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
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 $\Sigma = (\text{Func}, \text{Pred}, \text{Var})$

- Type spec $f : T_1 \times \ldots T_n \rightarrow T_0$ required for $f \in \text{Func}$,
- Type spec $p : T_1 \times \ldots T_n$ required for $p \in \text{Pred}$,
- Type spec $x : T$ required for $x \in \text{Var}$,
Typed First-Order Logic

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

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

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

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

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 $\Sigma = (\text{Func}, \text{Pred}, \text{Var})$

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

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

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$. 

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

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

is a term of type $T_0$.
Typed First-Order Logic

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

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

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

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

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

For $f \in \text{Func}$ with $f : T_1 \times \ldots T_n \rightarrow T_0$ and terms $\sigma(t_i) \sqsubseteq T_i$

$f(t_1, \ldots, t_n)$

is a term of type $T_0$.

Definition of Formulas Unchanged
How to Deal with Typed Logic

Transform into untyped predicate logic

How to Deal with Typed Logic

Transform into untyped predicate logic

- 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

- **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**
- 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

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

allLeft \[ \frac{\Gamma, (\forall A x)\varphi \Rightarrow \Delta}{\Gamma \vdash (\forall A x)\varphi, \Delta} \]
\[ t \text{ a ground term, } \sigma(t) \subseteq A \]

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

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

close False \[ \frac{\Gamma, false \Rightarrow \Delta}{\Gamma, \varphi \Rightarrow \varphi, \Delta} \]

close True \[ \frac{\Gamma \Rightarrow true, \Delta}{\Gamma \Rightarrow true, \Delta} \]

\[ \Gamma, \varphi \Rightarrow \varphi, \Delta \]
Equational Rules

**eqLeft**

\[
\frac{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta}{\Gamma, t_1 \vdash t_2, [z/t_2](\varphi) \Rightarrow \Delta}
\]

if \(\sigma(t_2) \sqsubseteq \sigma(t_1)\)

**eqRight**

\[
\frac{\Gamma, t_1 \vdash t_2 \Rightarrow [z/t_2](\varphi), [z/t_1](\varphi), \Delta}{\Gamma, t_1 \vdash t_2 \Rightarrow [z/t_1](\varphi), \Delta}
\]

if \(\sigma(t_2) \sqsubseteq \sigma(t_1)\)

**eqSymmLeft**

\[
\frac{\Gamma, t_2 \vdash t_1 \Rightarrow \Delta}{\Gamma, t_1 \vdash t_2 \Rightarrow \Delta}
\]

**eqReflLeft**

\[
\frac{\Gamma, t \vdash t \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}
\]
A Closer Look at eqLeft

\[
\text{eqLeft} \quad \frac{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta}{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi) \Rightarrow \Delta} \quad \text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)
\]
A Closer Look at \texttt{eqLeft}

\[
\begin{align*}
\text{eqLeft} & \quad \Gamma, t_1 \doteq t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta \\
& \quad \Gamma, t_1 \doteq t_2, [z/t_1](\varphi) \Rightarrow \Delta \\
& \quad \text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)
\end{align*}
\]

Why is the side condition \(\sigma(t_2) \sqsubseteq \sigma(t_1)\) necessary?
A Closer Look at $\text{eqLeft}$

\[
\begin{align*}
\text{eqLeft} & \quad \frac{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta}{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi) \Rightarrow \Delta} \\
\text{if } \sigma(t_2) \sqsubseteq \sigma(t_1)
\end{align*}
\]

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

Consider the signature:

$B \nleq A, a : \rightarrow A, b : \rightarrow B, p : B$. 
A Closer Look at eqLeft

\[
\text{eqLeft} \quad \frac{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta}{\Gamma, t_1 \vdash t_2, [z/t_1](\varphi) \Rightarrow \Delta}
\]

if \( \sigma(t_2) \sqsubseteq \sigma(t_1) \)

Why is the side condition \( \sigma(t_2) \sqsubseteq \sigma(t_1) \) necessary?
Consider the signature:
\( B \not< A, \ a : \rightarrow A, \ b : \rightarrow B, \ p : B. \)
Applying eqLeft without side condition on the sequent

\[
b \vdash a, p(b) \Rightarrow
\]

would result in

\[
b \vdash a, p(b), p(a) \Rightarrow
\]

with \( p(a) \) being not well-typed.
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:

\[
\text{eqLeft} \quad \frac{\Gamma, t_1 \doteqdot t_2, [z/t_1](\varphi), [z/t_2](\varphi) \Rightarrow \Delta}{\Gamma, t_1 \doteqdot t_2, [z/t_1](\varphi) \Rightarrow \Delta}
\]

provided $[z/t_2](\varphi)$ is welltyped

Likewise for eqRight.
Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up
\[ \neg (\exists x)(\exists y)(x \triangleleft y) \]

is tautology for this type hierarchy

The phenomenon that universal validity of a formula depends on symbols not occurring in it, is highly undesirable.
\( \neg(\exists x)(\exists y)(x \neq y) \) is tautology for this type hierarchy.

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

---

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

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

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

The phenomenon that universal validity of a formula depends on symbols not occurring in it, is highly undesirable.
Verification of programs using these classes should remain valid if another subclass is added to java.util.AbstractList.
Verification of programs using these classes should remain valid if another subclass is added to `java.util.AbstractList`. 
Type Hierarchies Extensions

\[(\text{TSym}_1, \sqsubseteq_1) \sqsubseteq (\text{TSym}_2, \sqsubseteq_2)\]
Type Hierarchies Extensions

\[(\text{TSym}_1, \sqsubseteq_1) \sqsubseteq (\text{TSym}_2, \sqsubseteq_2)\]

Only subtype relations may be added.
Logical Consequence Relation

Definition of Super-Consequence Relation

φ ∈ Fm_{T,\Sigma}, \Phi ⊆ Fm_{T,\Sigma}
for type hierarchy T and signature \Sigma.

\[ \Phi ⊨ \varphi \text{ iff for all type hierarchies } T' \text{ with } T ⊆ T' \]
and all \( T' - \Sigma \)-structures \( M \)
if \( M ⊨ \Phi \) then \( M ⊨ \varphi \)
Definition of Super-Consequence Relation

\[ \varphi \in \text{Fml}_{T, \Sigma}, \ \Phi \subseteq \text{Fml}_{T, \Sigma} \]
for type hierarchy \( T \) and signature \( \Sigma \).

\[ \Phi \models \varphi \ \text{iff} \ \text{for all type hierarchies} \ T' \text{ with} \ T \subseteq T' \]
\[ \text{and all} \ T' - \Sigma\text{-structures} \ M \]
\[ \text{if} \ M \models \Phi \text{ then} \ M \models \varphi \]
Definition of Super-Consequence Relation

\( \varphi \in \text{Fml}_{\mathcal{T}, \Sigma}, \ \Phi \subseteq \text{Fml}_{\mathcal{T}, \Sigma} \)

for type hierarchy \( \mathcal{T} \) and signature \( \Sigma \).

\[ \Phi \vdash \varphi \ \text{iff} \ \text{for all type hierarchies} \ \mathcal{T}' \text{ with} \ \mathcal{T} \subseteq \mathcal{T}' \text{ and all} \ \mathcal{T}' - \Sigma\text{-structures} \ \mathcal{M} \text{ if} \ \mathcal{M} \models \Phi \text{ then} \ \mathcal{M} \models \varphi \]

For the example on the previous slide.

\[ \emptyset \not\vdash \neg(\exists x)(\exists y)(x \not= y) \]

\[ A \]

\[ B \]

\[ A : x \]

\[ B : y \]
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.
Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up
Conditional Terms

Syntax

(if \( \varphi \) then \( t_1 \) else \( t_2 \)) is a term of type \( A \)
for a formula \( \varphi \) and terms \( t_i \) of type \( A_i \);
if \( A_2 \subseteq A_1 = A \) or \( A_1 \subseteq A_2 = A \).
Conditional Terms

Syntax

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

Semantics by Elimination (Example)

$(\forall x, y)(x \leq (if \ x \leq y \ then \ y \ else \ x))$
is replaced by

$(\forall x, y)(x \leq y \rightarrow x \leq y \land \neg(x \leq y) \rightarrow y \leq y)$
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 $N \geq 0$ and $a[i] \geq 0$ for $0 \leq i < N$, prove the post-condition that $\text{sum} \leq N \times \text{max}$
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 $N \geq 0$ and $a[i] \geq 0$ for $0 \leq i < N$, prove the post-condition that $\text{sum} \leq N \times \text{max}$

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

CADE Tutorial: The KeY Calculus: TU Darmstadt, KIT

150803 17 / 46
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

<table>
<thead>
<tr>
<th>Mathematical Notation</th>
<th>KeY Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{b_0 \leq vi &lt; b_1} s_1$</td>
<td>$bsum{ Int \ vi; }(b_0, b_1, s_1)$</td>
</tr>
<tr>
<td>$\prod_{b_0 \leq vi &lt; b_1} s_1$</td>
<td>$prod{ Int \ vi; }(b_0, b_1, s_1)$</td>
</tr>
<tr>
<td>$\bigcup_{-\infty &lt; vi &lt; \infty} s_2$</td>
<td>$infiniteUnion{ Int \ vi; }(s_2)$</td>
</tr>
<tr>
<td>$\langle s_3[b_0/vi], \ldots, s_3[(b_1 - 1)/vi] \rangle$</td>
<td>$seqDef{ Int \ vi; }(b_0, b_1, s_3)$</td>
</tr>
</tbody>
</table>

$b_0$, $b_1$, $s_1$ terms of type $Int$,  
s_2 term of type $LocSet$,  
s_3 term of type $Seq$,  
$vi$ would typically occur free in $s_1$, $s_2$, $s_3$
Theorem
For every formula containing variable binders there is a satisfiability equivalent formula without.
Theorem

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

Example

\[(\forall \text{Int } N)(\text{bsum\{}\text{Int } vi; }\}(0, N, a[ri]) \leq N \times \text{max})\]

is replaced by

\[
f(0) = 0 \quad \land \\
(\forall \text{Int } j)(f(j + 1) = f(j) + a[j]) \quad \land \\
(\forall \text{Int } N)(f(N) \leq N \times \text{max})
\]
Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up
Ways to deal with undefinedness

1. Remove it.
   e.g. set the empty sum to 0: \( \sum_{i=-2}^{i=0} s_i = 0 \).
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.
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. 
   \((\exists i)(\frac{1}{0} = i)\) 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.

$(\exists i)\left(\frac{1}{0} = i\right)$ is a tautology.

$\frac{1}{0} \div \frac{2}{0}$ 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.
   \[(\exists i)(\frac{1}{0} = i)\] is a tautology.
   \[\frac{1}{0} = \frac{2}{0}\] is not.
   Also \(\text{cast}_{\text{Int}}(c) \div 5 \rightarrow c \div 5\) is not a tautology. In case \(c\) is not of type \(\text{Int}\) the underspecified value for \(\text{cast}_{\text{Int}}(c)\) could be 5.
Typed Logic

Modular Reasoning

Extension of First-Order Logic

Undefinedness

Theories

Wrap-Up
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  $0 < i \lor 0 = i \lor 0 < (-i)$
O.2  $0 < i \land 0 < j \rightarrow 0 < i + j$
O.3  $0 < i \land 0 < j \rightarrow 0 < i * j$
O.4  $i < j \leftrightarrow 0 < j + (-i)$
O.5  $\neg(0 < 0)$
Ind  $\phi(0) \land (\forall i)(0 \leq i \land \phi(i) \rightarrow \phi(i + 1)) \rightarrow (\forall i)(0 \leq i \rightarrow \phi(i))$
Transitive Closure

Definition

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

\[
reachR(g, h, 0) \leftrightarrow g \equiv h
\]
\[
n \geq 0 \rightarrow (reachR(g, h, n + 1) \leftrightarrow (\exists G k)(reachR(g, k, n) \land R(k, h)))
\]
\[
TR(g, h) \leftrightarrow (\exists \text{Int } n)(n \geq 0 \land reachR(g, h, n))
\]

Here, $g, h$ are variables of type $G$ and $n$ of type $\text{Int}$. 
Transitive Closure

Definition

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

\[
\begin{align*}
\text{reach}_R(g, h, 0) & \iff g \vdash h \\
0 \leq n & \rightarrow (\text{reach}_R(g, h, n + 1) \iff \exists G \ k (\text{reach}_R(g, k, n) \land R(k, h))) \\
\text{TR}(g, h) & \iff \exists \text{Int} \ n (n \geq 0 \land \text{reach}_R(g, h, n))
\end{align*}
\]

Here, \(g, h\) are variables of type \(G\) and \(n\) of type \(\text{Int}\).
Transitive Closure

Definition

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

\[
\begin{align*}
\text{reach}_R(g, h, 0) & \iff g \vdash h \\
n \geq 0 \rightarrow (\text{reach}_R(g, h, n + 1) & \iff (\exists G k)(\text{reach}_R(g, k, n) \land R(k, h))) \\
TR(g, h) & \iff (\exists \text{Int } n)(n \geq 0 \land \text{reach}_R(g, h, n))
\end{align*}
\]

Here, \(g, h\) are variables of type \(G\) and \(n\) of type \(\text{Int}\).

\(TR\) is the transitive closure of \(R\).
Transitive Closure

**Definition**

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

\[
\begin{align*}
reach_R(g, h, 0) & \leftrightarrow g \equiv h \\
n \geq 0 & \rightarrow (reach_R(g, h, n + 1) \leftrightarrow (\exists G k)(reach_R(g, k, n) \land R(k, h))) \\
TR(g, h) & \leftrightarrow (\exists Int n)(n \geq 0 \land reach_R(g, h, n))
\end{align*}
\]

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 \(R\).
A Special Case of the Bellman-Ford Lemma

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

\[
\begin{align*}
\forall G\ g &\quad (g \neq s \rightarrow (\exists G\ h)(R(h, g) \land d(g) = d(h) + 1)) \\
\forall G\ g &\quad (\forall G\ h; (R(h, g) \rightarrow d(g) \leq d(h) + 1))
\end{align*}
\]

Then \(d(g)\) is the length of the shortest path from \(s\) to \(g\).
A Special Case of the Bellman-Ford Lemma

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

\[
\begin{align*}
    d(s) &= 0 \\
    (\forall G\ g)(g \neq s \rightarrow (\exists G\ h)(R(h, g) \land d(g) = d(h) + 1)) \\
    (\forall G\ g)(\forall G\ h; (R(h, g) \rightarrow d(g) \leq d(h) + 1))
\end{align*}
\]

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

$$(\forall \text{Int } m)(m > 0 \rightarrow (\exists \text{Int } n)(n > 1 \land G(m)(n) = 0)$$

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

\( \text{seqLen} : \text{Seq} \rightarrow \text{Int} \)

\( \text{seqGet}_A : \text{Seq} \times \text{Int} \rightarrow A \) \quad \text{for any type} \ A \subseteq \text{Any} \n
\( \text{seqGetOutside} : \text{Any} \)
The Data Type of Finite Sequences

Core Theory

\( \text{seqLen} : \text{Seq} \rightarrow \text{Int} \)

\( \text{seqGet}_A : \text{Seq} \times \text{Int} \rightarrow A \) for any type \( A \sqsubseteq \text{Any} \)

\( \text{seqGetOutside} : \text{Any} \)

Definitional Extension

\( \text{seqEmpty} : \text{Seq} \)

\( \text{seqSingleton} : \text{Any} \rightarrow \text{Seq} \)

\( \text{seqConcat} : \text{Seq} \times \text{Seq} \rightarrow \text{Seq} \)

\( \text{seqSub} : \text{Seq} \times \text{Int} \times \text{Int} \rightarrow \text{Seq} \)

\( \text{seqReverse} : \text{Seq} \rightarrow \text{Seq} \)

\( \text{seqIndexOf} : \text{Seq} \times \text{Any} \rightarrow \text{Int} \)

\( \text{seqNPerm}(\text{Seq}) \)

\( \text{seqPerm}(\text{Seq}, \text{Seq}) \)

\( \text{seqSwap} : \text{Seq} \times \text{Int} \times \text{Int} \rightarrow \text{Seq} \)

\( \text{seqRemove} : \text{Seq} \times \text{Int} \rightarrow \text{Seq} \)

\( \text{seqNPermInv} : \text{Seq} \rightarrow \text{Seq} \)

\( \text{seqDepth} : \text{Seq} \rightarrow \text{Int} \)
Sequence Comprehension

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

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

is interpreted as the sequence

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

If $ri \leq le$ it is the empty sequence.
Sequence Comprehension

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

\[
\text{seqDef}\{\text{Int } vi; \}(le, ri, t)
\]

is interpreted as the sequence

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

If \( ri \leq le \) it is the empty sequence.

Example

\[
\text{seqDef}\{\text{Int } vi; \}(2, 6, vi^2)
\]

is interpreted as the sequence

\[
\langle 4, 9, 25 \rangle
\]
Axioms of the core theory $CoT_{seq}$

1. $(\forall \text{Seq } s)(0 \leq \text{seqLen}(s))$
Axioms of the core theory $\mathcal{CoT}_{seq}$

1. $(\forall Seq s)(0 \leq seqLen(s))$

2. $(\forall Seq s_1, s_2)(s_1 \equiv s_2 \leftrightarrow$
   $seqLen(s_1) = seqLen(s_2) \land$
   $(\forall Int i)(0 \leq i < seqLen(s_1) \rightarrow seqGet_{Any}(s_1, i) = seqGet_{Any}(s_2, i)))$
Axioms of the core theory $CoT_{seq}$

1. $(\forall Seq s)(0 \leq seqLen(s))$
2. $(\forall Seq s_1, s_2)(s_1 \equiv s_2 \leftrightarrow$
   $$\quad seqLen(s_1) \equiv seqLen(s_2) \land$$
   $$\quad (\forall Int i)(0 \leq i < seqLen(s_1) \rightarrow seqGet_{Any}(s_1, i) \equiv seqGet_{Any}(s_2, i)))$
3. $(\forall Int ri, le)($
   $$\quad (le < ri \rightarrow seqLen((seqDef\{u\}(le, ri, t)) \equiv ri - li)$$
   $$\land$$
   $$\quad (ri \leq le \rightarrow seqLen(seqDef\{u\}(le, ri, t)) \equiv 0))$$
Axioms of the core theory $CoT_{\text{seq}}$

1. $(\forall \text{Seq } s)(0 \leq \text{seqLen}(s))$

2. $(\forall \text{Seq } s_1, s_2)(s_1 \equiv s_2 \iff$
   
   $\text{seqLen}(s_1) \equiv \text{seqLen}(s_2) \land$
   
   $(\forall \text{Int } i)(0 \leq i < \text{seqLen}(s_1) \rightarrow \text{seqGet}_{\text{Any}}(s_1, i) \equiv \text{seqGet}_{\text{Any}}(s_2, i)))$

3. $(\forall \text{Int } ri, le)(\neg\text{le} \leq ri \rightarrow \text{seqLen}((\text{seqDef}\{u\}(\text{le}, ri, t)) \equiv ri - li)$
   
   $\land$

   $(\text{ri} \leq \text{le} \rightarrow \text{seqLen}(\text{seqDef}\{u\}(\text{le}, ri, t)) \equiv 0))$

4. $(\forall \text{Int } i, ri, le)(\forall \text{Any } \bar{x})(((0 \leq i \land i < ri - le) \rightarrow$
   
   $\text{seqGet}_{A}(\text{seqDef}\{u\}(\text{le}, ri, t), i) \equiv \text{cast}_{A}(t\{(\text{le} + i)/u\}))$

   $\land$

   $(\neg(0 \leq i \land i < ri - le) \rightarrow$
   
   $\text{seqGet}_{A}(\text{seqDef}\{u\}(\text{le}, ri, t), i) \equiv \text{cast}_{A}(\text{seqGet}_{\text{Outside}}))))$
A Model for $CoT_{\text{seq}}$

The type domain $D_{\text{Seq}}$

Inductive Definition:

- $U = D^{\text{Any}} \setminus D_{\text{Seq}}$
- $D_{\text{Seq}}^0 = \{ \emptyset \}$
- $D_{\text{Seq}}^{n+1} = \{ \langle a_1, \ldots, a_k \rangle \mid k \in \mathbb{N} \text{ and } a_i \in D_{\text{Seq}}^n \cup U, 1 \leq i \leq k \}, n \geq 0$

$$D_{\text{Seq}} = \bigcup_{n \geq 0} D_{\text{Seq}}^n$$
A Model for $CoT_{\text{seq}}$

## Interpretation of the Vocabulary of $CoT_{\text{seq}}$

1. $\text{seqGet}^\mathcal{M}_A(\langle a_0, \ldots, a_{n-1} \rangle, i) =$
   \[
   \begin{cases}
   \text{cast}^\mathcal{M}_A(a_i) & \text{if } 0 \leq i < n \\
   \text{cast}^\mathcal{M}_A(\text{seqGetOutside}^\mathcal{M}) & \text{otherwise}
   \end{cases}
   \]

2. $\text{seqLen}^\mathcal{M}(\langle a_0, \ldots, a_{n-1} \rangle) = n$

3. $\text{seqGetOutside}^\mathcal{M} \in D^\text{Any}$ arbitrary.

4. $\text{seqDef}\{\text{iv}\}(\text{le}, \text{ri}, e)^\mathcal{M},\beta =$
   \[
   \begin{cases}
   \langle a_0, \ldots a_{k-1} \rangle & \text{if } \text{ri} - \text{le} = k > 0 \text{ and for all } 0 \leq i < k \\
   a_i = e^\mathcal{M},\beta_i & \text{with } \beta_i = \beta[\text{le} + i/\text{iv}] \\
   \text{\textcircled{\textcolor{white}{\textit{\textbullet}}}} & \text{otherwise}
   \end{cases}
   \]
Theorem

The theory $CoT_{seq}$ is consistent.
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

\[ seqEmpty \equiv seqDef\{iv\}(0, 0, x) \]
Definitional Extension $T_{seq}$

Set 1

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

$(\forall \text{Any } x)(\text{seqSingleton}(x) \equiv \text{seqDef}\{\text{iv}\}(0, 1, x))$
Definitional Extension $T_{seq}$

Set 1

\[
\text{seqEmpty} \triangleq \text{seqDef}\{\text{iv}\}(0, 0, x)
\]
\[
(\forall \text{Any } x)(\text{seqSingleton}(x) \triangleq \text{seqDef}\{\text{iv}\}(0, 1, x))
\]
\[
(\forall \text{Seq } s_1, s_2)(\text{seqConcat}(s_1, s_2) \triangleq \\
\text{seqDef}\{\text{iv}\}(0, \text{Len}(s_1) + \text{Len}(s_2), \text{if } \text{iv} < \text{Len}(s_1) \text{ then seqGet}_{\text{Any}}(s_1, \text{iv}) \text{ else seqGet}_{\text{Any}}(s_2, \text{iv} - \text{Len}(s_1)))
\]

For concise presentation we have used \text{Len} instead of \text{seqLen}. 

Set 1

\[ \text{seqEmpty} \triangleq \text{seqDef}\{iv\}(0, 0, x) \]

\[(\forall \text{Any } x)(\text{seqSingleton}(x) \triangleq \text{seqDef}\{iv\}(0, 1, x)) \]

\[(\forall \text{Seq } s_1, s_2)(\text{seqConcat}(s_1, s_2) \triangleq \text{seqDef}\{iv\}(0, \text{Len}(s_1) + \text{Len}(s_2), \text{if } iv < \text{Len}(s_1) \text{ then seqGet}_{\text{Any}}(s_1, iv) \text{ else seqGet}_{\text{Any}}(s_2, iv - \text{Len}(s_1))) \]

\[(\forall \text{Seq } s)(\forall \text{Int } i, j)(\text{seqSub}(s, i, j) \triangleq \text{seqDef}\{iv\}(i, j, \text{seqGet}_{\text{Any}}(s, iv))) \]
**Definitional Extension** \( T_{\text{seq}} \)

**Set 1**

\[ \text{seqEmpty} \doteq \text{seqDef}\{\text{iv}\}(0, 0, x) \]

\[ (\forall \text{Any } x)(\text{seqSingleton}(x) \doteq \text{seqDef}\{\text{iv}\}(0, 1, x)) \]

\[ (\forall \text{Seq } s_1, s_2)(\text{seqConcat}(s_1, s_2) \doteq \text{seqDef}\{\text{iv}\}(0, \text{Len}(s_1) + \text{Len}(s_2), \text{if } \text{iv} < \text{Len}(s_1)) \]

\[ \text{then seqGet}_{\text{Any}}(s_1, \text{iv}) \]

\[ \text{else seqGet}_{\text{Any}}(s_2, \text{iv} - \text{Len}(s_1)) \]

\[ (\forall \text{Seq } s)(\forall \text{Int } i, j)(\text{seqSub}(s, i, j) \doteq \text{seqDef}\{\text{iv}\}(i, j, \text{seqGet}_{\text{Any}}(s, \text{iv}))) \]

\[ (\forall \text{Seq } s)(\text{seqReverse}(s) \doteq \text{seqDef}\{\text{iv}\}(0, \text{Len}(s), \text{seqGet}_{\text{Any}}(s, \text{Len}(s) - \text{iv} - 1))) \]

For concise presentation we have used \( \text{Len} \) instead of \( \text{seqLen} \).
Set 2

(∀ Seq s)(seqNPPerm(s) ↔
  (∀ Int i)(0 ≤ i < Len(s) → (∃ Int j)(0 ≤ j < Len(s) ∧ seqGet_{Int}(s, j) = i)))

(∀ Seq s_1, s_2)(seqPerm(s_1, s_2) ↔ Len(s_1) = Len(s_2) ∧
  (∃ Seq s)(Len(s) = Len(s_1) ∧ seqNPPerm(s) ∧
  (∀ Int i)(0 ≤ i < Len(s) →
  seqGet_{Any}(s_1, i) = 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

$$T_{seq}$$ is complete

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

$$seqDepth(s) = \begin{cases} 0 & \text{if } \neg \text{instance}_{Seq}(s) \\ \max \{ seqDepth(seqGet_{Seq}(s, i)) \mid 0 \leq i < seqLen(s) \land \text{instance}_{Seq}(seqGet_{Seq}(s, i)) \} + 1 & \text{otherwise} \end{cases}$$
Derived Theorems in $T_{seq}$

1. $\text{getOfSeqConcat}$
   \[(\forall \text{Seq } s, s2)(\forall \text{Int } i)(\text{seqGet}_{\alpha}(\text{seqConcat}(s, s2), i) \doteq \)
   \text{if } i < \text{Len}(s) \text{ then seqGet}_{\alpha}(s, i) \text{ else seqGet}_{\alpha}(s2, i - \text{Len}(s))\]
1 \textit{getOfSeqConcat} \hfill
\textup{(\forall \text{ Seq } s, s2)(\forall \text{ Int } i)(\text{seqGet}_{\alpha}(\text{seqConcat}(s, s2), i) = \\
\text{if } i < \text{Len}(s) \text{ then seqGet}_{\alpha}(s, i) \text{ else seqGet}_{\alpha}(s2, i - \text{Len}(s)))}
Derived Theorems in $T_{seq}$

1. getOfSeqConcat
   
   $(\forall \text{Seq } s, s2)(\forall \text{Int } i)(\text{seqGet}_{\alpha}(\text{seqConcat}(s, s2), i) \doteq$
   
   if $i < \text{Len}(s)$ then $\text{seqGet}_{\alpha}(s, i)$ else $\text{seqGet}_{\alpha}(s2, i - \text{Len}(s))$
Derived Theorems in $T_{seq}$

1. **getOfSeqConcat**
   \[
   (\forall \text{Seq } s, s2)(\forall \text{Int } i)(\text{seqGet}_{\alpha}(\text{seqConcat}(s, s2), i) \doteq \\
   \text{if } i < \text{Len}(s) \text{ then } \text{seqGet}_{\alpha}(s, i) \text{ else } \text{seqGet}_{\alpha}(s2, i - \text{Len}(s)))
   \]

2. **getOfSeqSub**
   \[
   (\forall \text{Seq } s)(\forall \text{Int } f, t, i)(\text{seqGet}_{\alpha}(\text{seqSub}(s, f, t), i) \doteq \\
   \text{if } 0 \leq i \land i < (t - f) \text{ then } \text{seqGet}_{\alpha}(s, i + f) \text{ else } (\alpha)\text{seqGetOutside}
   \]
Derived Theorems in $T_{seq}$

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

2. `getOfSeqSub`
   $$(\forall \text{Seq } s)(\forall \text{Int } f, t, i)(\text{seqGet}_{\text{alpha}}(\text{seqSub}(s, f, t), i) \doteq \text{if } 0 \leq i \land i < (t - f) \text{ then } \text{seqGet}_{\text{alpha}}(s, i + f) \text{ else } (\text{alpha})\text{seqGetOutside})$$

3. `lenOfSeqConcat`
   $$(\forall \text{Seq } s, s2)(\text{Len}(\text{seqConcat}(s, s2)) \doteq \text{Len}(s) + \text{Len}(s2))$$
Derived Theorems in $T_{seq}$

1. $\text{getOfSeqConcat}$
   \[
   (\forall \text{Seq } s, s2)(\forall \text{Int } i)(\text{seqGet}_{\text{alpha}}(\text{seqConcat}(s, s2), i) \doteq \text{if } i < \text{Len}(s) \text{ then seqGet}_{\text{alpha}}(s, i) \text{ else seqGet}_{\text{alpha}}(s2, i - \text{Len}(s)))
   \]

2. $\text{getOfSeqSub}$
   \[
   (\forall \text{Seq } s)(\forall \text{Int } f, t, i)(\text{seqGet}_{\text{alpha}}(\text{seqSub}(s, f, t), i) \doteq \text{if } 0 \leq i \land i < (t - f) \text{ then seqGet}_{\text{alpha}}(s, i + f) \text{ else } (\alpha)\text{seqGetOutside})
   \]

3. $\text{lenOfSeqConcat}$
   \[
   (\forall \text{Seq } s, s2)(\text{Len} (\text{seqConcat}(s, s2)) \doteq \text{Len}(s) + \text{Len}(s2))
   \]

4. $\text{lenOfSeqSub}$
   \[
   (\forall \text{Seq } s)(\forall \text{Int } from, to)(\text{Len} (\text{seqSub}(s, from, to)) \doteq \text{if } from < to \text{ then } (to - from) \text{ else } 0)
   \]
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
\[(\forall \text{Seq } s)(\forall \text{Int } i)(\text{seqNPerm}(s) \land 0 \leq i \land i < \text{Len}(s)) \rightarrow (0 \leq \text{seqGet}_{\text{Int}}(s, i) \land \text{seqGet}_{\text{Int}}(s, i) < \text{Len}(s))\]

seqNPermInjective
\[(\forall \text{Seq } s)(\forall \text{Int } i, j)(\text{seqNPerm}(s) \land 0 \leq i \land i < \text{Len}(s) \land \text{seqGet}_{\text{Int}}(s, i) \equiv \text{seqGet}_{\text{Int}}(s, j)) \land 0 \leq j \land j < \text{Len}(s) \rightarrow i \equiv j)\]

seqNPermComp
\[(\forall \text{Seq } s_1, s_2)(\text{seqNPerm}(s_1) \land \text{seqNPerm}(s_2) \land \text{Len}(s_1) \equiv \text{Len}(s_2) \rightarrow \text{seqNPerm}((\text{seqDef}\{u\}(0, \text{Len}(s_1), \text{seqGet}_{\text{Int}}(s_1, \text{seqGet}_{\text{Int}}(s_2, u)))))))\]

seqPermTrans
\[(\forall \text{Seq } s_1, s_2, s_3)(\text{seqPerm}(s_1, s_2) \land \text{seqPerm}(s_2, s_3) \rightarrow \text{seqPerm}(s_1, s_3))\]

seqPermRefl
\[(\forall \text{Seq } s)(\text{seqPerm}(s, s))\]
The hereditary base-$n$ notation for a natural number $m$ is obtained from its ordinary base-$n$ notation

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

by also writing the exponents $k, k - 1, \ldots, n + 1$ in base-$n$ notation and again the thus arising exponents, and so on.
The hereditary base-$n$ notation for a natural number $m$ is obtained from its ordinary base-$n$ notation

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

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

**Example**

<table>
<thead>
<tr>
<th>Base</th>
<th>Number</th>
<th>Hereditary Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>base-2</td>
<td>35</td>
<td>$2^5 + 2^1 + 2^0$</td>
</tr>
<tr>
<td>hereditary base-2</td>
<td>35</td>
<td>$2^{2^2+1} + 2 + 1$</td>
</tr>
<tr>
<td>base-3</td>
<td>100</td>
<td>$3^4 + 2 \cdot 3^2 + 3^0$</td>
</tr>
<tr>
<td>hereditary base-3</td>
<td>100</td>
<td>$3^{3+1} + 2 \cdot 3^2 + 1$.</td>
</tr>
</tbody>
</table>


Goodstein Sequences
Taken from Wikipedia

**Definition**

\[ G(m)(1) = m \]
\[ G(m)(n + 1) \text{ write } G(m)(n) \text{ in hereditary base-}(n + 1) \text{ notation} \]
replace all bases \( n + 1 \) by \( n + 2 \)
subtract 1

**Example**

<table>
<thead>
<tr>
<th>Base</th>
<th>Hered. not.</th>
<th>G</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

CADE Tutorial: The KeY Calculus: TU Darmstadt, KIT
Goodstein Sequences

Taken from Wikipedia

**Definition**

\[ G(m)(1) = m \]
\[ G(m)(n + 1) \text{ write } G(m)(n) \text{ in hereditary base-}(n + 1) \text{ notation replace all bases } n + 1 \text{ by } n + 2 \]
\[ \text{subtract 1} \]

**Example**

<table>
<thead>
<tr>
<th>Base</th>
<th>Hered. not.</th>
<th>( G(3)(n) )</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( 2^1 + 1 )</td>
<td>3</td>
<td>Write 3 in base 2 notation</td>
</tr>
<tr>
<td>3</td>
<td>( 3^1 + 1 - 1 = 3^1 )</td>
<td>3</td>
<td>Switch 2 to 3, subtract 1</td>
</tr>
<tr>
<td>4</td>
<td>( 4^1 - 1 = 3 )</td>
<td>3</td>
<td>Switch 3 to 4, subtract 1. No 4s left</td>
</tr>
<tr>
<td>5</td>
<td>( 3 - 1 = 2 )</td>
<td>2</td>
<td>No 4s left. Just subtract 1</td>
</tr>
<tr>
<td>6</td>
<td>( 2 - 1 = 1 )</td>
<td>1</td>
<td>No 5s left. Just subtract 1</td>
</tr>
<tr>
<td>7</td>
<td>( 1 - 1 = 0 )</td>
<td>0</td>
<td>No 6s left. Just subtract 1</td>
</tr>
</tbody>
</table>