrainMoveForward
13 TrainMoveForward
History View
to inspect and
navigating current
animation trace
VisB (Train/Track.json)
VisB View
SVG-based visualization
of current state
Everything is OK

https://www.youtube.com/watch?v=FjKnugbmrP4

ProB in Action : Formal Models in Realtime



ProB running in real-time animating a
formal B model of the **Hybrid-Level 3** principles
developed by a team from the University of Düsseldorf and Thales with support from ClearSy

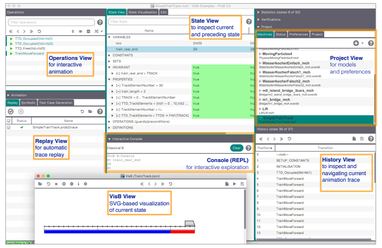
ProB for **Data Validation: Industrial Uses**

- Line 1 Paris, the second CDGVAL line LISA at the CDG airport in Paris, São Paulo line 4, ALGER line 1, Barcelona line 9, all by **Siemens** using **RDV** built-on top of **ProB**,
- more metro lines in Paris managed by **RATP** using **OVADO** which includes a tool developed called predicateB as first chain (development funded by **CLEARSY** and been maintained and evolved by Systerel for the last 15 years) and **ProB** as secondary tool chain
- **Alstom** for their **URBALIS** 400 CBTC system in 2014 using a tool based on **ProB** called **DTVT** developed by CLEARSY for various lines, e.g., in Mexico, Toronto, São Paulo and Panama
- **Alstom** and **SNCF** also applied data validation for ETCS-Level 1 software in 2018 using another tool developed by CLEARSY using **ProB**.
- Together with **Systerel**, **Alstom** conducted data validation of the Octys CBTC for RATP in 2017 using the **OVADO** tool.
- by **Thales** using a tool based on **ProB** called **Rubin** for checking engineering rules of their ETCS Radio Block Centre
- Other tools based on **ProB** were developed by **CLEARSY** such as **Dave** for **General Electric** or the latest generation tool called **Caval (**aka **ClearSy DataSolver)**.

FM Success Story

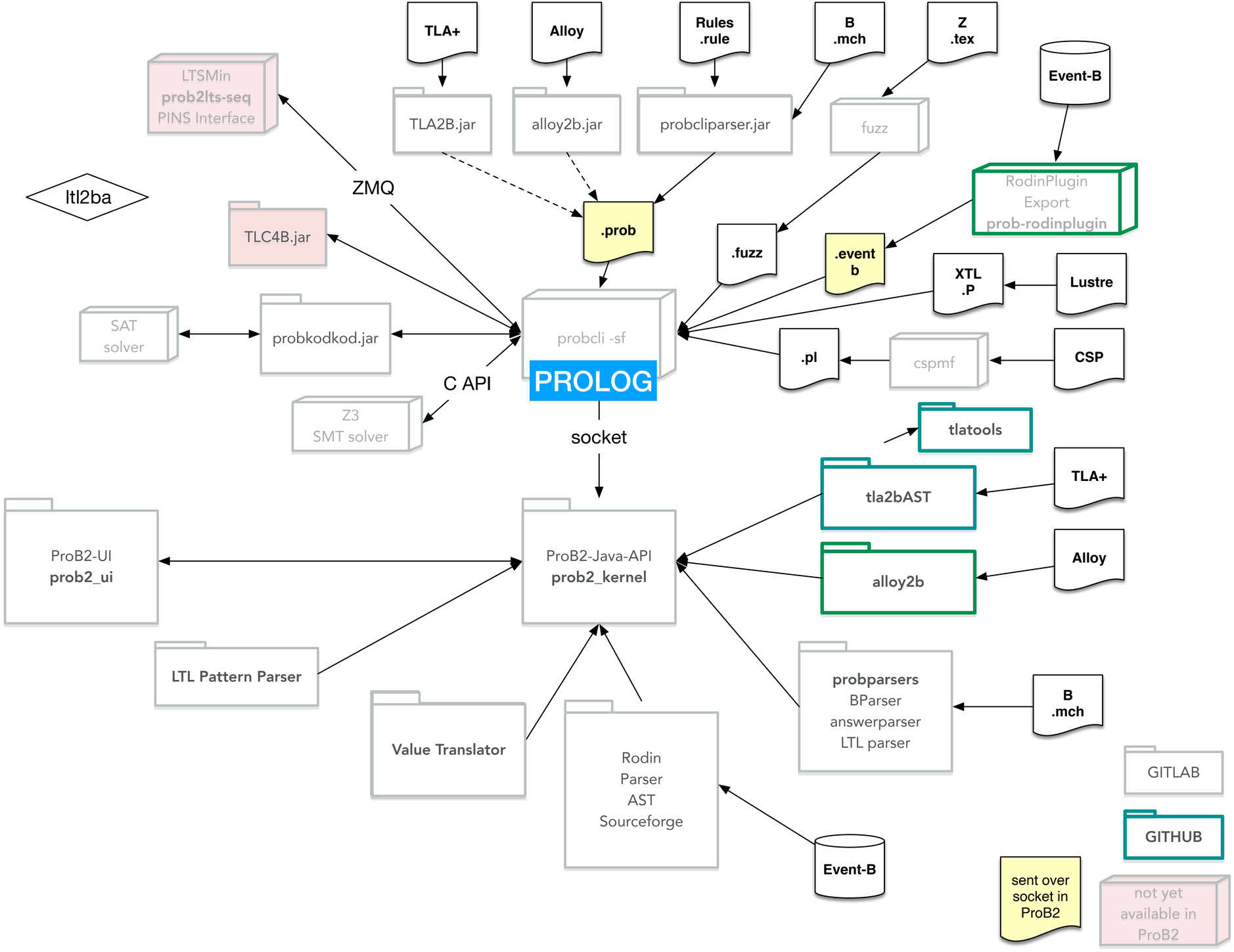
ProB Source Lines of Code

Component	SLOC	Language
ProB kernel	170,206	Prolog
B-Parser	12,079	Java
	1,774	SableCC
Tcl/Tk UI	18,294	Tcl
ProB2-API	25,232	Java
ProB2-UI	31,677	Java
	2302	FXML
	3120	properties



+ C/C++ extensions

ProB Components for interacting with other languages



Analysis and Translation Rules for other Languages in Prolog

Operational Semantics: Process Algebras (CSP)

$$\frac{}{a \to Y \overset{a}{\rightsquigarrow} Y} \qquad \frac{X \overset{a}{\rightsquigarrow} X'}{X \| Y \overset{a}{\rightsquigarrow} X' \| Y} \qquad \frac{Y \overset{a}{\rightsquigarrow} Y'}{X \| Y \overset{a}{\rightsquigarrow} X \| Y'}$$

- trans('->'(A,Y),A,Y).
- trans('||'(X,Y),A,'||'(X2,Y)) :- trans(X,A,X2).
- trans('||'(X,Y),A,'||'(X,Y2)) :- trans(Y,A,Y2).

```
| ?- trans('||'('->'(a,stop),'->'(b,stop)),A,R).
A = a,
R = '||'(stop,(b->stop)) ? ;
A = b,
R = '||'((a->stop),stop) ? ;
no
```

Teaching: A Slide about Big-Step Inference Rules

- eint(X:=V,In,Out) :- eval(V,In,Res),store(In,X,Res,Out).

$$\frac{\langle E, \sigma\rangle \Rightarrow V}{\sigma \rightsquigarrow_{x:=E} \sigma \lessdot \{x \mapsto V\}}\ x \in dom(\sigma)$$

- eint('def'(X),In,Out) :- def(In,X,Out).

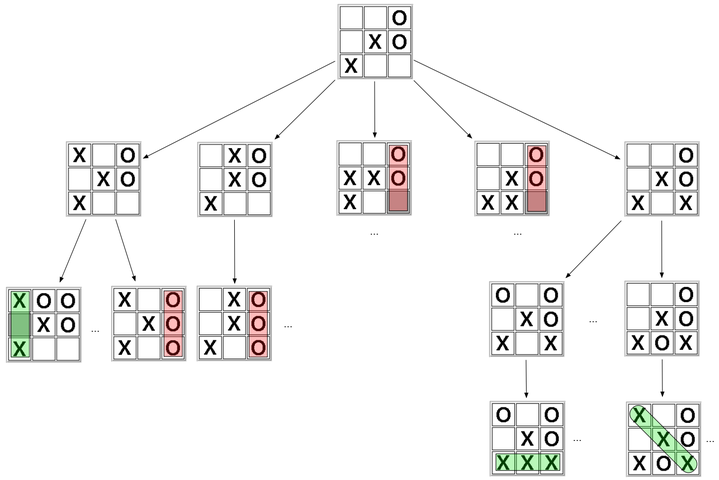
$$\frac{}{\sigma \rightsquigarrow_{def x} \sigma \lessdot \{x \mapsto \bot\}}\ x \notin dom(\sigma)$$

- eint((X;Y),In,Out) :- eint(X,In,I2), eint(Y,I2,Out).

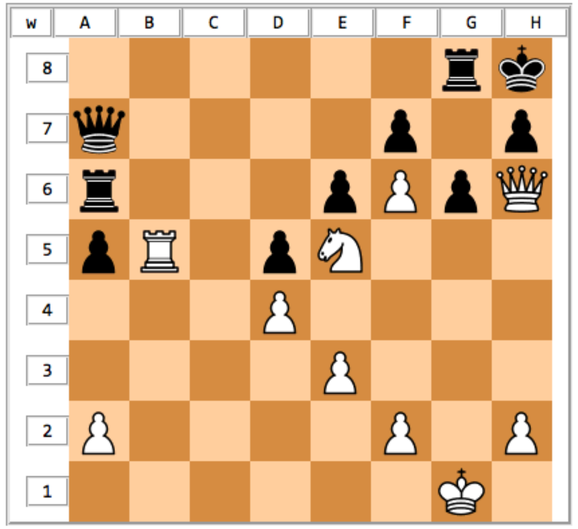
$$\frac{\sigma \rightsquigarrow_x \sigma', \sigma' \rightsquigarrow_y \sigma''}{\sigma \rightsquigarrow_{x;y} \sigma''}$$

cf. talk
by John Gallagher

Teaching: AI



- Prolog can be used to encode transition rules of puzzles or games to illustrate various symbolic AI algorithms:
  - IDS, Greedy Search, A* Search
  - Minimax, Alpha-Beta
  - …



chess.P - ProB2Tests - ProB 2.0

Operations

Filter Operations

move(pawn(black), 1, 5, false, 1, 4)
move(pawn(black), 7, 6, false, 7, 5)
move(queen(black), 1, 7, false, 1, 8)
move(queen(black), 1, 7, false, 2, 6)
move(queen(black), 1, 7, false, 2, 7)
move(queen(black), 1, 7, false, 3, 5)
move(que
Reaches unexplored state
move(queen(black), 1, 7, false, 4, 7)
move(queen(black), 1, 7, false, 5, 7)
move(queen(black), 1, 7, pawn(white), 4, 4)
move(queen(black), 1, 7, rook(white), 2, 8)
move(rook(black), 1, 6, false, 2, 6)
move(rook(black), 1, 6, false, 3, 6)
move(rook(black), 1, 6, false, 4, 6)
move(rook(black), 7, 8, false, 3, 8)
move(rook(black), 7, 8, false, 4, 8)
move(rook(black), 7, 8, false, 5, 8)
move(rook(black), 7, 8, false, 6, 8)
move(rook(black), 7, 8, rook(white), 2, 8)
unspecified_operation

Animation

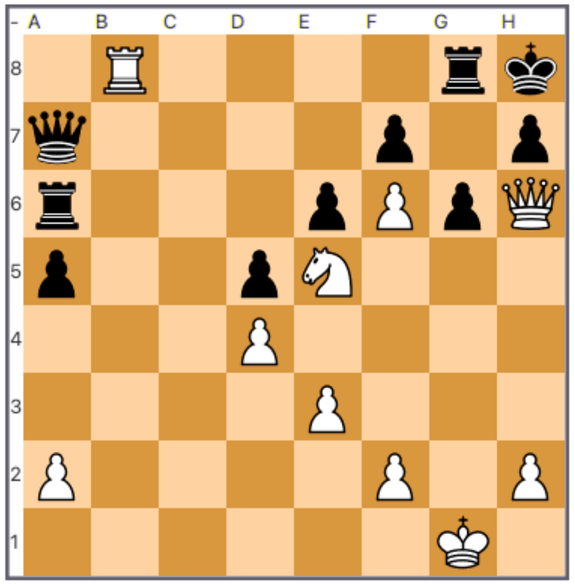
Replay | Symbolic | Test Case Generation

Status | Name

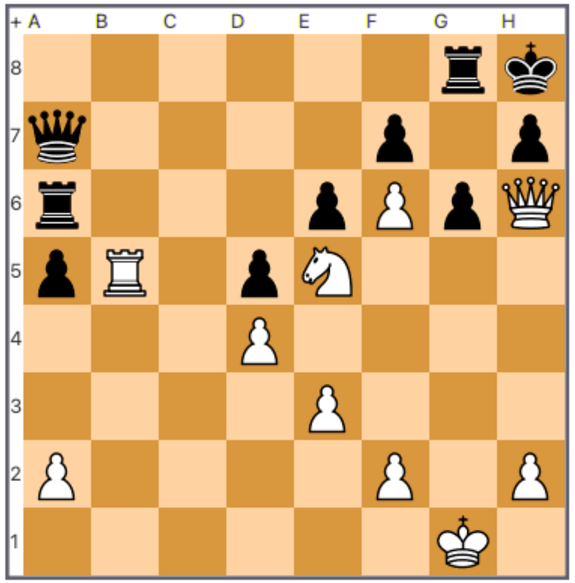
No Traces

State View | State Visualisation | Edit

Current State



Previous State



Interactive Console

Statistics (states 5 of 106)

Verifications

Project

Machines | Status | Preferences | Project

public_examples/CSP/mydemos/demo/crossing

BauerZiege
public_examples/XTL/BauerZiege.P

Connect4
public_examples/XTL/Connect4.P

chess (Timeout5)
public_examples/XTL/chess.P

NonDetChoiceOpCallTest
public_examples/B/Tester/NonDetChoiceOpCa

Ref4_ControllerHandle_mch
public_examples/EventBPrologPackages/ABZ_

Bakery0
public_examples/B/Demo/Bakery0.mch

Generated1000
public_examples/B/PerformanceTests/Generat

Generated2000

History (state 3 of 3)

Position ▲	Transition
0	---root---
1	start_xtl_system
2	PUZZLE(1)
3	**auto_play(4, 1260, move(ro...**

Everything is OK

chess.P - ProB2Tests - ProB 2.0

Operations

Filter Operations

auto_play(4, 1290, none)

Animation

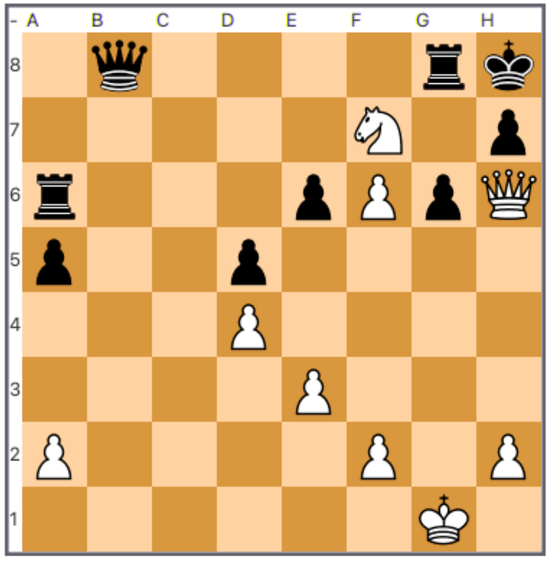
Replay | Symbolic | Test Case Generation

Status | Name

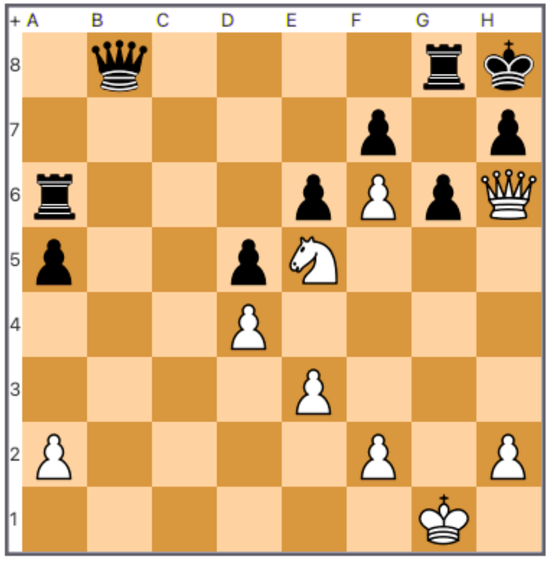
No Traces

State View | State Visualisation | Edit

Current State



Previous State



Interactive Console

Statistics (states 11 of 283)

Verifications

Project

Machines | Status | Preferences | Project

public_examples/CSP/mydemos/demo/crossing

BauerZiege
public_examples/XTL/BauerZiege.P

Connect4
public_examples/XTL/Connect4.P

chess (Timeout5)
public_examples/XTL/chess.P

NonDetChoiceOpCallTest
public_examples/B/Tester/NonDetChoiceOpCa

Ref4_ControllerHandle_mch
public_examples/EventBPrologPackages/ABZ_

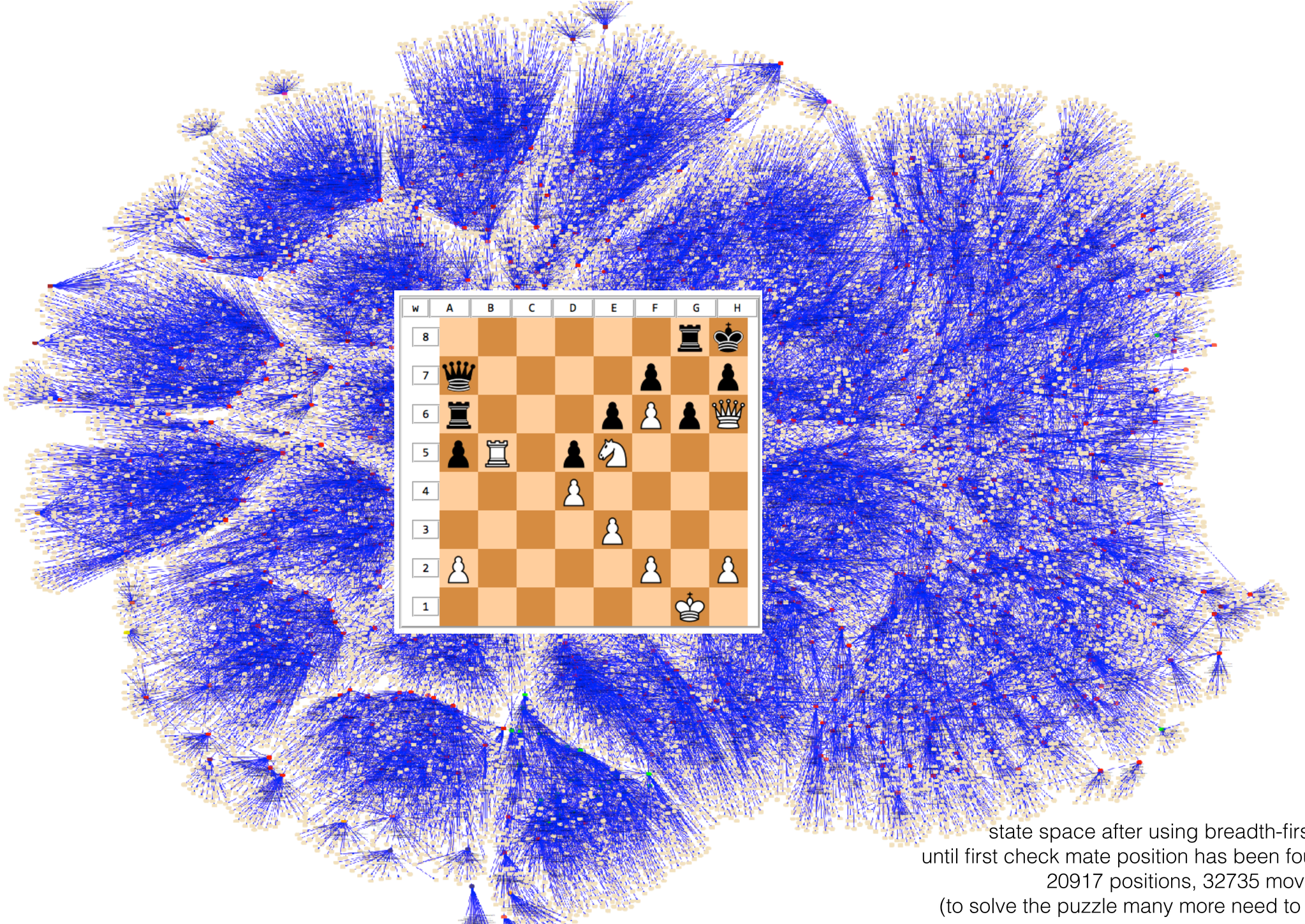
Bakery0
public_examples/B/Demo/Bakery0.mch

Generated1000
public_examples/B/PerformanceTests/Generat

History (state 5 of 5)

Position ▲	Transition
0	---root---
1	start_xtl_system
2	PUZZLE(1)
3	auto_play(4, 1260, move(roo...
4	move(queen(black), 1, 7, roo...
5	**auto_play(4, 1280, move(k...**

Everything is OK



state space after using breadth-first search
until first check mate position has been found after 4 plys:
20917 positions, 32735 moves
(to solve the puzzle many more need to be analysed!)

Hindley-Milner Type Inference

- Atelier-B requires left-to-right inference
- Easy to encode in Prolog
- cf. article

```
type([],set(_)) --> !, [].
type(union(A,B),set(R)) --> !,type(A,set(R)), type(B,set(R)).
type(intersect(A,B),set(R)) --> !,type(A,set(R)), type(B,set(R)).
type(plus(A,B),integer) --> !,type(A,integer), type(B,integer).
type(in_set(A,B),predicate) --> !,type(A,TA), type(B,set(TA)).
type(gt(A,B),predicate) --> !,type(A,integer), type(B,integer).
type(and(A,B),predicate) --> !,type(A,predicate),type(B,predicate).
type(eq(A,B),predicate) --> !,type(A,TA),type(B,TA).
type(Nr,integer) --> {number(Nr)},!.
type([H|T],set(TH)) --> !,type(H,TH), type(T,set(TH)).
type(ID,TID) --> {identifier(ID)},\+ defined(id(ID,_)),!,
                  add((id(ID,TID))). % creates fresh variable
type(ID,TID) --> {identifier(ID)},defined(id(ID,TID)),!.
type(Expr,T,Env,_) :-
              format('Type error for ~w (expected: ~w, Env: ~w)~n',[Expr,T,Env]),fail.

defined(X,Env,Env) :- member(X,Env).
add(X,Env,[X|Env]).

identifier(ID) :- atom(ID), ID \= [].

type(Expr,Result) :- type(Expr,Result,[],Env), format('Typing env: ~w~n',[Env]).
```

$$\{z\} \cup \{x, y\} = y \wedge z > v$$

```
| ?- type(and(eq(union([z],[x,y]),u),gt(z,v)),R).
Typing env: [id(v,integer),id(u,set(integer)),id(y,integer),id(x,integer),id(z,integer)]
R = predicate ?
yes
```

What about negation?

```
int(const(true)).
int(and(X,Y)) :- int(X), int(Y).
int(or(X,Y)) :- int(X) ; int(Y).
int(not(X)) :- \+ int(X).
```

Interpreter for propositional logic

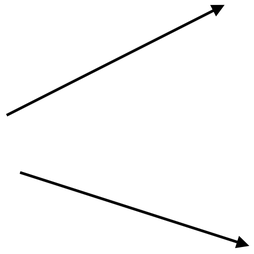
- Prolog uses negation as failure (\+)
- Many semantic rules rely on classical negation
  - CSP
  - B, TLA+, Alloy, …: all rooted in classical predicate logic
- How to implement negation in the interpreter?

Poor man’s constructive negation

- For treating predicates at the object level: two Prolog predicates
  - one for finding solutions
  - one for finding counter examples (solutions of the negation)

Interpreter for propositional logic

```
int(const(true)).
int(and(X,Y)) :- int(X), int(Y).
int(or(X,Y)) :- int(X) ; int(Y).
int(not(X)) :- \+ int(X).
```



```
int(const(true)).
int(and(X,Y)) :- int(X), int(Y).
int(or(X,Y)) :- int(X) ; int(Y).
int(not(X)) :- int_neg(X).
```

```
int_neg(const(false)).
int_neg(and(X,Y)) :- int_neg(X); int_neg(Y).
int_neg(or(X,Y)) :- int_neg(X) , int_neg(Y).
int_neg(not(X)) :- int(X).
```

Experience Report:
Prolog vs Java
for Analysis and Verification

Experience Report: Prolog vs Java for Analysis and Verification

Executive Summary

Semantic Translation Rules: Alloy2B

- Translator of Alloy [Jackson] to B
  - Adaptation of formal semantics of Alloy by simply using B syntax
  - Rules can be translated to Prolog clauses
- First version was written in Kotlin, then switched to Prolog as error prone and tedious to encode rules

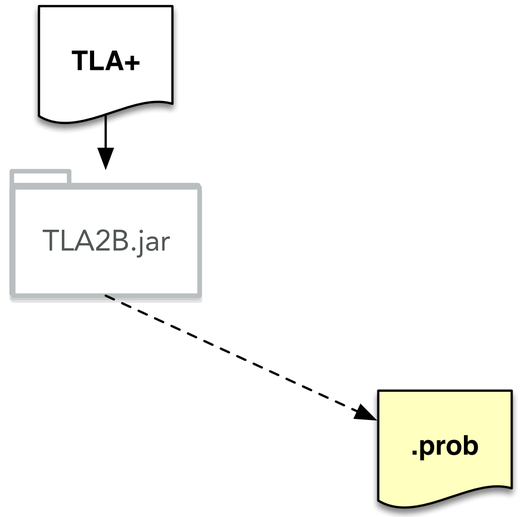
$$E[\![p + q]\!]i \;\widehat{=}\; E[\![p]\!]i \cup E[\![q]\!]i$$

$$E[\![p \;\&\; q]\!]i \;\widehat{=}\; E[\![p]\!]i \cap E[\![q]\!]i$$

$$E[\![p - q]\!]i \;\widehat{=}\; E[\![p]\!]i \setminus E[\![q]\!]i$$

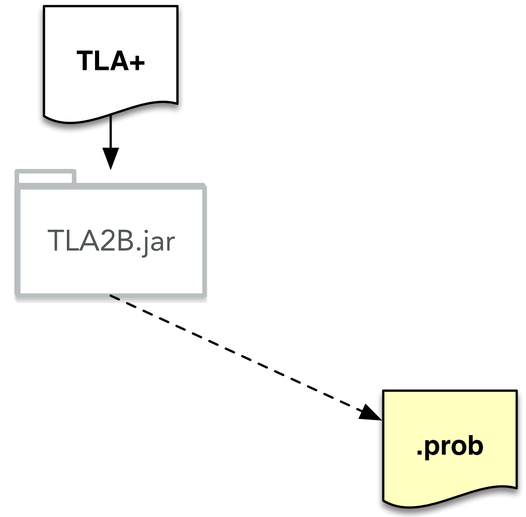
```
translate_binary_e_p(Binary, TBinary) :-
    Binary =.. [Op,P,Q,_,POS],
    alloy_to_b_binary_operator(Op, BOp),
    translate_e_p(P, TP),
    translate_e_p(Q, TQ),
    translate_pos(POS, BPOS),
    TBinary =.. [BOp,BPOS,TP,TQ].

alloy_to_b_binary_operator(plus, union).
alloy_to_b_binary_operator(intersection, intersection).
alloy_to_b_binary_operator(minus, set_subtraction).
alloy_to_b_binary_operator(implication, implication).
alloy_to_b_binary_operator(iff, equivalence).
…
```

TLA2B



- Translator from TLA+ [Lamport] to B for ProB
- Uses SANY parser (in Java)
- Translation rules also written in Java
  - I did not manage to coerce my student to write it in Prolog

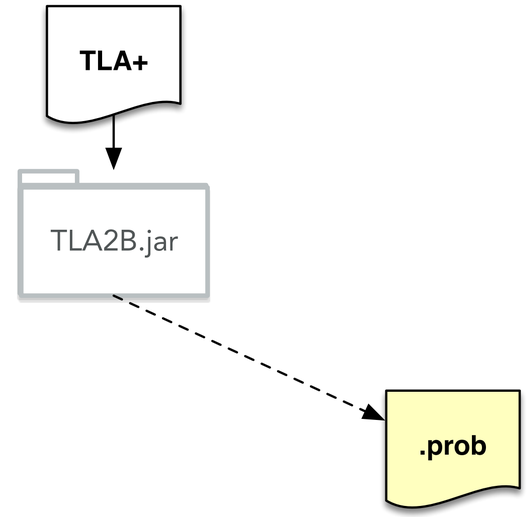
AST Traversal 1




```
sepGuards(land(A,B),...) :- sepGuards(A,...), sepGuards(B,...).
```

```
private void separateGuardsAndBeforeAfterPredicates(ExprOrOpArgNode node) {
    if (node instanceof OpApplNode) {
        OpApplNode opApplNode = (OpApplNode) node;
        if (opApplNode.getOperator().getKind() == BuiltInKind) {
            switch (getOpCode(opApplNode.getOperator().getName())) {
            case OPCODE_land: // \land
            {
                separateGuardsAndBeforeAfterPredicates(opApplNode.getArgs()[0]);
                separateGuardsAndBeforeAfterPredicates(opApplNode.getArgs()[1]);
                return;
```

AST Traversal 2



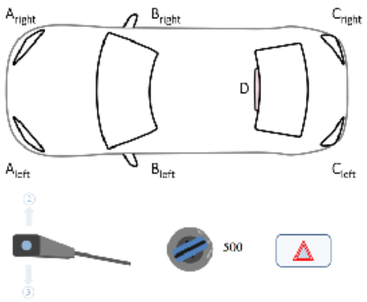

```
find_assign(eq(id(XPrime),RHS)) :- primed_id(XPrime,X), …

find(BAPres,Res) :- include(find_assign,BAPreds,Res).
```

```
private void findAssignments() {
    PrimedVariablesFinder primedVariablesFinder = new PrimedVariablesFinder(
            beforeAfterPredicates);
    for (ExprOrOpArgNode node : new ArrayList<ExprOrOpArgNode>(
            beforeAfterPredicates)) {
        if (node instanceof OpApplNode) {
            OpApplNode opApplNode = (OpApplNode) node;
            if (opApplNode.getOperator().getKind() == BuiltInKind) {

                if (OPCODE_eq == getOpCode(opApplNode.getOperator()
                        .getName())) {
                    ExprOrOpArgNode arg1 = opApplNode.getArgs()[0];
                    try {
                        OpApplNode arg11 = (OpApplNode) arg1;
                        if (getOpCode(arg11.getOperator().getName()) == OPCODE_prime) {
                            OpApplNode v = (OpApplNode) arg11.getArgs()[0];
                            SymbolNode var = v.getOperator();
```

VisB

- After getting annoyed with Java file hopping, and implementing language features (repeat constructs)
  - Reimplementation in Prolog; core functionality working in an afternoon
  - More powerful features, better error reporting
  - New features easier to implement: language features (definitions), HTML/JavaScript Export

VisB Architecture



SVG Graphics File

object ids and attributes

VisB JSON Glue File

expressions over variables, constants and events

Formal Model

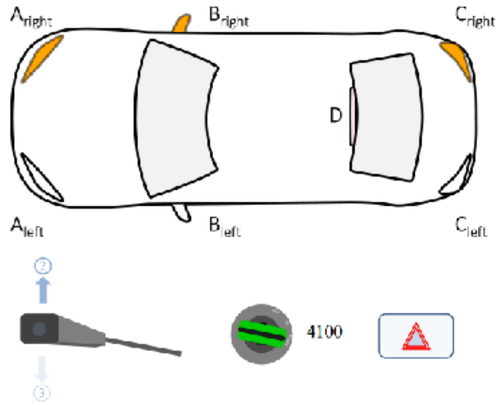
VisB

Evaluation of Formulas from Glue File

ProB2 Animator UI

Setting Attributes and On-click Callbacks

Modified SVG Graphics



Setter/Getter Ceremony

`visb_event(Id,Ev,Preds,Hovers)`

Prolog

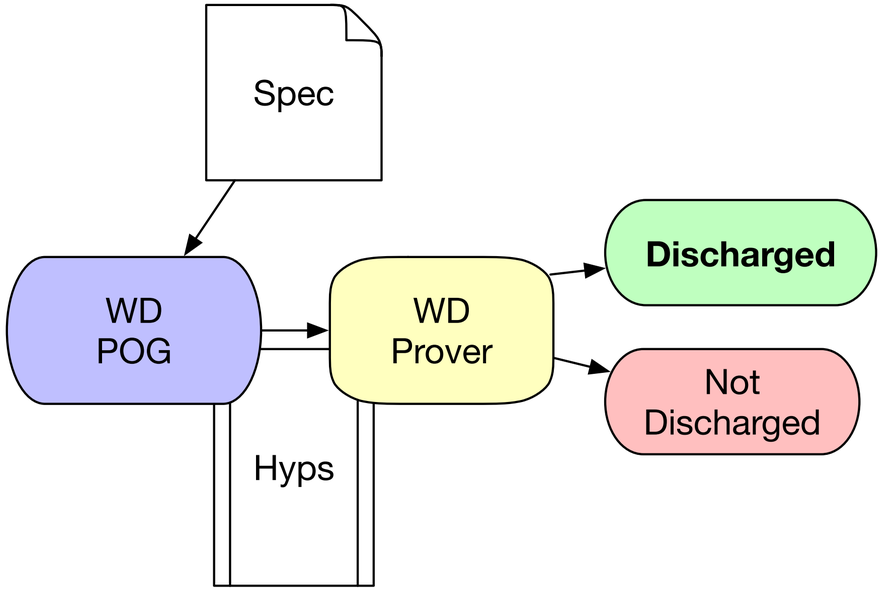
```
package de.prob.animator.domainobjects;

import de.prob.parser.BindingGenerator;
import de.prob.prolog.term.PrologTerm;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class VisBEvent {
    private String id;
    private String event;
    private List<String> predicates;
    private List<VisBHover> hovers;

    public VisBEvent(String id, String event,
            List<String> predicates, List<VisBHover> hovers){
        this.id = id;
        this.event = event;
        this.predicates = predicates;
        this.hovers = hovers;
    }

    public String getEvent() {
        return event;
    }

    public List<String> getPredicates() {
        return predicates;
    }

    public String getId() {
        return id;
    }
```

Java

```
    public List<VisBHover> getHovers() {
        return hovers;
    }

    public static VisBEvent fromPrologTerm(final PrologTerm te
final Map<String, List<VisBHover>> hoverMap) {
        BindingGenerator.getCompoundTerm(term, "execute_event"
3);
        final String id =
PrologTerm.atomicString(term.getArgument(1));
        final String event =
PrologTerm.atomicString(term.getArgument(2));
        final List<String> predicates =
PrologTerm.atomicStrings(BindingGenerator.getList(term.getArgu
(3)));
        final List<VisBHover> hovers = hoverMap.get(id) == nul
new ArrayList<>() : hoverMap.get(id);
        return new VisBEvent(id, event, predicates, hovers);
    }

    @Override
    public String toString(){
        return "ID: "+id+"\nEVENT: "+event+"\nPREDICATES:
"+predicates+"\n";
    }
}
```

Prolog Theorem Prover for Proving Well-Definedness

- WD Prover [iFM'2020] ot prove absence of division by zero, undefined function applications, cardinality of infinite sets, …
- Shared Hypothesis stack
  - pop via **Prolog backtracking**
  - Only **logarithmic** accesses to Hypotheses
  - Efficient **rule-based prover** using **Prolog unification**



Patterns for Lookups	
$finite(A)$	$A \in B$
$A = B$	$A \neq B$
$A \leq B$	$A \geq B$
$A \subseteq B$	$A \supseteq B$

WD Prover

One WD Prover Rule

“The range of a function is finite if the function is finite.”

```
check_finite(range(A),Hyp,ran(PT)) :- !,
                        check_finite(A,Hyp,PT)
```

Prolog

In Java:
82 lines of code
(9 lines are copyright notice)
The Prolog code is also very flexible:
it can be used for finding proofs
but also for re-playing or
checking proofs if the proof tree argument is provided

Rodin

```
/*******************************************************************************
 * Copyright (c) 2007, 2014 ETH Zurich and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     ETH Zurich - initial API and implementation
 *******************************************************************************/
package org.eventb.internal.core.seqprover.eventbExtensions;

import org.eventb.core.ast.Expression;
import org.eventb.core.ast.FormulaFactory;
import org.eventb.core.ast.Predicate;
import org.eventb.core.ast.SimplePredicate;
import org.eventb.core.ast.UnaryExpression;
import org.eventb.core.seqprover.IProofMonitor;
import org.eventb.core.seqprover.IProverSequent;
import org.eventb.core.seqprover.IReasonerInput;
import org.eventb.core.seqprover.IReasonerOutput;
import org.eventb.core.seqprover.ProverFactory;
import org.eventb.core.seqprover.ProverRule;
import org.eventb.core.seqprover.SequentProver;
import org.eventb.core.seqprover.IProofRule.IAntecedent;
import org.eventb.core.seqprover.eventbExtensions.Lib;
import org.eventb.core.seqprover.reasonerInputs.EmptyInputReasoner;

public class FiniteRan extends EmptyInputReasoner {

	public static final String REASONER_ID = SequentProver.PLUGIN_ID + ".finiteRan";

	@Override
	public String getReasonerID() {
		return REASONER_ID;
	}

	@ProverRule("FIN_REL_RAN_R")
	protected IAntecedent[] getAntecedents(IProverSequent seq) {
		Predicate goal = seq.goal();

		// goal should have the form finite(ran(r))
		if (!Lib.isFinite(goal))
			return null;
		SimplePredicate sPred = (SimplePredicate) goal;
		if (!Lib.isRan(sPred.getExpression()))
			return null;

		// There will be 1 antecidents
		IAntecedent[] antecidents = new IAntecedent[1];

		UnaryExpression expression = (UnaryExpression) sPred.getExpression();

		Expression r = expression.getChild();

		final FormulaFactory ff = seq.getFormulaFactory();
		// finite(r)
		Predicate newGoal = ff.makeSimplePredicate(Predicate.KFINITE, r, null);
		antecidents[0] = ProverFactory.makeAntecedent(newGoal);

		return antecidents;
	}

	protected String getDisplayName() {
		return "finite of range of a relation";
	}

	@Override
	public IReasonerOutput apply(IProverSequent seq, IReasonerInput input,
			IProofMonitor pm) {
		IAntecedent[] antecidents = getAntecedents(seq);
		if (antecidents == null)
			return ProverFactory.reasonerFailure(this, input,
					"Inference " + getReasonerID()
							+ " is not applicable");

		// Generate the successful reasoner output
		return ProverFactory.makeProofRule(this, input, seq.goal(),
				getDisplayName(), antecidents);
	}
```

Java

https://sourceforge.net/p/rodin-b-sharp/rodincore/ci/master/tree/org.eventb.core.seqprover/src/org/eventb/internal/core/seqprover/eventbExtensions/FiniteRan.java

WD Prover

```
check_finite(range(A),Hyp,ran(PT)) :- !,
                check_finite(A,Hyp,PT)
```

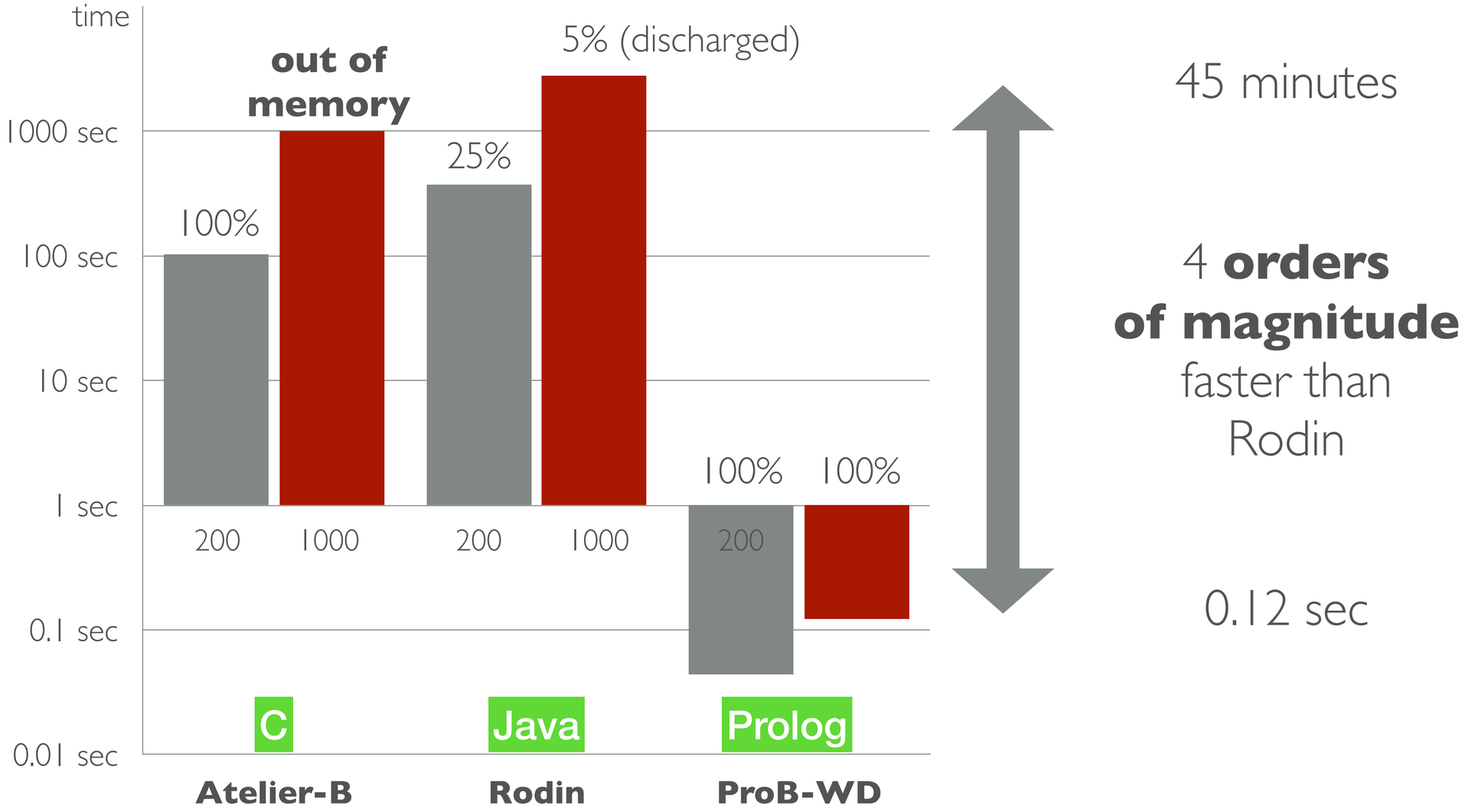
Prolog

```
public class FiniteRan extends EmptyInputReasoner {

    public static final String REASONER_ID = SequentProver.PLUGIN_

    @Override
    public String getReasonerID() {
        return REASONER_ID;
    }

    @ProverRule("FIN_REL_RAN_R")
    protected IAntecedent[] getAntecedents(IProverSequent seq) {
        Predicate goal = seq.goal();

        // goal should have the form finite(ran(r))
        if (!Lib.isFinite(goal))
            return null;
        SimplePredicate sPred = (SimplePredicate) goal;
        if (!Lib.isRan(sPred.getExpression()))
            return null;

        // There will be 1 antecidents
        IAntecedent[] antecidents = new IAntecedent[1];

        UnaryExpression expression = (UnaryExpression) sPred.getEx

        Expression r = expression.getChild();

        final FormulaFactory ff = seq.getFormulaFactory();
        // finite(r)
        Predicate newGoal = ff.makeSimplePredicate(Predicate.KFINI
        antecidents[0] = ProverFactory.makeAntecedent(newGoal);
```

Java

Artificial Benchmarks (Scaling)

```
MACHINE FunNrWD
CONSTANTS ff
PROPERTIES
    /* axm0 */ ff : 1 .. Nr --> 1 .. 90
  & /* axm1 */ ff(1) < 100
  & /* axm2 */ ff(2) < 100
  ...
  & /* axmNr */ ff(Nr) < 100
  & /* axm_nest_1 */ ff(ff(1)) < 100
  ...
  & /* axm_nest_Nr */ ff(ff(Nr)) < 100
INITIALISATION skip
END
```



45 minutes

4 **orders of magnitude** faster than Rodin

0.12 sec

Conclusions

Why not Prolog

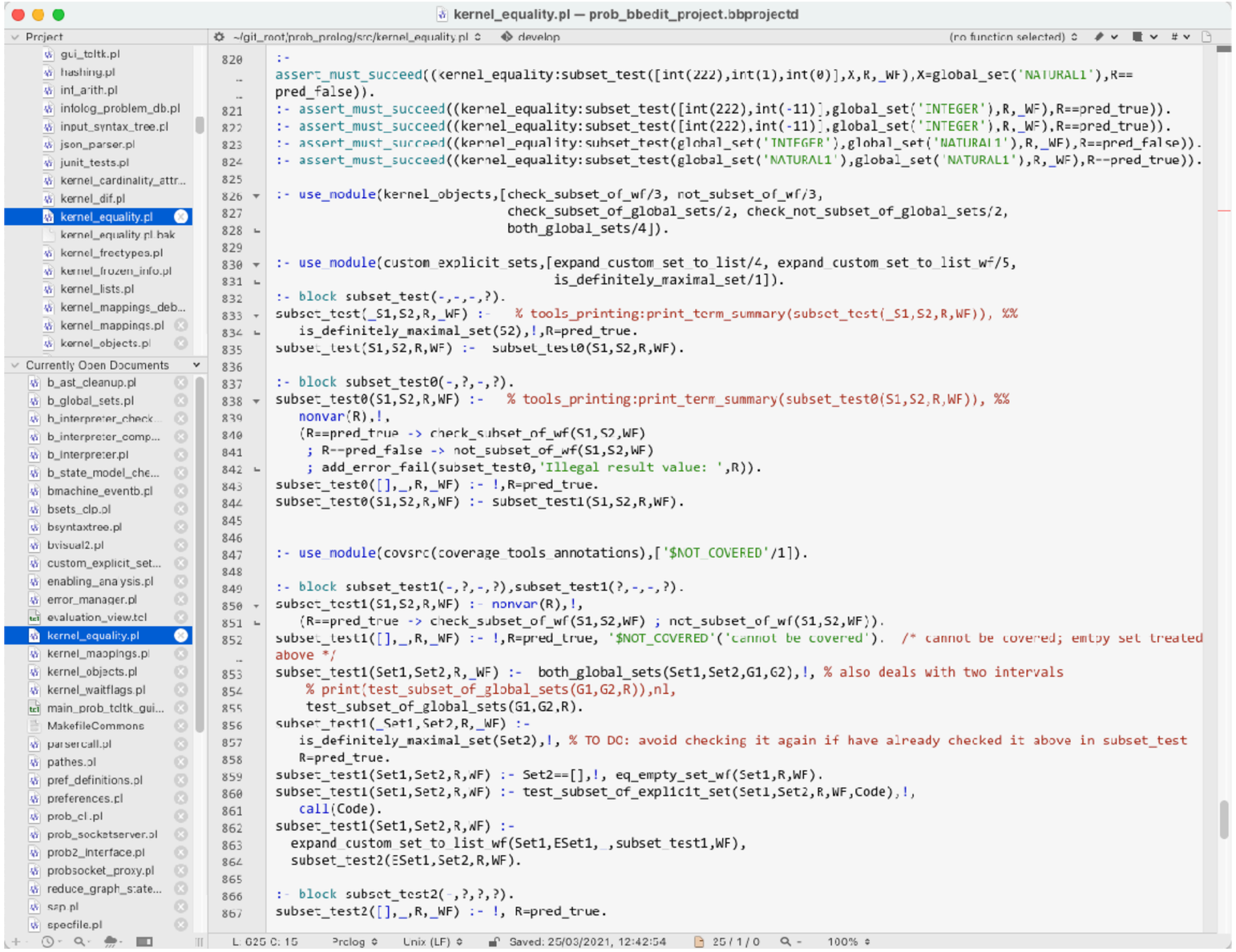
- No static type checker (but there are no NullPointerExceptions either)

Why not Prolog

- No static type checker (but there are no NullPointerExceptions either)
- Few standard libraries (e.g., no regexp library)
- Limited support for parallel execution (at least in SICStus Prolog; we use ZMQ in ProB)
- Performance: some things are inefficient (C++ priority queue in ProB for directed model checking, loop checking for LTL done in C)
- Limited IDE support

My Prolog IDE

BBEdit
with custom
language module



Prolog - SWOT

Strengths of Prolog and its implementations:

- clean, simple language syntax and semantics
- immutable persistent data structures, with "declarative" pointers (logic variables)
- arbitrary precision arithmetic
- safety (garbage collection, no Null-Pointer exceptions, ...)
- tail-recursion and last-call optimization
- efficient inference, pattern matching, and unification, DCGs
- meta-programming, programs as data
- constraint solving (3.3.3), independence of the selection rule (co-routines (3.3.2))
- indexing (3.3.2), efficient tabling (3.3.2)
- fast development, REPL (Read, Execute, Print, Loop), debugging (3.3.4)
- commercial (2.9.2) and open-source systems
- sophisticated tools: analyzers, partial evaluators, parallelizers, ...
- successful applications
  - program analysis,
  - domain-specific languages (Erlang (Armstrong 2007))
  - heterogeneous data integration
  - NLP (Watson (Lally et al. 2012))
  - efficient inference (expert systems, theorem provers), symbolic AI
- many books, courses and learning materials

Weaknesses of Prolog and its implementations:

- syntactically different from "traditional" programming languages, not a mainstream language
- learning curve, beginners can easily write programs that loop or consume a huge amount of resources
- static typing (see, however, 3.3.2)
- data hiding (see, however, 3.3.1)
- object orientation (see, however, 3.3.1)
- limited portability (see 4.4.1, 4.6)
- packages: availability and management
- IDEs: limited capabilities in some areas (e.g., refactoring)
- UI development (usually conducted in a foreign language via FLI (3.3.1))

Opportunities for Prolog:

- new application areas, addressing societal challenges 4.2:
  - neuro-Symbolic AI
  - explainable AI, verifiable AI
  - The Semantic Web
- new features and developments
  - probabilistic reasoning (3.4)
  - embedding ASP (3.4)
  - parallelism (2.7, 3.3.2) (resurrecting 80s, 90s research)
  - full-fledged JIT compiler

Threats to the existence or relevance of Prolog:

- comparatively small user base
- fragmented community with limited interactions (e.g., on StackOverflow, reddit), see 4.3
- further fragmentation of Prolog implementations, see 4.5 and 4.6
- new programming languages
- post-desktop world of JavaScript web-applications
- the perception that it is an "old" language

Source: Körner et al.
A Multi-Walk Through the Past, Present and Future of Prolog
(Submitted to TPLP)

Why Prolog

- Non-determinism and unification for encoding semantic / translation rules
- Co-routines (block/when)
- Constraints, in particular CLP(FD)
- Tabling for static analysis
- Partial evaluation for processing semantic rules
- Performance

Conclusion

- Prolog is a great language for writing tools for verification and transformation of programs
  - it is possible to write very ugly code, but
  - it is also possible to write elegant interpreters or encodings of PL semantics (cf earlier talk by John Gallagher)
- There are great Prolog compilers available: Ciao Prolog, SICStus Prolog, SWI Prolog, XSB Prolog, …
- There are a lot of useful tools for Prolog programs

Danny De Schreye
Maurice Bruynooghe
Bart Demoen
Marc Denecker
Bern Martens
Wim Vanhoof

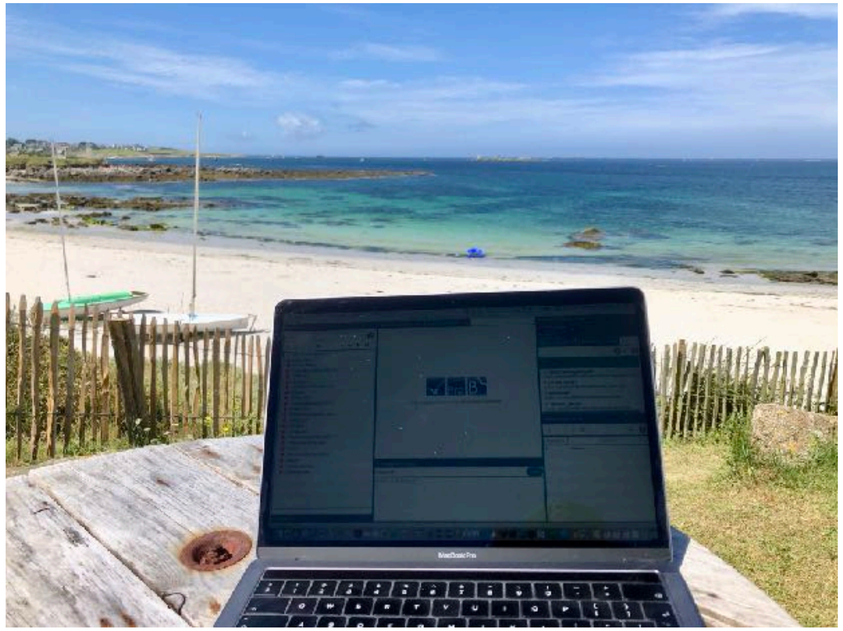
Robert Glück
Neil D. Jones
Jesper Jørgensen
Torben Mogensen

Fabio Fioravanti
Alberto Pettorossi
Maurizio Proietti
Emanuele De Angelis

John Gallagher
Manual Hermenegildo
German Puebla
Josep Silva
Salvador Tamarit
German Vidal

Kostis Sagonas
and many more

Thanks!



Thanks to

Jens Bendisposto
Carl Friedrich Bolz
Michael Butler
Joy Clark
Ivo Dobrikov
Jannik Dunkelau
Nadine Elbeshausen
Fabian Fritz
Marc Fontaine
Marc Frappier
David Gelessus
Stefan Hallerstede
Dominik Hansen
Christoph Heinzen
Michael Jastram
Philip Körner
Sebastian Krings
Lukas Ladenberger
Li Luo
Thierry Massart
Daniel Plagge

Antonia Pütz
Mireille Samia
Joshua Schmidt
David Schneider
Corinna Spermann
Yumiko Takahashi
Edd Turner
Michelle Werth
Dennis Winter
Fabian Vu

Alstom (Fernando Mejia,…)
ClearSy (Thierry Lecomte,…)
Siemens
Systerel
Thales (Nader Nayeri,…)