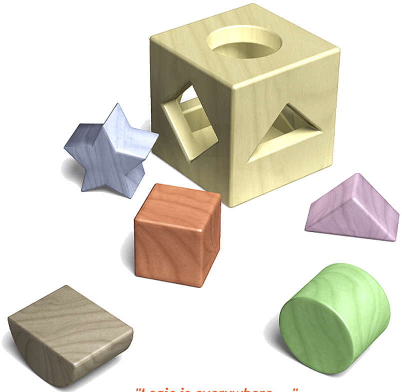
Deduction, Abduction and Induction

Steffen Hölldobler
International Center for Computational Logic
Technische Universität Dresden
Germany

- Introduction
- Deduction
- Sorts
- Abduction
- Induction



"Logic is everywhere ..."

Introduction to Abduction

- Consider $\mathcal{K} \models F$
  where $\mathcal{K}$ is a set of formulas called knowledge base and F is a formula
- In the next example I will use the following propositional atoms:
  grassIsWet, *wheelsAreWet*, *sprinklerIsRunning*, *raining*
- Let $\mathcal{K} = \{g \rightarrow w, \ s \rightarrow w, \ r \rightarrow g\}$
  - Does $\mathcal{K} \models w$ hold?
- Idea   Find an atom A such that $\mathcal{K} \cup \{A\} \models w$ and $\mathcal{K} \cup \{A\}$ is satisfiable
  - $A = w$
  - $A = g$
  - $A = s$ or $A = r$
- This process is called abduction

Introduction to Induction

- Let $\mathcal{K}_{plus} = \{ \ (\forall Y : number)\ plus(0, Y) \approx Y,\ (\forall X, Y : number)\ plus(s(X), Y) \approx s(plus(X, Y)) \ \}$
  - Does $\mathcal{K}_{plus} \models (\forall X, Y : number)\ plus(X, Y) \approx plus(Y, X)$ hold?
- Consider $\mathcal{D} = \mathbb{N} \cup \{\Diamond\}$ and

I	0	s	$plus$
	0	f	$\oplus$

where

  - $f(d) = \begin{cases} f(0) & \text{if } d = \Diamond \\ d + f(0) & \text{if } d \in \mathbb{N} \end{cases}$
  - $d \oplus e = \begin{cases} 0 & \text{if } d = e = \Diamond \\ \Diamond & \text{if } d = 0 \text{ and } e = \Diamond \\ d & \text{if } d \in \mathbb{N}^+ \text{ and } e = \Diamond \\ e & \text{if } d = \Diamond \text{ and } e \in \mathbb{N} \\ d + e & \text{if } d, e \in \mathbb{N} \end{cases}$
  - $+ : \mathbb{N} \to \mathbb{N}$ is the usual addition on $\mathbb{N}$ and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$
- Then $I \models \mathcal{K}_{plus}$ but $(\Diamond \oplus 0) \neq (0 \oplus \Diamond)$ $\rightsquigarrow$ Exercise

The Example Continued

- $\mathcal{K}_{plus} = \{ \; (\forall Y : number)\ plus(0, Y) \approx Y,\quad (\forall X, Y : number)\ plus(s(X), Y) \approx s(plus(X, Y)) \; \}$
  - Does $\mathcal{K}_{plus} \models (\forall X, Y : number)\ plus(X, Y) \approx plus(Y, X)$ hold?
- In order to prove the commutativity of *plus* add Peano's induction principle

$$(P(0) \wedge (\forall M : number)\ (P(M) \rightarrow P(s(M)))) \rightarrow (\forall M : number)\ P(M)$$

  to $\mathcal{K}_{plus}$ (where P is a relational variable)
- For the induction base ($X = 0$) we replace $P(Y)$ by $plus(Y, 0) \approx Y$
- Let $\mathcal{K}_I$ be an appropriate set of induction axioms then

$$\mathcal{K}_{plus} \cup \mathcal{K}_I \models (\forall X, Y : number)\ plus(X, Y) \approx plus(Y, X)$$

- How does $\mathcal{K}_I$ look like? $\leadsto$ Exercise

Deduction, Abduction and Induction

- Peirce 1931 $\mathcal{K}_{facts} \cup \mathcal{K}_{rules} \models G_{result}$
  - **Deduction**
    is an analytic process based on the application of general rules to particular facts, with the inference as a result
  - **Abduction**
    is synthetic reasoning which infers a fact from the rules and the result
  - **Induction**
    is synthetic reasoning which infers a rule from the facts and the result

Deduction

- All reasoning processes considered in the module Foundations so far are deductions
- The logics (first-order, equational) are unsorted
- They can be easily extended to sorted logics
- We will use a sorted logic in the subsection on Induction

Sorts

- $(\forall X, Y)\,(\text{number}(X) \wedge \text{number}(Y) \rightarrow plus(X, Y) \approx plus(Y, X))$
  - $(\forall X, Y : number)\; plus(X, Y) \approx plus(Y, X)$
- A first order language with sorts consists of
  - a first order language $\mathcal{L}(\mathcal{R}, \mathcal{F}, \mathcal{V})$ and
  - a function *sort* : $\mathcal{V} \rightarrow 2^{\mathcal{R}_S}$

  where $\mathcal{R}_S \subseteq \mathcal{R}$ is a finite set of unary predicate symbols called base sorts
- Elements of $2^{\mathcal{R}_S}$ are called sorts; $\emptyset \in 2^{\mathcal{R}_S}$ is called top sort
- We write $X : s$ if $sort(X) = s$
- We assume that for every sort s there are countably many variables $X : s \in \mathcal{V}$

Sorts – Semantics

- Let I be an interpretation with domain $\mathcal{D}$

$$I: \; s = \{p_1, \ldots, p_n\} \mapsto s^I = \mathcal{D} \cap p_1^I \cap \ldots \cap p_n^I$$

  - $I: \; \emptyset \mapsto \mathcal{D}$
- A variable assignment $\mathcal{Z}$ is sorted iff for all $X : s \in \mathcal{V}$ we find $X^{\mathcal{Z}} \in s^I$
- We assume that all sorts are non-empty
- $F^{I,\mathcal{Z}}$ is defined as usual except for

$$[(\exists X : s)\, F]^{I,\mathcal{Z}} = \top \quad \text{iff} \quad \text{there exists } d \in s^I \text{ such that } F^{I,\{X \mapsto d\}\mathcal{Z}} = \top$$

$$[(\forall X : s)\, F]^{I,\mathcal{Z}} = \top \quad \text{iff} \quad \text{for all } d \in s^I \text{ we find } F^{I,\{X \mapsto d\}\mathcal{Z}} = \top$$

Relativization

- Sorted formulas can be mapped onto unsorted ones by means of a **relativization function** *rel*

$$
\begin{array}{lcl}
rel(p(t_1, \ldots, t_n)) & = & p(t_1, \ldots, t_n) \\
rel(\neg F) & = & \neg rel(F) \\
rel(F_1 \wedge F_2) & = & rel(F_1) \wedge rel(F_2) \\
rel(F_1 \vee F_2) & = & rel(F_1) \vee rel(F_2) \\
rel(F_1 \rightarrow F_2) & = & rel(F_1) \rightarrow rel(F_2) \\
rel(F_1 \leftrightarrow F_2) & = & rel(F_1) \leftrightarrow rel(F_2) \\
rel((\forall X : s)\, F) & = & (\forall Y)\, (p_1(Y) \wedge \ldots \wedge p_n(Y) \rightarrow rel(F\{X \mapsto Y\})) \\
 & & \text{if } sort(X) = s = \{p_1, \ldots, p_n\} \text{ and } Y \text{ is a new variable} \\
rel((\exists X : s)\, F) & = & (\exists Y)\, (p_1(Y) \wedge \ldots \wedge p_n(Y) \wedge rel(F\{X \mapsto Y\})) \\
 & & \text{if } sort(X) = s = \{p_1, \ldots, p_n\} \text{ and } Y \text{ is a new variable}
\end{array}
$$

Sorting Function and Relation Symbols

- Each atom of the form $p(t_1, \ldots, t_n)$ can be equivalently replaced by

$$(\forall X_1 \ldots X_n)\ (p(X_1, \ldots, X_n) \leftarrow X_1 \approx t_1 \wedge \ldots \wedge X_n \approx t_n)$$

- Each atom $A\lceil f(t_1, \ldots, t_n)\rceil$ can be equivalently replaced by

$$(\forall X_1 \ldots X_n)\ A\lceil f(t_1, \ldots, t_n)/f(X_1, \ldots, X_n)\rceil \leftarrow X_1 \approx t_1 \wedge \ldots \wedge X_n \approx t_n$$

- Each formula F can be transformed into an equivalent formula F', in which
  - all arguments of function and relation symbols different from $\approx$ are variables and
  - all equations are of the form $t_1 \approx t_2$ or $f(X_1, \ldots, X_n) \approx t$, where $X_1, \ldots, X_n$ are variables and t, t_1, and t_2 are variables or constants
- Sorting the variables occurring in F' effectively sorts the function and relation symbols

Sort Declaration

- F' is usually quite lengthy and cumbersome to read
- If $sort(X) = s$ then the sort declaration for the variable X is

$$X : s$$

- Let s_i, $1 \leq i \leq n$, and s be sorts, f a function and p a relation symbol, both with arity n. Then

$$f : s_1 \times \ldots \times s_n \rightarrow s$$

and

$$p : s_1 \times \ldots \times s_n$$

are sort declarations for f and p, respectively

Abduction

- **Example** Starting a car
- **Applications**
  - fault diagnosis
  - medical diagnosis
  - high level vision
  - natural language understanding
  - reasoning about states, actions, and causality
  - knowledge assimilation

A First Characterization of Abduction

- **Given $\mathcal{K}$ and G; find explanation $\mathcal{K}'$ such that**
  - $\mathcal{K} \cup \mathcal{K}' \models G$ **and**
  - $\mathcal{K} \cup \mathcal{K}'$ **is satisfiable**

  The elements of $\mathcal{K}'$ are said to be abduced

- **Abducing atoms is no real restriction**
- **Weakness of this first characterization**
  We want to abduce causes of effects but no other effects

Restrictions

- **Abducible formulas**
  - set of pre-specified and domain-dependent formulas
  - abduction is restricted to this set
  - default in logic programming: set of undefined predicates
- Typical criteria for choosing a set of abducible formulas
  - an explanation should be **basic**,
    i.e., it cannot be explained by another explanation
  - an explanation should be **minimal**,
    i.e., it cannot be subsumed by another explanation
  - additional information
  - domain-dependent preference criteria
  - integrity constraints

Abductive Framework

- **Abductive framework** $\langle \mathcal{K}, \mathcal{K}_A, \mathcal{K}_{IC} \rangle$ **where**
  - $\mathcal{K}$ **is a set of formulas**
  - $\mathcal{K}_A$ **is a set of ground atoms called abducibles**
  - $\mathcal{K}_{IC}$ **is a set of integrity constraints**
- **Observation** G **is explained by** $\mathcal{K}'$ **iff**
  - $\mathcal{K}' \subseteq \mathcal{K}_A$
  - $\mathcal{K} \cup \mathcal{K}' \models G$ **and**
  - $\mathcal{K} \cup \mathcal{K}'$ **satisfies** $\mathcal{K}_{IC}$
- $\mathcal{K} \cup \mathcal{K}'$ **satisfies** $\mathcal{K}_{IC}$ **iff**
  - $\mathcal{K} \cup \mathcal{K}' \cup \mathcal{K}_{IC}$ **are satisfiable (satisfiability view) or**
  - $\mathcal{K} \cup \mathcal{K}' \models \mathcal{K}_{IC}$ **(theoremhood view)**

Knowledge Assimilation

- **Task** assimilate new knowledge into a given knowledge base
- **Example**
  - $\mathcal{K} = \{\ sibling(X, Y) \leftarrow parents(Z, X) \wedge parents(Z, Y),$
    $parents(X, Y) \leftarrow father(X, Y),$
    $parents(X, Y) \leftarrow mother(X, Y),$
    $father(john, \mathrm{mary}),$
    $mother(jane, \mathrm{mary})\ \}$
  - $\mathcal{K}_{IC} = \{\ X \approx Y \leftarrow father(X, Z) \wedge father(Y, Z),$
    $X \approx Y \leftarrow mother(X, Z) \wedge mother(Y, Z)\ \}$
  - $\mathcal{K}_A = \{A \mid A$ is a ground instance of $father(john, Y)$ or $mother(jane, Y)\}$
  - $\approx$ is a 'built–in' predicate such that
    - $X \approx X$ holds and
    - $s \not\approx t$ holds for all distinct ground terms s and t
  - **Task** assimilate $sibling(\mathrm{mary}, \mathrm{bob})$

The Example Continued

- Two minimal explanations
  - $\{father(john, \text{bob})\}$
  - $\{mother(jane, \text{bob})\}$
- What happens if we additionally observe that $mother(joan, \text{bob})$?
  - belief revision

Theory Revision

- **Default reasoning and jumping to a conclusion**
- **Example**
  - $\mathcal{K} = \{ \quad penguin(X) \rightarrow bird(X),$
    $birdsFly(X) \rightarrow (bird(X) \rightarrow fly(X)),$
    $penguin(X) \rightarrow \neg fly(X),$
    $penguin(tweedy),$
    $bird(john) \quad \}$
  - $\mathcal{K}_{IC} = \emptyset$
  - $\mathcal{K}_A = \{A \mid A$ is a ground instance of $birdsFly(X)\}$
- **Task 1** Explain $fly(john)$
- **Task 2** Explain $fly(tweedy)$
- **What happens if we additionally observe $penguin(john)$?**

Abduction and Model Generation

- **Example**
  - $\mathcal{K} = \{ \quad wobblyWheel \leftrightarrow brokenSpokes \vee flatTyre, \; flatTyre \leftrightarrow puncturedTube \vee leakyValve \quad \}$
  - $\mathcal{K}_{IC} = \emptyset$
  - $\mathcal{K}_A = \{brokenSpokes, \; puncturedTube, \; leakyValve\}$

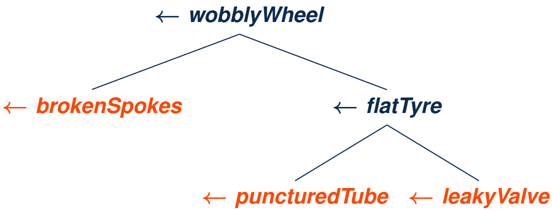
- $\mathcal{K} = \mathcal{K}_{\leftarrow} \cup \mathcal{K}_{\rightarrow}$ **where**
  - $\mathcal{K}_{\leftarrow} = \{ \quad wobblyWheel \leftarrow brokenSpokes, \; wobblyWheel \leftarrow flatTyre, \; flatTyre \leftarrow puncturedTube, \; flatTyre \leftarrow leakyValve \quad \}$
  - $\mathcal{K}_{\rightarrow} = \{ \quad wobblyWheel \rightarrow brokenSpokes \vee flatTyre, \; flatTyre \rightarrow puncturedTube \vee leakyValve \quad \}$

The Wobbly–Wheel Example

- **Observation** *wobblyWheel*
- **What are the minimal and basic explanations?**
- **How can these explanation be computed?**
  - **SLD–resolution**
  - **Model generation**

Abduction and SLD–Resolution

- Consider the SLD-derivation tree for $\leftarrow$ *wobblyWheel* wrt $\mathcal{K}_{\leftarrow}$



Abduction and Model Generation

- Remember $\mathcal{K}_{\rightarrow} = \{ \; \mathit{wobblyWheel} \rightarrow \mathit{brokenSpokes} \vee \mathit{flatTyre}, \; \mathit{flatTyre} \rightarrow \mathit{puncturedTube} \vee \mathit{leakyValve} \; \}$
- Add *wobblyWheel* to $\mathcal{K}_{\rightarrow}$
- What are the minimal models of the extended knowledge base?

$$\{\mathit{wobblyWheel}, \; \mathit{flatTyre}, \; \mathit{puncturedTube}\}$$
$$\{\mathit{wobblyWheel}, \; \mathit{flatTyre}, \; \mathit{leakyValve}\}$$
$$\{\mathit{wobblyWheel}, \; \mathit{brokenSpokes}\}$$

- Restrict these models to the abducible predicates

Mathematical Induction

- **Essential proof technique used to verify properties about recursively defined objects like natural numbers, lists, trees, logic formulas, etc.**
- **Central role in the fields of mathematics, algebra, logic, computer science, formal language theory, etc.**

Some Typical Questions

- Should induction be really used to prove a statement?
- Should the statement be generalized before an attempt is made to prove it by induction?
- Which variable should be the induction variable?
- What induction principle should used?
- What property should be used within the induction principle?
- Should nested induction be taken into account?

Data Structures

- Function symbols are split into constructors and defined function symbols
- Let $\mathcal{F}$ be the set of function symbols
  - Constructors $\mathcal{C} \subseteq \mathcal{F}$
  - Defined function symbols $\mathcal{D} \subseteq \mathcal{F}$
  - $\mathcal{C} \cap \mathcal{D} = \emptyset$
  - $\mathcal{C} \cup \mathcal{D} = \mathcal{F}$
  - $\mathcal{T}(\mathcal{C})$ is called the set of constructor ground terms
- Data structures (or sorts) are sets of constructor ground terms

Data Structures – Examples

- $0 : \textit{number}$
  $s : \textit{number} \to \textit{number}$
  - $\mathcal{T}(\{0, s\}) = \{0, s(0), s(s(0)), \ldots\}$ is called the sort *number*
- $\top : \textit{bool}$
  $\bot : \textit{bool}$
  - $\mathcal{T}(\{\top, \bot\}) = \{\top, \bot\}$ is called the sort *bool*
- $[\,] : \textit{list}(\textit{number})$
  $: : \textit{number} \times \textit{list}(\textit{number}) \to \textit{list}(\textit{number})$
  - $\mathcal{T}([\,], :\}) = \{[\,], [0], [0, 0], [s(0)], \ldots\}$ is called the sort *list(number)*

Well-Sortedness and Selectors

- **Well-Sortedness**
  - Constants and variables are well-sorted
  - If $f : sort_1 \times \ldots \times sort_n \rightarrow sort$
    and for all $1 \leq i \leq n$ we find that t_i is well-sorted and of sort $sort_i$
    then $f(t_1, \ldots, t_n)$ is well-sorted and of sort $sort$
- **Assumption** All terms are well-sorted!
- **Selectors**
  - For each n-ary constructor c we have n unary selectors s_i
    such that for all $1 \leq i \leq n$ we find $s_i(c(t_1, \ldots, t_n)) \approx t_i$

Data Structures – Requirements

- **Different constructors denote different objects**
- **Constructors are injective**
- **Each object can be denoted as an application of some constructor to its selectors (if any exist)**
- **Each selector is 'inverse' to the constructor it belongs to**
- **Each selector returns a so-called witness term if applied to a constructor it does not belong to**

Requirements for Numbers

- The requirements can be translated into first order formulas
- The requirements for *number* are

$$\mathcal{K}_{\text{number}} = \{ \begin{array}{l} (\forall N : \mathit{number})\ 0 \not\approx s(N), \\ (\forall N, M : \mathit{number})\ (s(N) \approx s(M) \rightarrow N \approx M), \\ (\forall N : \mathit{number})\ (N \approx 0 \vee N \approx s(p(N))), \\ (\forall N : \mathit{number})\ p(s(N)) \approx N, \\ p(0) \approx 0, \end{array} \}$$

  where
  - p is the selector for the only argument of the constructor s and
  - 0 is the witness term assigned to $p(0)$
- **Note** p is a defined function symbol!

Defined Function Symbols

- Functions are defined on top of data structures
- We define functions with the help of a set of conditional equations, i.e., universally closed equations of the form

$$\forall\, l \approx r \leftarrow Body$$

such that l is a non-variable term (i.e. of the form $g(s_1, \ldots, s_n)$),

$$var(l) \supseteq var(r) \cup var(Body)$$

and *Body* denotes a conjunction of literals

- We sometimes omit the universal quantifiers in writing conditional equations
- g is a **defined function symbol** wrt a set $\mathcal{K}$ of conditional equations if $\mathcal{K}$ contains a conditional equation of the form

$$g(t_1, \ldots, t_n) \approx r \leftarrow Body$$

The set of conditional equations of this form in $\mathcal{K}$ is called **definition of g** wrt $\mathcal{K}$

Defined Function Symbols – Examples

- **Predecessor** on *number* $\mathcal{K}_p$

$$(\forall N : number)\ p(s(N)) \approx N$$
$$p(0) \approx 0$$

- **Addition** on *number* $\mathcal{K}_{plus}$

$$(\forall X, Y : number)\ (plus(X, Y) \approx Y \leftarrow X \approx 0)$$
$$(\forall X, Y : number)\ (plus(X, Y) \approx s(plus(p(X), Y)) \leftarrow X \not\approx 0)$$

- **Less-than** on *number* $\mathcal{K}_{\mathrm{lt}}$

$$(\forall X, Y : number)\ (\mathrm{lt}(X, Y) \approx \bot \leftarrow Y \approx 0)$$
$$(\forall X, Y : number)\ (\mathrm{lt}(X, Y) \approx \top \leftarrow X \approx 0 \wedge Y \not\approx 0)$$
$$(\forall X, Y : number)\ (\mathrm{lt}(X, Y) \approx \mathrm{lt}(p(X), p(Y)) \leftarrow X \not\approx 0 \wedge Y \not\approx 0)$$

Rewriting Extended to Conditional Equations

- Let $\mathcal{K}$ be a finite set of conditional equations
- A term t can be **rewritten** wrt $\mathcal{K}$ iff
  1. t is well-sorted and ground
  2. t contains a subterm of the form $g(t_1, \ldots, t_n)$ where for all $1 \leq i \leq n$ we find that t_i is a constructor ground term
  3. $g(s_1, \ldots, s_n) \approx r \leftarrow \mathit{Body} \in \mathcal{K}$ and
  4. we find an mgu θ for $g(s_1, \ldots, s_n)$ and $g(t_1, \ldots, t_n)$ such that $\mathcal{K} \models \mathit{Body}\theta$
- In this case t is rewritten to the term obtained from t by replacing $g(t_1, \ldots, t_n)$ by $r\theta$
- Note θ is a matcher because t is ground

Cases

- Let $g(s_1, \ldots, s_n) \approx r \leftarrow \mathit{Body}$ be a rule and $X_1, \ldots, X_n$ new variables

$$g(X_1, \ldots, X_n) \approx r \leftarrow X_1 \approx s_1 \wedge \ldots \wedge X_n \approx s_n \wedge \mathit{Body}$$

  is called homogeneous form of this rule

- Example

$$(\forall X, N : \mathit{number})\ (p(X) \approx N \leftarrow X \approx s(N))$$

  is the homogeneous form of

$$(\forall N : \mathit{number})\ p(s(N)) \approx N$$

- Obervation A rule is semantically equivalent to its homogeneous form
- The case of a rule is the condition of its homogeneous form

Programs

- A **program** is a set of clauses consisting of data structure declarations and function definitions
- **Example** $\mathcal{K}_{\textbf{number}} \cup \mathcal{K}_{\textit{plus}}$ is a program

Properties of Programs

- A program $\mathcal{K}$ is
  - **well-formed** iff it can be ordered such that each function symbol occurring in the definition of a function g in $\mathcal{K}$ either is introduced before by a data structure declaration or another function definition or, otherwise, is g in which case the function is **recursive**
  - **well-sorted** iff each term occurring in $\mathcal{K}$ is well-sorted
  - **deterministic** iff for each function definition occurring in $\mathcal{K}$ the cases are mutually exclusive
  - **case-complete** iff for each function definition of an n-ary function g occurring in $\mathcal{K}$ and each well-sorted n-tuple of constructor ground terms given as input to g there is at least one of the cases which is satisfied
  - **terminating** iff there is no infinite rewriting sequence for any well-sorted ground term
  - **admissible** iff it is well-formed, well-sorted, deterministic, case-complete and terminating
- The rewrite relation wrt an admissible program is confluent $\leadsto$ **Exercise**

Evaluation

- Admissible programs $\mathcal{K}$ define a unique evaluator $eval_{\mathcal{K}}$ which maps terms to their normal form
- $eval_{\mathcal{K}} : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{C})$
- $eval_{\mathcal{K}}(t)$ is called value of t
- $eval_{\mathcal{K}}$ is an interpretation with domain $\mathcal{T}(\mathcal{C})$
- $eval_{\mathcal{K}}$ is called standard interpretation of $\mathcal{K}$
- Example Consider $\mathcal{K}_{\text{number}} \cup \mathcal{K}_{plus}$

$$
\begin{aligned}
& plus(s(0), s(0)) \\
\to\ & s(plus(p(s(0)), s(0))) \\
\to\ & s(plus(0, s(0))) \\
\to\ & s(s(0))
\end{aligned}
$$

Evaluation – Example

- Consider $\mathcal{K} = \mathcal{K}_{\text{number}} \cup \mathcal{K}_{plus}$
  - $eval_{\mathcal{K}} \models \mathcal{K}$
    $eval_{\mathcal{K}} \models (\forall X, Y : number)\ plus(X, Y) \approx plus(Y, X)$,
    $eval_{\mathcal{K}} \models (\forall X : number)\ X \not\approx s(X)$
    - $\rightsquigarrow$ Exercise
  - $\mathcal{K} \not\models (\forall X, Y : number)\ plus(X, Y) \approx plus(Y, X)$
    $\mathcal{K} \not\models (\forall X : number)\ X \not\approx s(X)$
    - $\rightsquigarrow$ Exercise

Theory of Admissible Programs

- Let $\mathcal{K}$ be an admissible program
- We consider $\{G \mid eval_{\mathcal{K}} \models G\}$
- In other words, we restrict us to one specific interpretation
  This interpretation is sometimes called standard or intended interpretation
- Idea   Add formulas to $\mathcal{K}$ such that non-standard interpretations are no longer models of $\mathcal{K}$
  - These formulas are called induction axioms
  - Let $\mathcal{K}_I$ be a decidable set of induction axioms such that $eval_{\mathcal{K}} \models \mathcal{K}_I$

Induction – Example

- Let $\mathcal{K} = \mathcal{K}_{\text{number}} \cup \mathcal{K}_{plus}$
- Let $\mathcal{K}_I$ be the set of all formulas of the form

$$(P(0) \wedge (\forall X : number)\,(P(X) \rightarrow P(s(X)))) \rightarrow (\forall X : number)\,P(X)$$

- This scheme can be instantiated by, e.g., replacing $P(X)$ by $X \not\approx s(X)$

$$(0 \not\approx s(0) \wedge (\forall X : number)\,(X \not\approx s(X) \rightarrow s(X) \not\approx s(s(X)))) \rightarrow (\forall X : number)\,X \not\approx s(X) \quad (1)$$

- $eval_{\mathcal{K}} \models (1) \quad \leadsto$ **Exercise**
- $\mathcal{K} \cup \{(1)\} \models (\forall X : number)\,X \not\approx s(X) \quad \leadsto$ **Exercise**
  - The proof is finite
    (in contrast to a proof of $eval_{\mathcal{K}} \models (\forall X : number)\,X \not\approx s(X)$)

Inductive Theorem Proving

- **Theorem proving by induction is incomplete (Gödel's incompleteness theorem)**
- **Induction axioms may be computed from inductively defined data structures**
- **Heuristics may guide selection of**
  - **the induction variable**
  - **the induction schema and**
  - **the induction axiom**
- **Several theorem provers based on induction are available, e.g.,**
  - NQTHM
  - OYSTER-CLAM
  - INKA