CADE Tutorial
The Sequent Calculus of the KeY Tool
Part II

Reiner Hähnle & Peter H. Schmitt

Department of Computer Science
Technische Universität Darmstadt / Karlsruhe Institute of Technology

3 August 2015

Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up

Typed First-Order Logic

A type hierarchy $\mathcal{H} = (\mathcal{T}, \sqsubseteq)$ consists of

- a set of types $\mathcal{T}$ and a subtype relation $\sqsubseteq$ on $\mathcal{T} \times \mathcal{T}$.

Typed First-Order Logic

A type hierarchy $\mathcal{H} = (\mathcal{T}, \sqsubseteq)$ consists of

- a set of types $\mathcal{T}$ and a subtype relation $\sqsubseteq$ on $\mathcal{T} \times \mathcal{T}$.

For Vocabulary $\Sigma = (\mathit{Func}, \mathit{Pred}, \mathit{Var})$

- Type spec $f : T_1 \times \ldots T_n \to T_0$ required for $f \in \mathit{Func}$,
- Type spec $p : T_1 \times \ldots T_n$ required for $p \in \mathit{Pred}$,
- Type spec $x : T$ required for $x \in \mathit{Var}$,

Typed First-Order Logic

A type hierarchy $\mathcal{H} = (\mathcal{T}, \sqsubseteq)$ consists of

- a set of types $\mathcal{T}$ and a subtype relation $\sqsubseteq$ on $\mathcal{T} \times \mathcal{T}$.

For Vocabulary $\Sigma = (\mathit{Func}, \mathit{Pred}, \mathit{Var})$

- Type spec $f : T_1 \times \ldots T_n \to T_0$ required for $f \in \mathit{Func}$,
- Type spec $p : T_1 \times \ldots T_n$ required for $p \in \mathit{Pred}$,
- Type spec $x : T$ required for $x \in \mathit{Var}$,

Recursive Definition of Terms $(\sigma(t)$ is the type of term $t)$

Typed First-Order Logic

A type hierarchy $\mathcal{H} = (\mathcal{T}, \sqsubseteq)$ consists of

- a set of types $\mathcal{T}$ and a subtype relation $\sqsubseteq$ on $\mathcal{T} \times \mathcal{T}$.

For Vocabulary $\Sigma = (\mathit{Func}, \mathit{Pred}, \mathit{Var})$

- Type spec $f : T_1 \times \ldots T_n \to T_0$ required for $f \in \mathit{Func}$,
- Type spec $p : T_1 \times \ldots T_n$ required for $p \in \mathit{Pred}$,
- Type spec $x : T$ required for $x \in \mathit{Var}$,

Recursive Definition of Terms $(\sigma(t)$ is the type of term $t)$

For $f \in \mathit{Func}$ with $f : T_1 \times \ldots T_n \to T_0$ and terms $\sigma(t_i) \sqsubset T_i$

$$f(t_1, \ldots, t_n)$$

is a term of type T_0.

Typed First-Order Logic

A type hierarchy $\mathcal{H} = (\mathcal{T}, \sqsubseteq)$ consists of

- a set of types $\mathcal{T}$ and a subtype relation $\sqsubseteq$ on $\mathcal{T} \times \mathcal{T}$.

For Vocabulary $\Sigma = (\mathit{Func}, \mathit{Pred}, \mathit{Var})$

- Type spec $f : T_1 \times \ldots T_n \to T_0$ required for $f \in \mathit{Func}$,
- Type spec $p : T_1 \times \ldots T_n$ required for $p \in \mathit{Pred}$,
- Type spec $x : T$ required for $x \in \mathit{Var}$,

Recursive Definition of Terms ($\sigma(t)$ is the type of term t)

For $f \in \mathit{Func}$ with $f : T_1 \times \ldots T_n \to T_0$ and terms $\sigma(t_i) \sqsubset T_i$

$$f(t_1, \ldots, t_n)$$

is a term of type T_0.

Definition of Formulas Unchanged

How to Deal with Typed Logic

Transform into untyped predicate logic

- Logic textbooks by Donald Monk (1997), Maria Manzano (1996).

How to Deal with Typed Logic

Transform into untyped predicate logic

- Logic textbooks by Donald Monk (1997), Maria Manzano (1996).
- A Polymorphic Intermediate Verification Language:
  Design and Logical Encoding
  K. Rustan M. Leino and Philipp Rümmer
  TACAS 2010, SLNCS 4015, pp 321-327

How to Deal with Typed Logic

Transform into untyped predicate logic

- Logic textbooks by Donald Monk (1997), Maria Manzano (1996).
- A Polymorphic Intermediate Verification Language:
  Design and Logical Encoding
  K. Rustan M. Leino and Philipp Rümmer
  TACAS 2010, SLNCS 4015, pp 321-327

How to Deal with Typed Logic

Transform into untyped predicate logic

- Logic textbooks by Donald Monk (1997), Maria Manzano (1996).
- A Polymorphic Intermediate Verification Language:
  Design and Logical Encoding
  K. Rustan M. Leino and Philipp Rümmer
  TACAS 2010, SLNCS 4015, pp 321-327

Extend Calculus

- A Calculus for Type Predicates and Type Coercion
  Martin Giese
  TABLEAUX 2005, SLNCS Vol.3702, pp 123-137
- A Calculus for Typed First-Order Logic
  Peter H. Schmitt and Mattias Ulbrich
  FM 2015

First-Order Rules

$$\text{allRight}\ \frac{\Gamma \Longrightarrow [x/c](\varphi), \Delta}{\Gamma \Longrightarrow (\forall\, A\, x)\varphi, \Delta}$$
$c : \to A$ a new constant

$$\text{allLeft}\ \frac{\Gamma, \forall\, A\, x)\varphi, [x/t](\varphi) \Longrightarrow \Delta}{\Gamma, (\forall\, A\, x)\varphi \Longrightarrow \Delta}$$
t ground term, $\sigma(t) \sqsubseteq A$

$$\text{exLeft}\ \frac{\Gamma, [x/c](\varphi) \Longrightarrow \Delta}{\Gamma, (\exists\, A\, x).\varphi \Longrightarrow \Delta}$$
$c : \to A$ a new constant

$$\text{exRight}\ \frac{\Gamma \Longrightarrow (\exists\, A\, x)\varphi, [x/t](\varphi), \Delta}{\Gamma \Longrightarrow (\exists\, A\, x)\varphi, \Delta}$$
t a ground term, $\sigma(t) \sqsubseteq A$

$$\text{close}\ \frac{}{\Gamma, \varphi \Longrightarrow \varphi, \Delta}$$

$$\text{closeFalse}\ \frac{}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$$

$$\text{closeTrue}\ \frac{}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$$

Equational Rules

$$\text{eqLeft}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$
$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

$$\text{eqRight}\ \frac{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_2](\varphi), [z/t_1](\varphi), \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_1](\varphi), \Delta}$$
$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

$$\text{eqSymmLeft}\ \frac{\Gamma, t_2 \doteq t_1 \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta} \qquad \text{eqReflLeft}\ \frac{\Gamma, t \doteq t \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

A Closer Look at eqLeft

$$\text{eqLeft}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$

$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

A Closer Look at eqLeft

$$\text{eqLeft} \ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$
$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

Why is the side condition $\sigma(t_2) \sqsubseteq \sigma(t_1)$ necessary?

A Closer Look at eqLeft

$$\text{eqLeft}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$

$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

Why is the side condition $\sigma(t_2) \sqsubseteq \sigma(t_1)$ necessary?

Consider the signature:

$B \sqsubsetneq A$, $a : \ \to A$, $b : \ \to B$, $p : B$.

A Closer Look at eqLeft

$$\text{eqLeft}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$
$$\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)$$

Why is the side condition $\sigma(t_2) \sqsubseteq \sigma(t_1)$ necessary?
Consider the signature:
$B \sqsubsetneqq A$, $a : \ \to A$, $b : \ \to B$, $p : B$.
Applying eqLeft without side condition on the sequent

$$b \doteq a, p(b) \Longrightarrow$$

would result in

$$b \doteq a, p(b), p(a) \Longrightarrow$$

with $p(a)$ being not well-typed.

Soundness and Completeness

Martin Giese's TABLEAUX paper

presents a sound and complete calculus
provided the type hierarchy $\mathcal{H}$ is closed under greatest lower bounds.

Soundness and Completeness

Martin Giese's TABLEAUX paper

presents a sound and complete calculus
provided the type hierarchy $\mathcal{H}$ is closed under greatest lower bounds.

Optimization

A sound and complete calculus without restrictions on $\mathcal{H}$
can be obtained by the following modification:

$$\text{eqLeft}\ \frac{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Longrightarrow \Delta}$$
$$\text{provided } [z/t_2](\varphi) \text{ is welltyped}$$

Likewise for eqRight.

Modular Reasoning

Theoretical Motivation

$\neg(\exists x)(\exists y)(x \doteq y)$
is tautology
for this type hierarchy

$$
\begin{tikzcd}
 & \top & \\
A \arrow[ur] & & B \arrow[ul] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

$A : x$
$B : y$

Modular Reasoning

Theoretical Motivation

$\neg(\exists x)(\exists y)(x \doteq y)$
is tautology
for this type hierarchy

$$\begin{tikzcd}
& \top & \\
A \arrow[ur] & & B \arrow[ul] \\
& \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

$A : x$
$B : y$

$\neg(\exists x)(\exists y)(x \doteq y)$
is not a tautology
for the extended
type hierarchy

$$\begin{tikzcd}
& \top & \\
A \arrow[ur] & & B \arrow[ul] \\
& C \arrow[ul] \arrow[ur] & \\
& \bot \arrow[u] &
\end{tikzcd}$$

Modular Reasoning

Theoretical Motivation

$\neg(\exists x)(\exists y)(x \doteq y)$
is tautology
for this type hierarchy

$$\begin{tikzcd}
 & \top & \\
A \arrow[ur] & & B \arrow[ul] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

$A : x$
$B : y$

$\neg(\exists x)(\exists y)(x \doteq y)$
is not a tautology
for the extended
type hierarchy

$$\begin{tikzcd}
 & \top & \\
A \arrow[ur] & & B \arrow[ul] \\
 & C \arrow[ul] \arrow[ur] & \\
 & \bot \arrow[u] &
\end{tikzcd}$$

The phenomenon that universal validity of a formula depends on symbols not occurring in it, is highly undesirable.

Modular Reasoning

Practical Motivation from Program Verification

Segment of Java Library

$$
\begin{tikzcd}
 & \text{java.lang.Object} & \\
 & \text{java.util.AbstractCollection} \arrow[u] & \\
 & \text{java.util.AbstractList} \arrow[u] & \\
\text{java.util.ArrayList} \arrow[ur] & \text{AbstractSequentialList} \arrow[u] & \text{Vector} \arrow[ul]
\end{tikzcd}
$$

Modular Reasoning

Practical Motivation from Program Verification

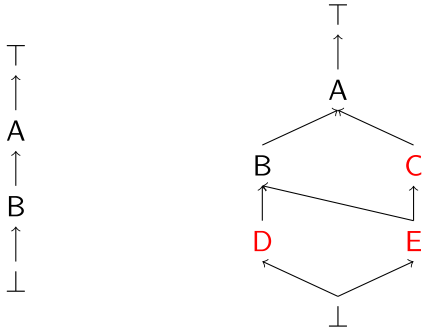
Segment of Java Library

java.lang.Object
↑
java.util.AbstractCollection
↑
java.util.AbstractList
↗ ↑ ↖
java.util.ArrayList AbstractSequentialList Vector

Verification of programs using these classes should remain valid if another subclass is added to java.util.AbstractList.

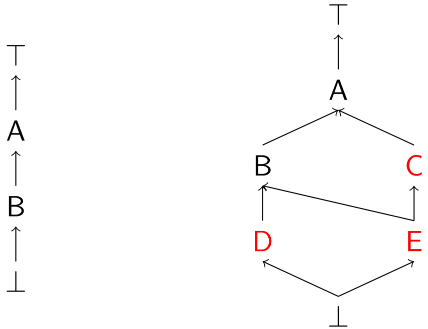
Type Hierarchies Extensions

$$(\mathrm{TSym}_1, \sqsubseteq_1) \quad \sqsubseteq \quad (\mathrm{TSym}_2, \sqsubseteq_2)$$



Type Hierarchies Extensions

$$(\mathrm{TSym}_1, \sqsubseteq_1) \quad \sqsubseteq \quad (\mathrm{TSym}_2, \sqsubseteq_2)$$



Only subtype relations may be added.

Logical Consequence Relation

Definition of Super-Consequence Relation

$\varphi \in \mathrm{Fml}_{\mathcal{T},\Sigma}$, $\Phi \subseteq \mathrm{Fml}_{\mathcal{T},\Sigma}$
for type hierarchy $\mathcal{T}$ and signature Σ.

$\Phi \vdash \varphi$ iff for all type hierarchies $\mathcal{T}'$ with $\mathcal{T} \sqsubseteq \mathcal{T}'$
and all $\mathcal{T}' - \Sigma$-structures $\mathcal{M}$
if $\mathcal{M} \models \Phi$ then $\mathcal{M} \models \varphi$

Logical Consequence Relation

Definition of Super-Consequence Relation

$\varphi \in \mathrm{Fml}_{\mathcal{T},\Sigma}$, $\Phi \subseteq \mathrm{Fml}_{\mathcal{T},\Sigma}$
for type hierarchy $\mathcal{T}$ and signature Σ.

$\Phi \vdash \varphi$ iff for all type hierarchies $\mathcal{T}'$ with $\mathcal{T} \sqsubseteq \mathcal{T}'$
and all $\mathcal{T}' - \Sigma$-structures $\mathcal{M}$
if $\mathcal{M} \models \Phi$ then $\mathcal{M} \models \varphi$

Logical Consequence Relation

Definition of Super-Consequence Relation

$\varphi \in \mathrm{Fml}_{\mathcal{T},\Sigma}$, $\Phi \subseteq \mathrm{Fml}_{\mathcal{T},\Sigma}$
for type hierarchy $\mathcal{T}$ and signature Σ.

$$\Phi \vdash \varphi \quad \text{iff} \quad \begin{array}{l} \text{for all type hierarchies } \mathcal{T}' \text{ with } \mathcal{T} \sqsubseteq \mathcal{T}' \\ \text{and all } \mathcal{T}' - \Sigma\text{-structures } \mathcal{M} \\ \text{if } \mathcal{M} \models \Phi \text{ then } \mathcal{M} \models \varphi \end{array}$$

For the example on the previous slide.

$$\emptyset \not\vdash \neg(\exists x)(\exists y)(x \doteq y)$$

$$\begin{tikzcd}
 & \top & \\
A \arrow[ur] & & B \arrow[ul] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

$A : x$
$B : y$

Note

Completeness

A calculus that is complete for the ordinary consequence relation is also complete for the super-consequence relation.

Note

Completeness

A calculus that is complete for the ordinary consequence relation is also complete for the super-consequence relation.

Note

Completeness

A calculus that is complete for the ordinary consequence relation
is also complete for the super-consequence relation.

Soundness

A rule that is sound for the ordinary consequence relation
need not be sound for the super-consequence relation.

Conditional Terms

Syntax

(if φ then t_1 else t_2) is a term of type A
for a formula φ and terms t_i of type A_i
if $A_2 \sqsubseteq A_1 = A$ or $A_1 \sqsubseteq A_2 = A$.

Conditional Terms

Syntax

$(\text{if } \varphi \text{ then } t_1 \text{ else } t_2)$ is a term of type A
for a formula φ and terms t_i of type A_i
if $A_2 \sqsubseteq A_1 = A$ or $A_1 \sqsubseteq A_2 = A$.

Semantics by Elimination (Example)

$(\forall x, y)(x \leqslant (\text{if } x \leqslant y \text{ then } y \text{ else } x))$
is replaced by
$(\forall x, y)(x \leqslant y \to x \leqslant y \land \neg(x \leqslant y) \to y \leqslant y)$

Motivation

At the VSTTE'10 (Verified Software:Theories, Tools and Experiments) conference 2010 in Edinburgh.

VSComp: The Verified Software Competition

Problem 1

- Description: Given an N-element array of natural numbers, write a program to compute the sum and the maximum of the elements in the array.
- Properties: Given that $N \geqslant 0$ and $a[i] \geqslant 0$ for $0 \leqslant i < N$, prove the post-condition that $sum \leqslant N * max$

Motivation

At the VSTTE'10 (Verified Software:Theories, Tools and Experiments) conference 2010 in Edinburgh.

VSComp: The Verified Software Competition

Problem 1

- Description: Given an N-element array of natural numbers, write a program to compute the sum and the maximum of the elements in the array.
- Properties: Given that $N \geqslant 0$ and $a[i] \geqslant 0$ for $0 \leqslant i < N$, prove the post-condition that $sum \leqslant N * max$

We need a way to express $\Sigma_{i=b}^{i=e}$.

Variable Binders

Explanation

Variable Binders are function symbols which bind a variable ranging over a set of values.

Application of a variable binder results in a term.

Variable Binders available in KeY

Mathematical Notation	KeY Syntax
$\sum_{b_0 \leqslant vi < b_1} s_1$	$bsum\{Int\ vi;\}(b_0, b_1, s_1)$
$\prod_{b_0 \leqslant vi < b_1} s_1$	$prod\{Int\ vi;\}(b_0, b_1, s_1)$
$\bigcup_{-\infty < vi < \infty} s_2$	$infiniteUnion\{Int\ vi;\}(s_2)$
$\langle s_3[b_0/vi], \ldots, s_3[(b_1 - 1)/vi]\rangle$	$seqDef\{Int\ vi;\}(b_0, b_1, s_3)$

b_0, b_1, s_1 terms of type *Int*,
s_2 term of type *LocSet*,
s_3 term of type *Seq*,
vi would typically occur free in s_1, s_2, s_3

Status of Variable Binders

Theorem

For every formula containing variable binders there is a satisfiability equivalent formula without.

Status of Variable Binders

Theorem

For every formula containing variable binders there is a satisfiability equivalent formula without.

Example

$(\forall\, Int\ N)(bsum\{Int\ vi;\}(0, N, a[vi]) \leqslant N * \ max)$

is replaced by

$$
\begin{array}{l}
f(0) = 0 \quad \wedge \\
(\forall\, Int\, j)(f(j+1) = f(j) + a[j]) \quad \wedge \\
(\forall\, Int\ N)(f(N) \leqslant N * \ max)
\end{array}
$$

Ways to deal with undefinedness

1. Remove it.
   e.g. set the empty sum to 0: $\Sigma_{i=0}^{i=-2} s_i = 0$.
2. Introduce new error elements.
3. Use some kind of 3-valued logic.
4. Use underspecification.

Underspecification

1. All functions are total.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.
4. Downside: user has to understand some non-intuitive behaviour.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.
4. Downside: user has to understand some non-intuitive behaviour.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.
4. Downside: user has to understand some non-intuitive behaviour.
   $(\exists\, i)(\frac{1}{0} \doteq i)$ is a tautology.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.
4. Downside: user has to understand some non-intuitive behaviour.
   $(\exists\, i)(\frac{1}{0} \doteq i)$ is a tautology.

   $\frac{1}{0} \doteq \frac{2}{0}$ is not.

Underspecification

1. All functions are total.
2. Function values at argument positions that are intended to be undefined are chosen arbitrarily within the correct type.
3. Upside: No changes to the logic needed.
4. Downside: user has to understand some non-intuitive behaviour.
   $(\exists\, i)(\frac{1}{0} \doteq i)$ is a tautology.

   $\frac{1}{0} \doteq \frac{2}{0}$ is not.

   Also $cast_{Int}(c) \doteq 5 \rightarrow c \doteq 5$ is not a tautology. In case c is not of type *Int* the underspecified value for $cast_{Int}(c)$ could be 5.

Axioms for $\mathbb{Z}$

$A.1$	$(i + j) + k \doteq i + (j + k)$
$A.2$	$i + j \doteq j + i$
$A.3$	$0 + i \doteq i$
$A.4$	$i + (-i) \doteq 0$
$M.1$	$(i * j) * k \doteq i * (j * k)$
$M.2$	$i * j \doteq j * i$
$M.3$	$1 * x \doteq x$
$M.4$	$i * (j + k) \doteq i * j + i * k$
$M.5$	$1 \neq 0$
$O.1$	$0 < i \vee 0 = i \vee 0 < (-i)$
$O.2$	$0 < i \wedge 0 < j \rightarrow 0 < i + j$
$O.3$	$0 < i \wedge 0 < j \rightarrow 0 < i * j$
$O.4$	$i < j \leftrightarrow 0 < j + (-i)$
$O.5$	$\neg(0 < 0)$
Ind	$\varphi(0) \wedge (\forall i)(0 \leqslant i \wedge \varphi(i) \rightarrow \varphi(i + 1)) \rightarrow (\forall i)(0 \leqslant i \rightarrow \varphi(i))$

Transitive Closure

Definition

Let (G, R) be a graph and G a type.

$$reachR(g, h, 0) \leftrightarrow g \doteq h)$$
$$n \geqslant 0 \rightarrow (reachR(g, h, n + 1) \leftrightarrow (\exists\, G\, k)(reachR(g, k, n) \wedge R(k, h)))$$
$$TR(g, h) \leftrightarrow (\exists\, Int\, n)(n \geqslant 0 \wedge reachR(g, h, n))$$

Here, g, h are variables of type G and n of type Int.

Transitive Closure

Definition

Let (G, R) be a graph and G a type.

$$reachR(g, h, 0) \leftrightarrow g \doteq h)$$
$$n \geqslant 0 \rightarrow (reachR(g, h, n+1) \leftrightarrow (\exists\, G\, k)(reachR(g, k, n) \wedge R(k, h)))$$
$$TR(g, h) \leftrightarrow (\exists\, Int\, n)(n \geqslant 0 \wedge reachR(g, h, n))$$

Here, g, h are variables of type G and n of type Int.

Transitive Closure

Definition

Let (G, R) be a graph and G a type.

$$reachR(g, h, 0) \leftrightarrow g \doteq h)$$
$$n \geqslant 0 \rightarrow (reachR(g, h, n+1) \leftrightarrow (\exists\, G\, k)(reachR(g, k, n) \wedge R(k, h)))$$
$$TR(g, h) \leftrightarrow (\exists\, Int\, n)(n \geqslant 0 \wedge reachR(g, h, n))$$

Here, g, h are variables of type G and n of type Int.

TR is the transitive closure of R.

Transitive Closure

Definition

Let (G, R) be a graph and G a type.

$$reachR(g, h, 0) \leftrightarrow g \doteq h)$$
$$n \geqslant 0 \rightarrow (reachR(g, h, n+1) \leftrightarrow (\exists\, G\, k)(reachR(g, k, n) \wedge R(k, h)))$$
$$TR(g, h) \leftrightarrow (\exists\, Int\, n)(n \geqslant 0 \wedge reachR(g, h, n))$$

Here, g, h are variables of type G and n of type Int.

TR is the transitive closure of R.
More precisely:
In any interpretation satisfying the above three axioms the interpretation of TR is the transitive closure of R.

Proof Exercise

A Special Case of the Bellman-Ford Lemma

Let (G, R) be a graph, $s \in G$ the start element.
Let $d : G \to \mathbb{N}$ be a function satisfying:

$$
\begin{array}{l}
d(s) = 0 \\
(\forall\, G\, g)(g \neq s \to (\exists\, G\, h)(R(h, g) \land d(g) = d(h) + 1)) \\
(\forall\, G\, g)(\forall G\ \ h; (R(h, g) \to d(g) <= d(h) + 1))
\end{array}
$$

Then $d(g)$ is the length of the shortest path from s to g.

Proof Exercise

A Special Case of the Bellman-Ford Lemma

Let (G, R) be a graph, $s \in G$ the start element.
Let $d : G \to \mathbb{N}$ be a function satisfying:

$$
\begin{aligned}
&d(s) = 0\\
&(\forall\, G\, g)(g \neq s \to (\exists\, G\, h)(R(h, g) \wedge d(g) = d(h) + 1))\\
&(\forall\, G\, g)(\forall G\;\, h; (R(h, g) \to d(g) <= d(h) + 1))
\end{aligned}
$$

Then $d(g)$ is the length of the shortest path from s to g.

KeY Demo

A Note on Completeness

Undecidability of $\mathbb{Z}$

The set of all first-order formulas true in $\mathbb{Z}$ is not recursively enumerable.

A Note on Completeness

Undecidability of $\mathbb{Z}$

The set of all first-order formulas true in $\mathbb{Z}$ is not recursively enumerable.

The theory of $\mathbb{Z}$ cannot be axiomatized.

A Note on Completeness

Undecidability of $\mathbb{Z}$

The set of all first-order formulas true in $\mathbb{Z}$ is not recursively enumerable.

The theory of $\mathbb{Z}$ cannot be axiomatized.

The axioms on $\mathbb{Z}$ shown previously fall short of being complete.

A Note on Completeness

Undecidability of $\mathbb{Z}$

The set of all first-order formulas true in $\mathbb{Z}$ is not recursively enumerable.

The theory of $\mathbb{Z}$ cannot be axiomatized.

The axioms on $\mathbb{Z}$ shown previously fall short of being complete.

Goodstein's theorem

$$(\forall\, Int\, m)(m > 0 \rightarrow (\exists\, Int\, n)(n > 1 \wedge G(m)(n) \doteq 0$$

cannot be derived.
$G(m)(1), G(m)(2), \ldots, G(m)(n), \ldots$ is the Goodstein sequence of m.

The Data Type of Finite Sequences

Core Theory

$seqLen : Seq \to Int$

$seqGet_A : Seq \times Int \to A$ for any type $A \sqsubseteq Any$

$seqGetOutside : Any$

The Data Type of Finite Sequences

Core Theory

$seqLen : Seq \to Int$

$seqGet_A : Seq \times Int \to A$ for any type $A \sqsubseteq Any$

$seqGetOutside : Any$

Definitional Extension

$seqEmpty : Seq$

$seqSingleton : Any \to Seq$

$seqConcat : Seq \times Seq \to Seq$

$seqSub : Seq \times Int \times Int \to Seq$

$seqReverse : Seq \to Seq$

$seqIndexOf : Seq \times Any \to Int$

$seqNPerm(Seq)$

$seqPerm(Seq, Seq)$

$seqSwap : Seq \times Int \times Int \to Seq$

$seqRemove : Seq \times Int \to Seq$

$seqNPermInv : Seq \to Seq$

$seqDepth : Seq \to Int$

Sequence Comprehension

Let le, ri be integers, t a term of arbitrary type, typically containing variable vi:

$$seqDef\{Int\ vi;\}(le, ri, t)$$

is interpreted as the sequence

$$\langle t(le/vi), \ldots, t((ri-1)/vi)\rangle$$

If $ri \leqslant le$ it is the empty sequence.

Sequence Comprehension

Let le, ri be integers, t a term of arbitrary type, typically containing variable vi:

$$seqDef\{Int\ vi;\}(le, ri, t)$$

is interpreted as the sequence

$$\langle t(le/vi), \ldots, t((ri-1)/vi) \rangle$$

If $ri \leqslant le$ it is the empty sequence.

Example

$$seqDef\{Int\ vi;\}(2, 6, vi^2)$$

is interpreted as the sequence

$$\langle 4, 9, 25 \rangle$$

Axioms of the core theory CoT_{seq}

1 $(\forall\, Seq\, s)(0 \leqslant seqLen(s))$

Axioms of the core theory CoT_{seq}

1 $(\forall\, Seq\; s)(0 \leqslant seqLen(s))$

2 $(\forall\, Seq\; s_1, s_2)(s_1 \doteq s_2 \leftrightarrow$
$seqLen(s_1) \doteq seqLen(s_2) \;\wedge$
$(\forall\, Int\; i)(0 \leqslant i < seqLen(s_1) \rightarrow seqGet_{Any}(s_1, i) \doteq seqGet_{Any}(s_2, i)))$

Axioms of the core theory CoT_{seq}

1 $(\forall\, Seq\, s)(0 \leqslant seqLen(s))$

2 $(\forall\, Seq\, s_1, s_2)(s_1 \doteq s_2 \leftrightarrow$
$seqLen(s_1) \doteq seqLen(s_2) \ \wedge$
$(\forall\, Int\, i)(0 \leqslant i < seqLen(s_1) \rightarrow seqGet_{Any}(s_1, i) \doteq seqGet_{Any}(s_2, i)))$

3 $(\forall\, Int\, ri, le)($
$(le < ri \rightarrow seqLen((seqDef\{u\}(le, ri, t)) \doteq ri - li)$
$\wedge$
$(ri \leqslant le \rightarrow seqLen(seqDef\{u\}(le, ri, t)) \doteq 0))$

Axioms of the core theory CoT_{seq}

1. $(\forall\, Seq\, s)(0 \leqslant seqLen(s))$

2. $(\forall\, Seq\, s_1, s_2)(s_1 \doteq s_2 \leftrightarrow$
   $seqLen(s_1) \doteq seqLen(s_2) \;\wedge$
   $(\forall\, Int\, i)(0 \leqslant i < seqLen(s_1) \rightarrow seqGet_{Any}(s_1, i) \doteq seqGet_{Any}(s_2, i)))$

3. $(\forall\, Int\, ri, le)($
   $(le < ri \rightarrow seqLen((seqDef\{u\}(le, ri, t)) \doteq ri - li)$
   $\wedge$
   $(ri \leqslant le \rightarrow seqLen(seqDef\{u\}(le, ri, t)) \doteq 0))$

4. $(\forall\, Int\, i, ri, le)(\forall\, Any\, \bar{x})(((0 \leqslant i \wedge i < ri - le) \rightarrow$
   $seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(t\{(le + i)/u\}))$
   $\wedge$
   $(\neg(0 \leqslant i \wedge i < ri - le) \rightarrow$
   $seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(seqGetOutside)))$

A Model for CoT_{seq}

The type domain D^{Seq}

Inductive Definition:

$$
\begin{array}{lcl}
U & = & D^{Any} \setminus D^{Seq} \\
D^0_{\mathrm{Seq}} & = & \{\langle\rangle\} \\
D^{n+1}_{\mathrm{Seq}} & = & \{\langle a_1, \ldots, a_k \rangle \mid k \in \mathbb{N} \text{ and } a_i \in D^n_{Seq} \cup U, 1 \leqslant i \leqslant k\}, n \geqslant 0
\end{array}
$$

$$
D^{Seq} \quad := \quad \bigcup_{n \geqslant 0} D^n_{\mathrm{Seq}}
$$

A Model for CoT_{seq}

Interpretaion of the Vocabulary of CoT_{seq}

1. $seqGet_A^{\mathcal{M}}(\langle a_0, \ldots, a_{n-1}\rangle, i) =$
$$\begin{cases} cast_A^{\mathcal{M}}(a_i) & \text{if } 0 \leqslant i < n \\ cast_A^{\mathcal{M}}(seqGetOutside^{\mathcal{M}}) & \text{otherwise} \end{cases}$$
2. $seqLen^{\mathcal{M}}(\langle a_0, \ldots, a_{n-1}\rangle) = n$
3. $seqGetOutside^{\mathcal{M}} \in D^{Any}$ arbitrary.
4. $seqDef\{iv\}(le, ri, e)^{\mathcal{M},\beta} =$
$$\begin{cases} \langle a_0, \ldots a_{k-1}\rangle & \text{if } ri - le = k > 0 \text{ and for all } 0 \leqslant i < k \\ & a_i = e^{\mathcal{M},\beta_i} \text{ with } \beta_i = \beta[le + i/iv] \\ \langle\rangle & \text{otherwise} \end{cases}$$

Consistency

Theorem

The theory CoT_{seq} is consistent.

Consistency

Theorem

The theory CoT_{seq} is consistent.

Proof

Check that all axioms of CoT_{seq} are true in the model $\mathcal{M}$.

Definitional Extension T_{seq}

Set 1

$$seqEmpty \doteq seqDef\{iv\}(0, 0, x)$$

Definitional Extension T_{seq}

Set 1

$$seqEmpty \doteq seqDef\{iv\}(0, 0, x)$$

$$(\forall\, Any\, x)(seqSingleton(x) \doteq seqDef\{iv\}(0, 1, x))$$

Definitional Extension T_{seq}

Set 1

$$seqEmpty \doteq seqDef\{iv\}(0, 0, x)$$

$$(\forall\, Any\; x)(seqSingleton(x) \doteq seqDef\{iv\}(0, 1, x))$$

$$\begin{array}{l}(\forall\, Seq\; s_1, s_2)(seqConcat(s_1, s_2) \doteq \\ \quad seqDef\{iv\}(0, Len(s_1) + Len(s_2), \begin{array}[t]{l}\text{if } iv < Len(s_1) \\ \text{then } seqGet_{Any}(s_1, iv) \\ \text{else } seqGet_{Any}(s_2, iv - Len(s_1))\end{array}))\end{array}$$

Definitional Extension T_{seq}

Set 1

$$seqEmpty \doteq seqDef\{iv\}(0, 0, x)$$

$$(\forall\, Any\; x)(seqSingleton(x) \doteq seqDef\{iv\}(0, 1, x))$$

$$\begin{array}{l}(\forall\, Seq\; s_1, s_2)(seqConcat(s_1, s_2) \doteq \\ \quad seqDef\{iv\}(0, Len(s_1) + Len(s_2), \begin{array}{l}\text{if } iv < Len(s_1) \\ \text{then } seqGet_{Any}(s_1, iv) \\ \text{else } seqGet_{Any}(s_2, iv - Len(s_1))\end{array}))\end{array}$$

$$(\forall\, Seq\; s)(\forall\, Int\; i, j)(seqSub(s, i, j) \doteq seqDef\{iv\}(i, j, seqGet_{Any}(s, iv)))$$

Definitional Extension T_{seq}

Set 1

$$seqEmpty \doteq seqDef\{iv\}(0, 0, x)$$

$$(\forall\, Any\, x)(seqSingleton(x) \doteq seqDef\{iv\}(0, 1, x))$$

$$\begin{array}{l}(\forall\, Seq\, s_1, s_2)(seqConcat(s_1, s_2) \doteq \\ \quad seqDef\{iv\}(0, Len(s_1) + Len(s_2), \begin{array}[t]{l}\text{if } iv < Len(s_1) \\ \text{then } seqGet_{Any}(s_1, iv) \\ \text{else } seqGet_{Any}(s_2, iv - Len(s_1))\end{array}))\end{array}$$

$$(\forall\, Seq\, s)(\forall\, Int\, i, j)(seqSub(s, i, j) \doteq seqDef\{iv\}(i, j, seqGet_{Any}(s, iv)))$$

$$\begin{array}{l}(\forall\, Seq\, s)(seqReverse(s) \doteq \\ \quad seqDef\{iv\}(0, Len(s), seqGet_{Any}(s, Len(s) - iv - 1)))\end{array}$$

For concise presentation we have used *Len* instead of *seqLen*.

Definitional Extension T_{seq}

Set 2

$$(\forall\, Seq\, s)(seqNPerm(s) \leftrightarrow$$
$$(\forall\, Int\, i)(0 \leqslant i < Len(s) \rightarrow (\exists\, Int\, j)(0 \leqslant j < Len(s) \wedge seqGet_{Int}(s, j) \doteq i)))$$

$$(\forall\, Seq\, s_1, s_2)(seqPerm(s_1, s_2) \leftrightarrow Len(s_1) \doteq Len(s_2) \wedge$$
$$(\exists\, Seq\, s)(Len(s) \doteq Len(s_1) \wedge seqNPerm(s) \wedge$$
$$(\forall\, Int\, i)(0 \leqslant i < Len(s) \rightarrow$$
$$seqGet_{Any}(s_1, i) \doteq seqGet_{Any}(s_2, seqGet_{Int}(s, i))))))))))$$

For concise presentation we have again used *Len* instead of *seqLen*.

Consistency

Theorem

The theory T_{seq} is consistent.

Consistency

Theorem

The theory T_{seq} is consistent.

Proof

We refer to

If T_2 is obtained from T_1 by definitional extension then T_2 is consistent iff T_1 is consistent.

and the fact that T_{seq} is a definitional extension of CoT_{seq}.

Relative Completeness

Theorem

If the union of the component theories is complete then

$$T_{seq} \text{ is complete}$$

provided that the function symbol and axioms for *seqDepth* are added.

$$\begin{aligned} seqDepth(s) &= 0 \quad \text{if } \neg instance_{\mathrm{Seq}}(s) \\ seqDepth(s) &= max\{seqDepth(seqGet_{Seq}(s,i)) \mid \\ &\quad 0 \leqslant i < seqLen(s) \wedge instance_{\mathrm{Seq}}(seqGet_{Seq}(s,i))\} + 1 \end{aligned}$$

Derived Theorems in T_{seq}

1 getOfSeqConcat

$$(\forall\, Seq\; s, s2)(\forall\, Int\; i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq$$
$$\text{if } \; i < Len(s) \text{ then } \; seqGet_{alpha}(s, i) \text{ else } \; seqGet_{alpha}(s2, i - Len(s))$$

Derived Theorems in T_{seq}

1 $getOfSeqConcat$

$$(\forall\, Seq\, s, s2)(\forall\, Int\, i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq \text{if } \ i < Len(s) \text{ then } \ seqGet_{alpha}(s, i) \text{ else } \ seqGet_{alpha}(s2, i - Len(s))$$

KeY Demo

Derived Theorems in T_{seq}

1 g*etOfSeqConcat*

$$(\forall\, Seq\; s, s2)(\forall\, Int\; i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq$$
$$\text{if } \; i < Len(s) \text{ then } \; seqGet_{alpha}(s, i) \text{ else } \; seqGet_{alpha}(s2, i - Len(s))$$

Derived Theorems in T_{seq}

1 getOfSeqConcat
$(\forall\, Seq\; s, s2)(\forall\, Int\; i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq$
$\text{if } \; i < Len(s) \text{ then } \; seqGet_{alpha}(s, i) \text{ else } \; seqGet_{alpha}(s2, i - Len(s))$

2 getOfSeqSub
$(\forall\, Seq\; s)(\forall\, Int\; f, t, i)(seqGet_{alpha}(seqSub(s, f, t), i) \doteq$
$\text{if } 0 \leqslant i \wedge i < (t - f) \text{ then } \; seqGet_{alpha}(s, i + f) \text{ else } \; (alpha)seqGetOutside$

Derived Theorems in T_{seq}

1. getOfSeqConcat
$(\forall\, Seq\; s, s2)(\forall\, Int\; i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq$
$\text{if } \; i < Len(s) \text{ then } \; seqGet_{alpha}(s, i) \text{ else } \; seqGet_{alpha}(s2, i - Len(s))$
2. getOfSeqSub
$(\forall\, Seq\; s)(\forall\, Int\; f, t, i)(seqGet_{alpha}(seqSub(s, f, t), i) \doteq$
$\text{if } 0 \leqslant i \wedge i < (t - f) \text{ then } \; seqGet_{alpha}(s, i + f) \text{ else } \; (alpha)seqGetOutside$
3. lenOfSeqConcat
$(\forall\, Seq\; s, s2)(Len(seqConcat(s, s2)) \doteq Len(s) + Len(s2))$

Derived Theorems in T_{seq}

1. getOfSeqConcat
   $(\forall\, Seq\, s, s2)(\forall\, Int\, i)(seqGet_{alpha}(seqConcat(s, s2), i) \doteq$
   $\text{if } \ i < Len(s) \text{ then } \ seqGet_{alpha}(s, i) \text{ else } \ seqGet_{alpha}(s2, i - Len(s))$
2. getOfSeqSub
   $(\forall\, Seq\, s)(\forall\, Int\, f, t, i)(seqGet_{alpha}(seqSub(s, f, t), i) \doteq$
   $\text{if } 0 \leqslant i \wedge i < (t - f) \text{ then } \ seqGet_{alpha}(s, i + f) \text{ else } \ (alpha)seqGetOutside$
3. lenOfSeqConcat
   $(\forall\, Seq\, s, s2)(Len(seqConcat(s, s2)) \doteq Len(s) + Len(s2))$
4. lenOfSeqSub
   $(\forall\, Seq\, s)(\forall\, Int\, from, to)(Len(seqSub(s, from, to)) \doteq$
   $\text{if } \ from < to \ \text{ then } \ (to - from) \ \text{ else } \ 0)$

Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up

Requirements on the KeY Calculus, Re-Revisited

for serious program verification

- Full typed first-order logic
- Partially ordered extensible type hierarchies reflecting Java's type system
- Rich expressive power, even if theoretically redundant
- Coverage of partial functions
- Combination of automatic and interactive proving
- Extensible: many theories
- Supportive user interface

THE END

Derived Theorems on Permutations

seqNPermRange

$$(\forall\, Seq\, s)(\forall\, Int\, i)(seqNPerm(s) \wedge 0 \leqslant i \wedge i < Len(s))$$
$$\quad \rightarrow (0 \leqslant seqGet_{\mathrm{Int}}(s,i) \wedge seqGet_{\mathrm{Int}}(s,i) < Len(s))$$

seqNPermInjective

$$(\forall\, Seq\, s)(\forall\, Int\, i,j)(seqNPerm(s) \wedge 0 \leqslant i \wedge i < Len(s) \wedge$$
$$seqGet_{\mathrm{Int}}(s,i) \doteq seqGet_{\mathrm{Int}}(s,j)) \wedge 0 \leqslant j \wedge j < Len(s)$$
$$\quad \rightarrow i \doteq j)$$

seqNPermComp

$$(\forall\, Seq\, s1, s2)(seqNPerm(s1) \wedge seqNPerm(s2) \wedge Len(s1) \doteq Len(s2) \rightarrow$$
$$seqNPerm(seqDef\{u\}(0, Len(s1), seqGet_{\mathrm{Int}}(s1, seqGet_{\mathrm{Int}}(s2,u))))))$$

seqPermTrans

$$(\forall\, Seq\, s1, s2, s3)(seqPerm(s1,s2) \wedge seqPerm(s2,s3)$$
$$\quad \rightarrow seqPerm(s1,s3))$$

seqPermRefl $\quad (\forall\, Seq\, s)(seqPerm(s,s))$

Hereditary base-n notation
Taken from Wikipedia

The hereditary base-n notation for a natural number m is obtained from its ordinary base-n notation

$$m = m_k \cdot n^k + m_{k-1} \cdot n^{k-1} + \ldots m_1 \cdot n + m_0, \quad 0 \leqslant m_i < n, m_k \neq 0$$

by also writing the exponents k, $k-1$, ..., $n+1$ in base-n notation and again the thus arising exponents, and so on.

Hereditary base-n notation

Taken from Wikipedia

The hereditary base-n notation for a natural number m is obtained from its ordinary base-n notation

$$m = m_k \cdot n^k + m_{k-1} \cdot n^{k-1} + \dots m_1 \cdot n + m_0, \quad 0 \leqslant m_i < n, m_k \neq 0$$

by also writing the exponents k, $k-1$, ..., $n+1$ in base-n notation and again the thus arising exponents, and so on.

Example

base-2	35	$=$	$2^5 + 2^1 + 2^0$
hereditary base-2	35	$=$	$2^{2^2+1} + 2 + 1$
base-3	100	$=$	$3^4 + 2 \cdot 3^2 + 3^0$
hereditary base-3	100	$=$	$3^{3+1} + 2 \cdot 3^2 + 1.$

Goodstein Sequences

Taken from Wikipedia

Definition

$G(m)(1) = m$	
$G(m)(n+1)$	write $G(m)(n)$ in hereditary base-$(n+1)$ notation
	replace all bases $n+1$ by $n+2$
	subtract 1

Goodstein Sequences

Taken from Wikipedia

Definition

$$
\begin{array}{ll}
G(m)(1) = m & \\
G(m)(n+1) & \text{write } G(m)(n) \text{ in hereditary base-}(n+1) \text{ notation} \\
 & \text{replace all bases } n+1 \text{ by } n+2 \\
 & \text{subtract } 1
\end{array}
$$

Example

Base	Hered. not.	$G(3)(n)$	Notes
2	$2^1 + 1$	3	Write 3 in base 2 notation
3	$3^1 + 1 - 1 = 3^1$	3	Switch 2 to 3, subtract 1
4	$4^1 - 1 = 3$	3	Switch 3 to 4, subtract 1.No 4s left
5	$3 - 1 = 2$	2	No 4s left. Just subtract 1
6	$2 - 1 = 1$	1	No 5s left. Just subtract 1
7	$1 - 1 = 0$	0	No 6s left. Just subtract 1