UNIVERSITÄT KARLSRUHE (TH)
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Mattias Ulbrich

A Short Introduction to First-Order Theorem Proving with KeY

1 What is KeY?

1.1 Software Verification

The KeY system allows to verify software written in Java Card against its formal specification. The concept of *design by contract* is captured in a *dynamic logic for Java Card (Java Card DL)*. This logic is a sorted, multi-modal logic whose modalities are induced by code in Java Card.

A typical proof obligation for a method contract is

$$\alpha \rightarrow \langle \texttt{o.m();} \rangle \beta$$

stating that if a pre-condition α holds before a method call `o.m()`, and a post-condition β holds after the call.

The KeY system uses a *sequent calculus* to prove formulas. Proofs can be performed both automatically and interactively. The user can apply rules of the calculus at their own discretion at any time, but due to the repetitive nature of proof tasks, strategies exist that allow to conduct many proofs automatically (or at least to a great deal automatically).

1.2 KeY for First-Order Proving

If one forgets about modal operators, Java Card DL is a sorted first order predicate logic and, thus, proofs can also performed in that logic within the KeY system.

Due to the fact that the KeY system is optimised for software verification purposes and not for general first-order proofs, proofs that are likely to be conducted automatically in other proof system, may not be closed in KeY.

However, KeY carriers some advantages over such (highly efficient) totally automatic proof systems.

- The sequent calculus allows the user to understand any intermediate state of the proof – the formulas are merely decomposed than drastically changed (like happens for instance for any resolution calculus)
- Interaction and automatisation can be combined as the use wishes which allows to perform the few crucial steps manually and the remainder automatically.
- If one (accidently or deliberately) tries to prove a formula which is not universally valid, one can propably read a counter example from an appropriately performed partial proof.

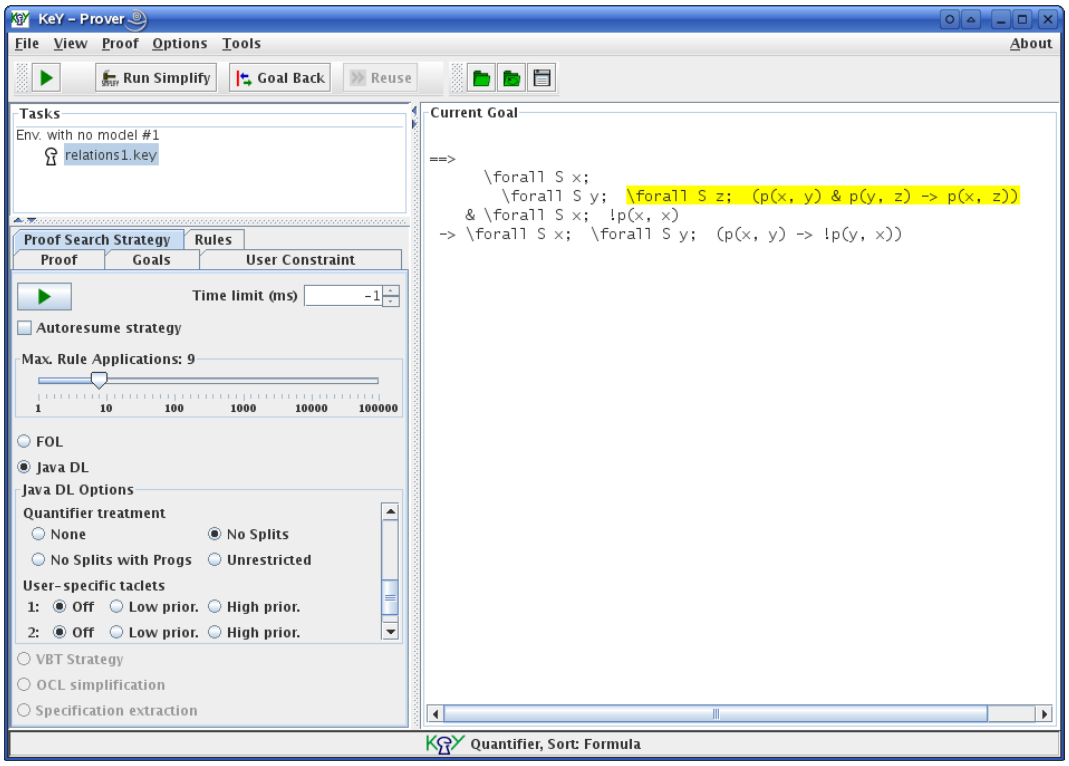
2 Using the KeY System

2.1 Startup

The KeY prover is entirely written in Java and will therefore run on any system on which a Java Virtual Machine is available. The software can be downloaded from the project's homepage `http://www.key-project.org`.

If you follow the "Java WebStart" links on the page and have Java appropriately installed, the KeY system will immediately start up without the need to install it on your computer at all. You can, however, also download it as a tarball and install it locally on your computer. Please follow the installation instructions given on the KeY web site.

2.2 Appearance



```
==>
     \forall S x;
       \forall S y;  \forall S z;  (p(x, y) & p(y, z) -> p(x, z))
   & \forall S x;  !p(x, x)
 -> \forall S x;  \forall S y;  (p(x, y) -> !p(y, x))
```

Time limit (ms) -1

Autoresume strategy [ ]

Max. Rule Applications: 9

FOL [✓ Java DL]

Quantifier treatment: None [✓ No Splits] No Splits with Progs Unrestricted

1: [✓ Off] Low prior. High prior.

2: [✓ Off] Low prior. High prior.

Quantifier, Sort: Formula

Figure 1: Screenshot of KeY

KeY is a tool that is still very much evolving, so some details may differ in the version that you are using. But the general ideas should still apply.

When you have started KeY, a window similar to the one in Figure 1 will pop up (but somewhat more empty). This the KeY main window and can be subdivided into three areas:

Upper left hand side: You can load an start several proofs at a time. You can choose the one to work upon in this area

Right hand side: This is the pane in which the current sequent is presented. You can interactively apply calculus rules in this area.

Lower left hand side: There are several tabs of which the more interesting ones are briefly described here:

Proof Search Strategy allows to set up the parameters of the automatic proof search. As we are only interested in first-order logic, there are only few switches that matter:

- The slider allows you to set how many steps the automatic strategy will perform if used.
- FOL (short for First-Order Logic) – is suitable for most of the problems that one encounters in FO.
- Java DL – is primarily for software verification but can also be helpful for first order.

- If choosing Java DL, decide how splitting rules (such as and-right) are to be treated.
- Quantifier treatment – this can be used to control the automatic instantiation of quantified formulas ("No Splits" is most often the best choice)

Proof provides the complete proof tree of the sequent calculus. Closed braches are marked with red leafs and open branches carry green leafs.

Goals contains a list of all leafs (i.e. goals) that have not yet been closed.

2.3 Problem Files

Let us assume that we want to prove the following theorem:

Any binary relation which is both transitive and irreflexive is also antisymmetric.

Formalising this in predicate logic using a binary predicate p yields

$$\begin{array}{ll} & \forall x.\forall y.\forall z.(p(x,y) \land p(y,z) \to p(x,z)) \\ \land & \forall x.\neg p(x,x) \\ \to & \forall x.\forall y.(p(x,y) \to \neg p(y,x)) \end{array}$$

But how can we feed this problem statement into KeY? We have to create a problem file which encodes this proof task.

This file is an ASCII-file so that we have to replace the special characters of the logic with ASCII counterparts. The table in Fig. 2 lists all such replacements. (Hint: The characters ressemble the ones in Java.)

FOL	**KeY**
$\forall x.\phi$	`\forall S x;` ϕ'
$\exists x.\phi$	`\exists S x;` ϕ'
$\phi \land \psi$	ϕ' `&` ψ'
$\phi \lor \psi$	ϕ' `|` ψ'
$\phi \to \psi$	ϕ' `->` ψ'
$\neg\phi$	`!` ϕ'

Figure 2: Character replacement in problem files

In addition to the problem itself, the used signature (predicate and function symbols) must be declared. Since KeY uses a sorted (typed) logic, sorts have to be declared too. If you do not want to use the feature of multiple sorts, just declare a single sort and always use this.

The example theorem can therefore be captured in a problem file like the source printed in Figure 3. We save such problem statements in a file whose file name extension is `.key`. For our purposes such a file consists of four compartments:

1. The sort declarations. List all sorts to be used here, separated by semicola.
2. The predicate symbol declarations. List all predicates here, separated by semicola. Since we are in a sorted enviroment, the arguments are sorted as well.
3. The function symbol declarations[1]. List all functions here, separated by semicola. Since we are in a sorted enviroment, the arguments and the return value are typed as well.
4. The problem statement. State the problem in Java DL which you wish to prove universally valid.

[1]For the example in this introduction, the function symbol is not needed but included for illustration purposes

```
/* Example No 1 */
/* (Comments can be written using the C++/Java syntax) */

\sorts {
   S;
}

\predicates {
   p(S,S);
}

\functions {
   S f(S); // This function is not needed but included for demonstration purposes
}

\problem {
 (
   \forall S x; (\forall S y; (\forall S z; ((p(x,y) & p(y,z)) -> p(x,z))))
  & \forall S x; ! p(x,x)
 )
 -> \forall S x; (\forall S y; (p(x,y) -> ! p(y,x)))
}
```

Figure 3: Problem file for the example

2.4 Applying Rules

This problem can now be saved as a file and loaded into KeY (via the menu File.Load for instance). Your window should now look rather similar to the one in Fig. 1.

Before we let the KeY machine run automatically, we apply some rules manually. Moving the mouse cursor across the sequent on the right hand pane, you can select the entity to apply a rule onto (either the entire squent, a formula on either side, a subformula or a term within a formula).

Move your mouse so that the entire formula but not the sequent is coloured and press the left button. A menu will pop up from which you can choose the rule to apply on the marked entity. Choose impRight to distribute the implication around the sequent's arrow. Now select the formula in the antedecent (left side of the `==>`) and apply the rule andLeft. Then select the formula
`\forall S x; \forall S y; (p(x, y) -> !p(y, x))`
on the succedent side (right side of the `==>`) and apply the rule allRight. One univeral quantifier vanishes and a new symbol `x_1` replaces the variable `x`: Skolemisation has happened.

Some rules require more user interaction, such as some quantifier instantiations (γ-rules). Please select `\forall S x; !p(x, x)` and apply the rule leftAll on it. A new window will arise and allow you to input the instantiation of the variable. Enter `x_1` into the input field for the term t and press "Apply". A correspondingly instantiated formula appears on antecedental side of the sequent.

Instead of opening the instantiation window, quantor instantiations (and other rules) can be made using "drag'n'drop" with the mouse. Drag a term onto a quantified formula to instantiate it.

With the general idea of interactive rule application sorted out, we can now procede to automatic proving. Make sure you have "FOL" selected in the Proof Search Strategy tab and have set the slider to a value round 100. Now press the green 'go' button in the toolbar and after a short time the proof is completed.

Study the proof in the proof in the Proof pane. Perhaps you want reload the file and try to prove it in less steps than the automatic strategy.

2.5 Saving Results

At any time during a proof, the current proof tree can be saved to a file. Thus, you can continue a proof at a later time. Use the menu File.Save to save the current proof. Usually, the file name extension for saved KeY files is ".key.proof"

3 Further Reading

1. Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt *Verification of Object-Oriented Software. The KeY Approach*, Springer-Verlag, Lecture Notes in Artificial Intelligence Vol. 4334, 2006.

   In particular Chapter 2 (First-Order Logic) and Chapter 10 (Using KeY) are relevant for first-order proofs in KeY.

2. David Harel. Dynamic Logic. In *Handbook of Philosophical Logic*, D. Reidel Publishing Company, Extensions of Classical Logic vol. 2, p. 497–604, 1984.

3. The KeY project homepage `http://www.key-project.org`

   It contains links to various papers, documentation, tutorials around the KeY system.