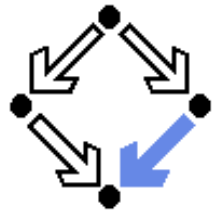
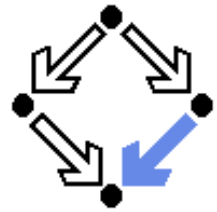
The RISC ProofNavigator
A First Status Report

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at



Motivation

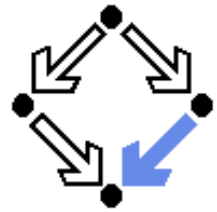


Vision: integrated environment for program/system design and reasoning.

- Modular environment.
  - Various kinds of programming/system models.
  - Various kinds of program/system specifications.
  - Computer support for reasoning about program/system designs.
    - Help human to investigate and understand their designs.
    - Understand why design has or does not have specific properties.
- Computer-aided reasoning modules.
  - Specifications and designs generate verification conditions.
    - Simpler conditions can be discharged automatically.
    - More complex conditions need human/system interaction.
    - Results are more or less trusted condition certificates (proofs).
  - Certified conditions may become input for code generation.
    - Additional knowledge that compiler cannot derive on its own.

First step: investigate interactive reasoning environments.

Interactive Reasoning Environments

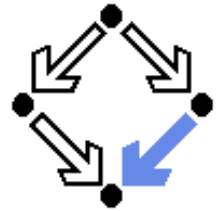


Last semester: investigation of PVS, Isabelle/HOL, Coq.

- Investigation on the basis of concrete test cases.
  - Reasoning about datatype "array" (specified axiomatically and constructively), verification of linear search, verification of binary search, verification of mutual exclusion in a client/server system.
- Presentations in the "Formal Methods Seminar".
  - "The Prototype Verification System PVS", May 2005.
  - "The Theorem Proving Environments Isabelle and Coq: A Personal Experience", October 2005.

An analysis based on personal "hands-on" experience.

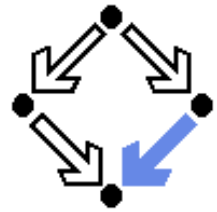
Interactive Reasoning Environments (Contd)



Some observation made.

- Proof state simplification and presentation is crucial.
  - Human gets easily overwhelmed by large proof state.
- Proof navigation is crucial.
  - Human gets easily lost in a proof.
  - Mantra: a proof is a tree is a tree is a tree . . . (not a sequence)!
- Most valuable was automation of low-level steps.
  - Equality reasoning, linear arithmetic, propositional logic, application of elimination rules to assumptions and of introduction rules to goals.
- Automatic proof search was not of much practical help.
  - Only yielded results in the final steps of a proof branch.
- Only few kinds of creative user activities were crucial.
  - Which definitions to expand, which lemmas to use (and to prove).
  - How to cut proofs by assumptions and case distinctions.
  - How to instantiate universal assumptions and existential goals.

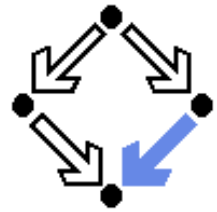
Personal Preferences



How should an environment (for me!) look like?

- Do not overwhelm me with choices and options.
  - Small set of commands whose effect is easily understandable.
  - Better remove a command than add one.
- Do not allow genericity to make simple things complicated.
  - Builtin knowledge of basic arithmetic and classical logic.
- Perform tasks in which you are good.
  - Decision procedures for various theories.
  - Standard steps in proof decomposition.
  - (At most) shallow proof search.
- Do not attempt tasks in which you are not (yet) good.
  - Deep proof search (unless in background).
    - Maybe different some day...

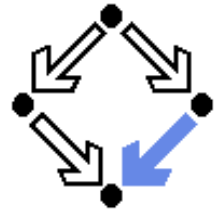
Personal Preferences (Contd)



- Allow me to guide the proof in a convenient way.
  - Commands for the creative proof steps.
- Help me in exploring the proof.
  - Aggressive proof state simplification.
  - Proof state display that scales to a large number of formulas.
  - Proof navigation that scales to a large number of states.
- Be open for customization.
  - Communication interfaces based on established standards.
  - Programming library with well-documented API.
  - Open source in a language that many people use and for which well-supported development tools exist.

Well, let me not only criticize but do something...

The RISC ProofNavigator 0.1



An attempt at an interactive proof assistant that suits my preferences.

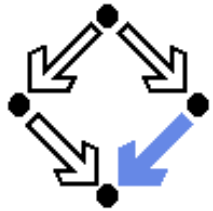
- Program/library being developed in Java (since August 2005).
  - Specification language similar to PVS.
  - Engine for representing and handling proofs.
  - Interface to the validity checker CVC Lite (CVCL).
  - GUI for proof state display and proof navigation.
  - Storage representation of proofs in OMDoc/OpenMath.
  - Proof presentation in XHTML/MathML.
- Web page

  `http://www.risc.uni-linz.ac.at`
  `/research/formal/software/ProofNavigator`

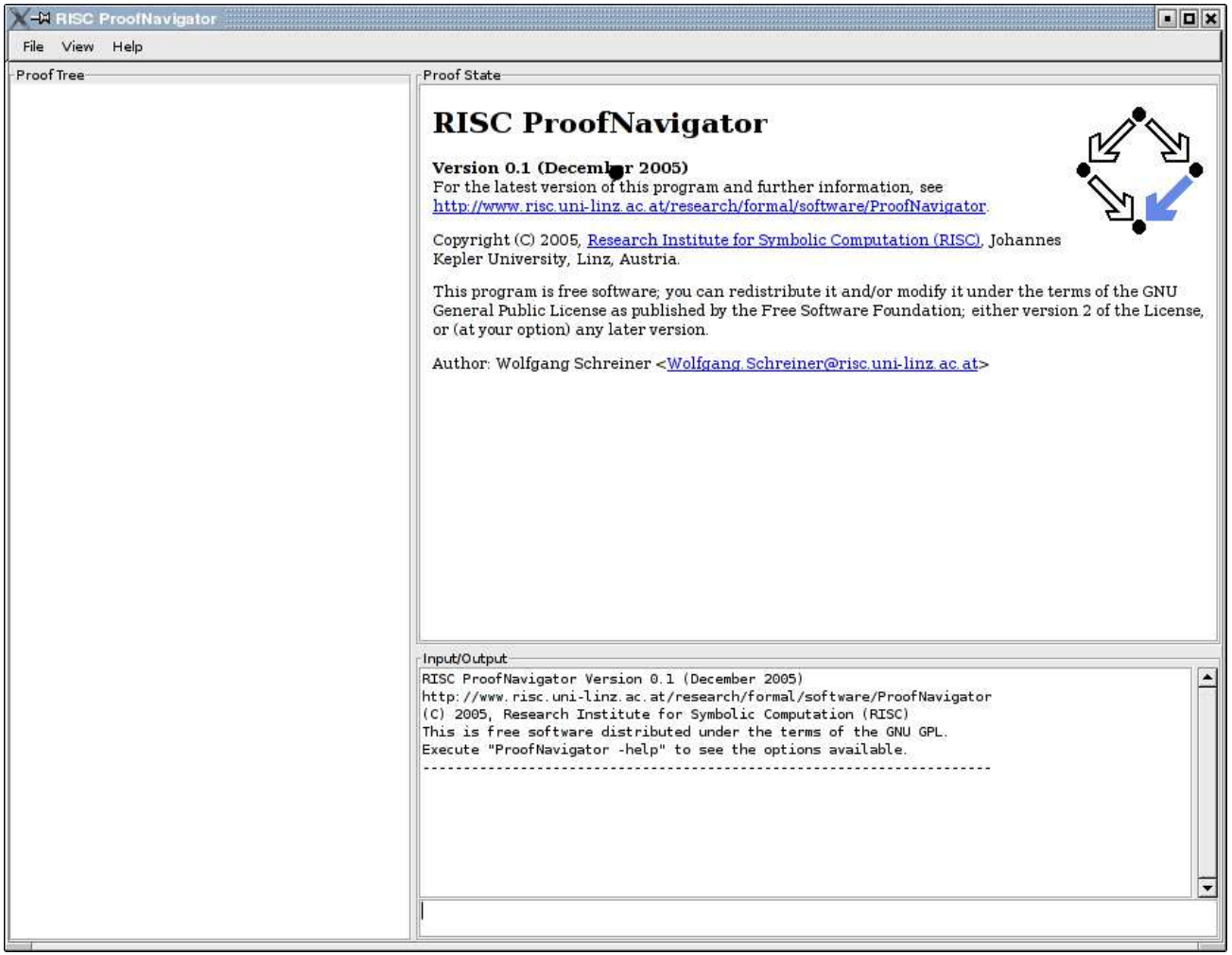
  - Binary distribution for GNU/Linux ix86 computers.
  - Source anonymously available from Subversion repository.
  - JavaDoc documentation for the program classes.
  - No user documentation yet.

Still alpha state, evaluation with my test cases will start soon.

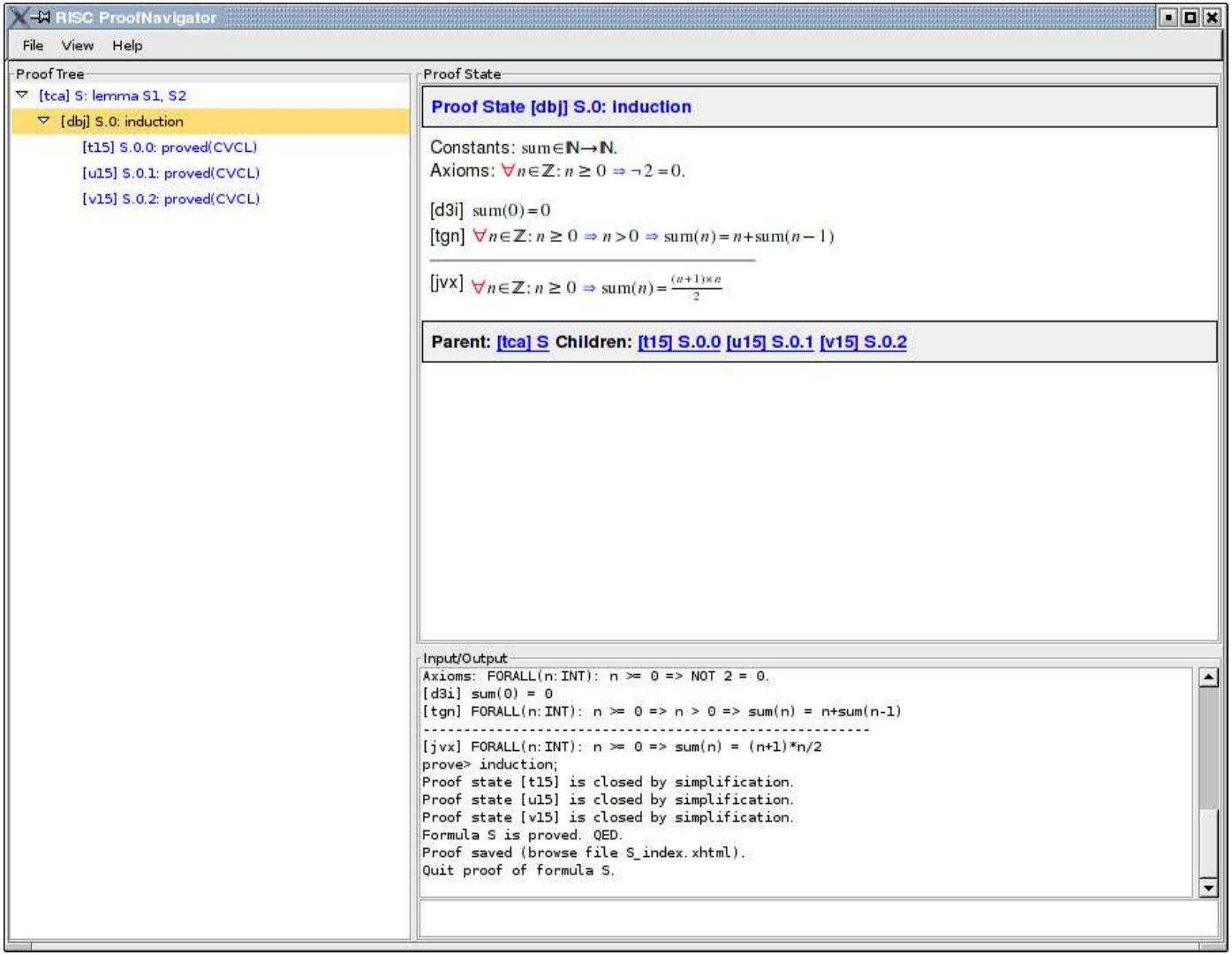
ProofNavigator GUI



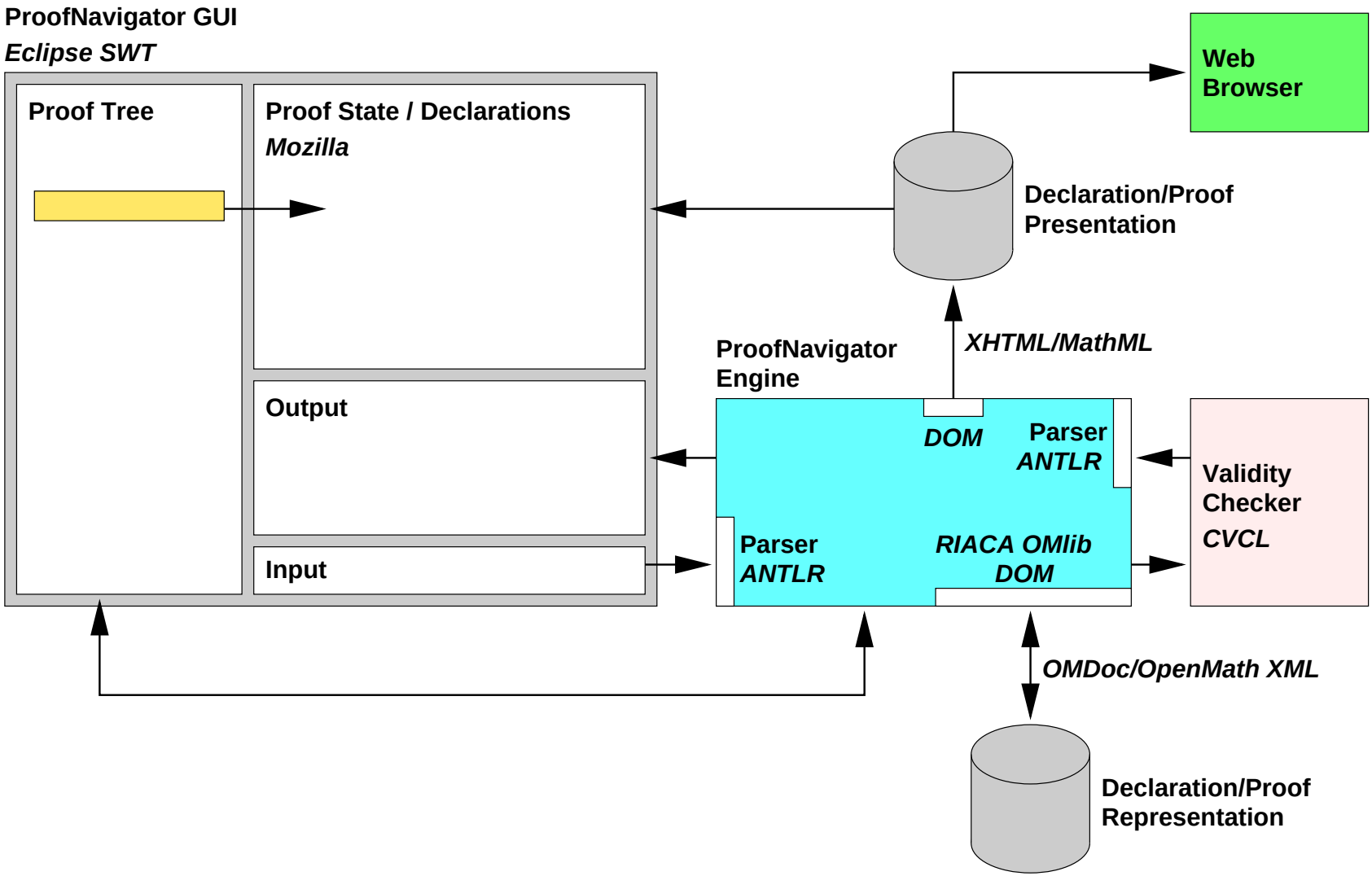
RISC environment: `/zvol/formal/bin/ProofNavigator &`



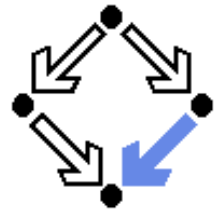
A Sample Proof



ProofNavigator Architecture

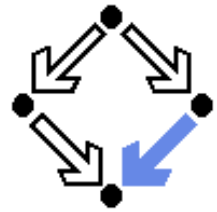


ProofNavigator Components



- Specification language.
  - Generated type checking conditions forwarded to validity checker.
- Proof engine.
  - Commands for navigating the proof.
  - Interaction with validity checker to simplify proof states.
- Validity checker.
  - Simplifying formulas
  - Checking the validity of formulas.
  - Producing counterexamples for (presumedly) invalid formulas.
- Object repository.
  - Proof persistence.
  - Proof status management.
- Presentation interface.
  - Display of declarations and proof state.

The Specification Language



- Definitions of types, constants (functions/predicates), formulas.

```
I: TYPE = INT;
a: I;
f: INT = LAMBDA(x:INT): IF x >= 0 THEN a+x ELSE a-x ENDIF;
F: FORMULA a = 0 => (FORALL(x:INT): f(x) = a);
```

- Builtin types with arithmetic subtype relations.

```
BOOLEAN, [-5, 5], NAT, INT, REAL.
```

- Tuples, Arrays, Records, Functions.

```
(INT, INT), ARRAY(INT,INT), RECORD(a:INT, b:INT), INT->INT.
```

- Uninterpreted types.

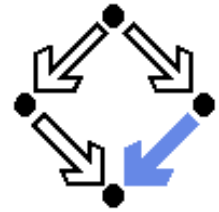
```
T: TYPE;
```

- Subtypes.

```
N: TYPE = SUBTYPE(x:INT): x >= 0;
```

The ProofNavigator borrows the language of PVS (and CVCL).

Subtyping



- Builtin subtyping with automatic type propagation.
  - `[-5,5] <= INT`, `NAT <= INT`, `INT <= REAL`.
  - Without arithmetic subtyping, reasoning becomes very cumbersome!
- User-defined subtypes.

```
NAT:TYPE = SUBTYPE(x:INT): x>=0
```

  - `a:NAT = b-1`

```
a:INT = b-1
a >= 0
```

  - `f:NAT->NAT = LAMBDA(x:NAT):x+1;`

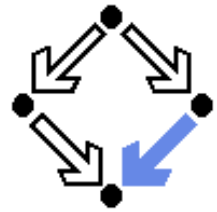
```
f:INT->INT = LAMBDA(x:INT):x+1
FORALL(x:INT): x >= 0 => f(x) >= 0
```

  - `F:FORMULA FORALL(x:NAT): EXISTS(y:NAT): y-x = 0`

```
F:FORMULA FORALL(x:INT): x>=0 =>
    EXISTS(y:INT): y >= 0 AND y-x = 0;
```

Declarations of subtype constants induce type checking conditions.

Type-Checking Conditions



PVS pioneered the idea of combining static type checking with proving.

- Declarations of subtype constants generate TCCs.

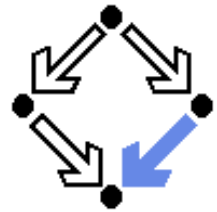
```
a:NAT
   ⇝ a >= 0
f:INT->INT = LAMBDA(x:INT): LET y:NAT = x*x IN y-1
   ⇝ FORALL(x:INT): x*x >= 0
```

- Applications of functions with subtype parameters generate TCCs.

```
f:(SUBTYPE(x:INT):p(x))->INT; b:INT=f(a)
   ⇝ p(a)
g:INT->REAL = LAMBDA(x:INT): 1/(x*x+1)
   ⇝ FORALL(x:INT): x*x+1 /= 0
a:[1,n] -> T; i:INT = ...; e:INT = a[i]
   ⇝ 1 <= i <= n
```

The ProofNavigator handles subtypes by generating corresponding type checking conditions and forwarding them to the validity checker.

Proof States



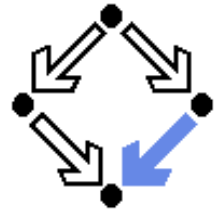
Constants: $x_0 \in S_0, \ldots$
Axioms: $A_0, \ldots$

$$\begin{array}{ll} [l_0] & F_0 \\ \ldots & \\ [l_{n-1}] & F_{n-1} \\ \hline [l_n] & F_n \\ \ldots & \\ [l_{m-1}] & F_{m-1} \end{array}$$

Counterexample: $x_0 = T_0, \ldots$

- Logical interpretation:
  - Prove: $x_0 \in S_0, \ldots, A_0, \ldots \vdash (F_0 \wedge \ldots \wedge F_{n-1}) \Rightarrow (F_n \vee \ldots \vee F_{m-1})$
    - $x_0 \in S_0, \ldots, A_0, \ldots \vdash \neg F_0 \vee \ldots \vee \neg F_{n-1} \vee F_n \vee \ldots \vee F_m$
  - Claim: $(A_0 \wedge \ldots \wedge F_0 \wedge \ldots \wedge F_{n-1} \wedge \neg F_n \wedge \ldots \wedge \neg F_m)[T_0/x_0, \ldots]$
- Label l_i is hashed from formula F_i: $l_i = h(F_i)$.
  - Some hope to make proofs more robust.
- Formula F_i may be also a negation.
  - Command available to flip formula from one side to the other.

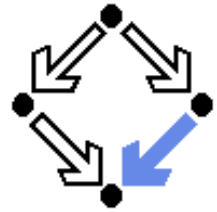
Proof Commands



- Proof decomposition:
  - `flatten`: split all conjunctive assumptions and disjunctive goals.
  - `skolemize`: skolemize all universal goals and existential assumptions.
  - `simplify`: repeatedly apply `flatten` and `skolemize`.
  - `induction`: apply mathematical induction to universal goal over `NAT`.
- Creative steps:
  - `instantiate`: instantiate universal assumption or existential goal.
  - `lemma`: import lemma as an additional assumption.
  - `expand`: expand definition (in some or all formulas).
  - `case`: split proof state by case distinction.
  - `assume`: split proof state by assumption (which is then proved).
- Proof management.
  - `prove`, `quit`: initiate and terminate a proof.
  - `flip`: move formula to the other side of the proof state line.
  - `undo`, `next`, `prev`, `goto`: switch to other proof state.

Leave propositional reasoning and term rewriting to validity checker.

Proof State Simplification



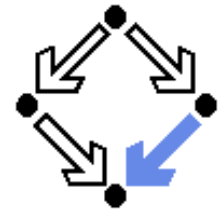
When a proof state $\ldots \vdash F_0 \vee F_1$ is entered,

- Use the identity $F_0 \vee F_1 \equiv (\neg F_1 \Rightarrow F_0) \vee (\neg F_0 \Rightarrow F_1)$
  - to simplify F_0 under the assumption $\neg F_1, \ldots,$
  - to simplify F_1 under the assumption $\neg F_0, \ldots$.

  "Simplification": validity checker computes canonical representation.
    - Re-translation to human-friendly form may be necessary.
- Present the simplified proof state to the validity checker.
  - Answers: "valid", "invalid", "unknown".
    - "Invalid": a counterexample can be requested.
  - If answer is "valid", state is closed.
    - "Invalid": state remains open, because checker may be incomplete.
- Present the simplified proof state to the user.
  - Extended by validity checker's counterexample.

Translation of formulas to the validity checker's theory may be necessary.

Validity Checkers

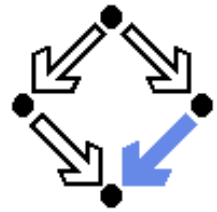


Growing interest in combining decision procedures for various theories relevant for verification.

- Supported theories.
  - Equality and uninterpreted function symbols.
  - Rational linear arithmetic, real linear arithmetic.
  - Functional arrays, products and sums, inductive datatypes, bitvectors.
  - Also: heuristic extensions for quantifier reasoning.
- Combination of decision procedures.
  - Nelson-Oppen method, Shostak method, ...
  - Manna and Zarba, "Combining Decision Procedures", Formal Methods at the Crossroads, Springer LNCS 2787, 2003.
  - Zarba, "The Combination Problem in Automated Reasoning", PhD Thesis, Stanford, 2004.

Basis of various software packages.

Validity Checkers

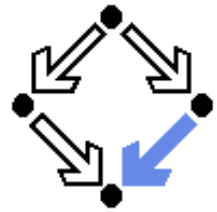


Several prominent examples of validity checkers.

- Simplify (ESC/Java2), Compaq.
- Integrated Canonizer and Solver (IC), SRI.
- Stanford Validity Checker (SVC), Stanford.
- Cooperating Validity Checker (CVC), Stanford.
- CVC Lite (CVCL), Stanford.
- UCLID, CMU.
- Math-SAT, Univ. Trento.

Survey Paper: de Moura and Rueß, "An Experimental Evaluation of Ground Decision Procedures", CAV 2004.

CVC Lite

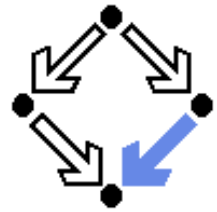


From `http://verify.stanford.edu/CVCL`:

> *CVC Lite is an efficient decision procedure for quantifier-free first-order logic over several interpreted theories (linear arithmetic, uninterpreted functions, arrays, etc.). It is actively developed by the Formal Verification Group at Stanford University and the Analysis of Computer Systems (ACSys) group at New York University.*
>
> *The theoretical basis of CVC Lite is the framework for cooperating decision procedures developed by Clark Barrett. Roughly speaking, the theoretical framework is a happy marriage of Nelson-Oppen and Shostak methods with some enhancements and bug fixes.*
>
> *It is a complete rewrite of the Cooperating Validity Checker (CVC), originally written by Aaron Stump. CVC, in turn, was a rewrite of the Stanford Validity Checker (SVC). That is, we are talking about a third-generation tool here, so it had better be good..*

Current Version 2.0 (2005).

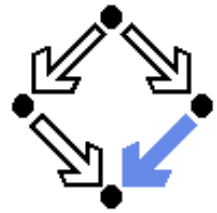
CVC Lite



RISC environment: /zvol/formal/bin/cvcl +int

```
crow!10> cvcl +int
CVC> a:INT; b:INT; c:INT;
CVC> ASSERT a>0;
CVC> PUSH; QUERY b<c => a+b < a+c; POP;
Valid.
CVC> PUSH; QUERY b<=c => a+b < a+c; POP;
InValid.
CVC> COUNTEREXAMPLE;
Current scope level is 3.
%Satisfiable  Variable Assignment: %
ASSERT (a = 1);
ASSERT (b = 0);
ASSERT (c = 0);
CVC> TRANSFORM IF a>0 THEN  a+1+2*(b+1) ELSE 0 ENDIF;
(3 + (2 * b) + a)
```

Object Repository

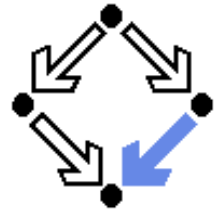


XML-based repository of declarations and proofs.

- OMDoc: Open Mathematical Documents
  - `http://www.mathweb.org/omdoc`
  - Michael Kohlhase, International University of Bremen.
  - Based on the OpenMath (OM) standard.
    - `http://www.openmath.org`
- Every declaration is stored as an OMDoc snippet.
  - Proof context can be combined to complete OMDoc document.
- Proofs are stored as XML trees.
  - ProofNavigator-specific representation.
- Dependencies among declarations and proofs are managed.
  - Proofs are automatically read from repository and tagged.
  - "Trusted" / "Untrusted": have dependencies changed?
    - When replaying a trusted proof, no error can occur.
  - "Open/closed": does proof have open states (or depend on such)?
    - A closed proof represents a formula certificate.

Subdirectory `./ProofNavigator` holds object repository.

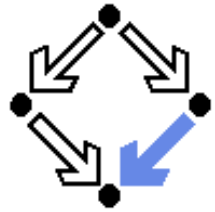
Declaration Representation



```
f:FORMULA a>0 => a-1 >= 0;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<omdoc:omgroup xmlns:omdoc="http://www.mathweb.org/omdoc" ...>
  <omdoc:assertion type="formula" xml:id="f">
    <om:FMP>
      <om:OMA>
        <om:OMS cd="logic1" name="implies"/>
        <om:OMA>
          <om:OMS cd="relation1" name="gt"/>
          <om:OMV name="a"/>
          <om:OMI>0</om:OMI>
        </om:OMA>
        <om:OMA>
          <om:OMS cd="relation1" name="geq"/>
            <om:OMA>
              <om:OMS cd="arith1" name="minus"/>
              <om:OMV name="a"/>
              <om:OMI>1</om:OMI>
            </om:OMA>
          ...
</omdoc:omgroup>
```

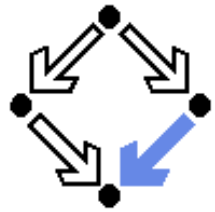
Proof Representation



```
prove g;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<fmrisc:proof xmlns:fmrisc="http://www.risc.uni-linz.ac.at/FM-RISC"
  xmlns:om="http://www.openmath.org/OpenMath" closed="false" name="g">
  <fmrisc:state>
    <fmrisc:command name="case">
      <om:OMA>
        <om:OMS cd="relation1" name="eq"/>
        <om:OMV name="a"/>
        <om:OMI>1</om:OMI>
      </om:OMA>
    </fmrisc:command>
    <fmrisc:state>
      <fmrisc:command name="proved">CVCL</fmrisc:command>
    </fmrisc:state>
    <fmrisc:state>
      <fmrisc:command name="empty"/>
    </fmrisc:state>
  </fmrisc:state>
</fmrisc:proof>
```

Presentation Interface

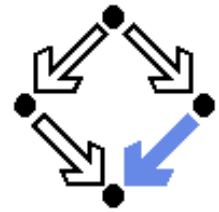


Based on XHTML/MathML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
  "http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  ...
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mo>-</mo>
      <mn>1</mn>
      <mo lspace="1px" rspace="1px">+</mo>
      <mi>a</mi>
      <mo lspace="3px" rspace="3px">=</mo>
      <mn>0</mn>
    </math>
  ...
</html>
```

MathML “presentation” subset understood by most browsers.

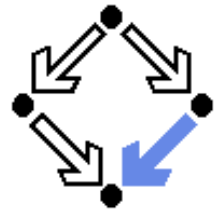
Current Status and Future Work



- Basic framework operational and stable.
  - About 150 classes and 20000 lines of code.
  - But only toy examples have been tested so far.
- Next: better CVCL interaction.
  - Error detection and recovery.
  - Re-translation of canonized formulas to human-friendly form.
  - Decomposing formulas for validity checking of individual parts.
- Goal: run my test cases successfully and analyze the results.
  - Certainly more work will arise from this...
- Finally: documentation and paper.
  - Release of system planned for first half of 2006.
- Perhaps: extension by other automated decision procedures.
  - Interfaces for Simplify and ICS may be interesting.

Hopefully I can show some more examples soon.

Acknowledgements



The ProofNavigator is based on various open source software.

- CVC Lite 2.0
- RIACA OpenMath Library 2.0
- General Purpose Hash Function Algorithms Library
- ANTLR 2.7.6b2
- Eclipse Standard Widget Toolkit 3.1.1
- The Mozilla Browser 1.7.8 GTK2
- The GIMP Toolkit GTK+ 2.X
- Java Development Kit 5.0

Contributing to the open source pool is important.