Inductive Definitions and Fixed Points

Introduction

Constructs for defining types and functions
Isabelle/HOL provides two core constructs for conservative extensions:

1. Constant definitions
2. Type definitions

Based on the core construct, there are further constructs:

- Recursive function definitions (primrec, fun, function)
- Recursive datatype definitions (datatype)
- Co-/inductively defined sets (inductive_set, coinductive_set)
- Co-/inductively defined predicates (inductive, coinductive)

Motivation

Goals

- Learn about inductive definitions:  
  \(\leadsto\) important concept in computer science!  
  E.g., to define operational semantics.
- Learn the underlying fixed point theory:  
  \(\leadsto\) fundamental theory in computer science!
- Learn how to apply it to transition systems  
  \(\leadsto\) central modeling concept for operational behavior!
6. Inductive Definitions and Fixed Points

6.1 Inductively defined sets and predicates

---

### Introductory example

**Informally:**
- 0 is even
- If \( n \) is even, so is \( n + 2 \)
- These are the only even numbers

**In Isabelle/HOL:**

![Isabelle code]

---

**Format of inductive definitions**

![Format code]

---

**Embedding inductive definitions into HOL**

**Conservative theory extension**

From an inductive definition, Isabelle
- generates a *definition* using a fixed point operator and
- proves theorems about it that can be used as proof rules

The theory underlying fixed point definitions is explained in Subsect. 6.2.
6. Inductive Definitions and Fixed Points
6.1 Inductively defined sets and predicates

Generated rules

Rules
Generated rules include

- the introduction rules of the definition, e.g.,
  
  \[
  0 \in \text{even} \quad \text{(even.zero)} \\
  n \in \text{even} \Rightarrow n + 2 \in \text{even} \quad \text{(even.step)}
  \]

- an elimination rule for case analysis
- an induction rule

Example 1:

Lemma: \(4 \in \text{even}\)
Proof: \(0 \in \text{even} \Rightarrow 2 \in \text{even} \Rightarrow 4 \in \text{even}\)

Discussion:

- Simple: Use \text{even.zero} and apply \text{even.step} finitely many times.
- Works because there is no free variable

Example 2:

Lemma: \(m \in \text{even} \Rightarrow \exists k. 2 \ast k = m\)
Proof: Idea:
- For rules of the form \(a \in S\): Show that property holds for \(a\)
- For rules of the form \(\[ a_1 \in S; \ldots; a_n \in S; \ldots \] \Rightarrow a_0 \in S\): Show that assuming \(a_1 \in S; \ldots; a_n \in S; \ldots \) and property holds for terms \(a_1, \ldots, a_n\), it holds for term \(a_0\)

Applied to \text{even}, we have to show:
- \(\exists k. 2 \ast k = 0\): trivial
- Assuming \(n \in \text{even}\) and \(\exists k. 2 \ast k = n\), show \(\exists k. 2 \ast k = n + 2\):
  simple arithmetic

Rule induction for \text{even}

To prove \(n \in \text{even} \Rightarrow P n\) by rule induction, one has to show:
- \(P 0\)
- \(P n \Rightarrow P (n + 2)\)

Isabelle provides the rule \text{even.induct}:

\[
\[ n \in \text{even}; P 0; \bigwedge n. P n \Rightarrow P(n + 2) \] \Rightarrow P n
\]
6. Inductive Definitions and Fixed Points

6.1 Inductively defined sets and predicates

Rule induction vs. natural/structural induction

Remarks:
- Rule induction uses the induction steps of the inductive definition and not of the underlying datatype! It differs from natural/structural induction.
- In the context of partial recursive functions, a similar proof technique is often called computational or fixed point induction.

Rule induction in general

Let $S$ be an inductively defined set.
To prove $x \in S \implies P x$ by rule induction on $x \in S$, we must prove for every rule:
$$\left[ a_1 \in S; \ldots; a_n \in S \right] \implies a \in S$$
that $P$ is preserved:
$$\left[ P a_1; \ldots; P a_n \right] \implies P a$$

In Isabelle/HOL: `apply (induct rule: S.induct)`

Inductive predicates

Isabelle/HOL also supports the inductive definition of predicates:

Example:

```
inductive even:: "nat => bool" where
"even 0" |
"even n => even (n+2)"
```

Comparison:
- predicate: simpler syntax
- set: direct usage of set operation, like $\cup$, etc.

Inductive predicates can be of type $\alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n \Rightarrow \text{bool}$

Further aspects

- Rule inversion and inductive cases (see IHT 7.1.5)
- Mutual inductive definitions (see IHT 7.1.6)
- Parameters in inductive definitions (see IHT 7.2)
Motivation

Introduction:
Inductive definitions can be considered as:
- Constant definition: define exactly one set (semantic interpretation)
- Axiom system: except all sets that satisfy the rules (axiomatic interpretation)
- Derivation system: show that an element is in a set by applying the rules (derivational interpretation)

Isabelle/HOL is based on the semantic interpretation. In addition, it allows to use the rules as part of the derivation system.

Remark
The interpretations have advantages and disadvantages/problems.

Illustrating the problems

Problem of semantic interpretation:
We have to assign a set to any well-formed inductive definition.

Example:
Which set should be assigned to $\text{fooset}$?

\begin{verbatim}
inductive_set fooset :: "nat set" where
  "n ∈ fooset =⇒ n+1 ∈ fooset"
\end{verbatim}

Problem of derivational interpretation
The rules of the definition are too weak. E.g., we cannot prove:

\[ 3 ∉ \text{even} \]

Problem of axiomatic interpretation:
There are usually many sets satisfying the rules of an inductive definition.

Example:
The following set $\text{even2}$ satisfies the rules of $\text{even}$:

\begin{verbatim}
definition even2 :: "nat set" where
  "even2 ≡ \{ n. n ≠ 1 \}"
\end{verbatim}

\begin{verbatim}
lemma "0 ∈ even2"
lemma "n ∈ even2 =⇒ n+2 ∈ even2"
\end{verbatim}
Semantics of inductive definition

**Definition**

Let \( f : T \Rightarrow T \) be a function. A value \( x \) is called a fixed point of \( f \) if \( x = f x \).

**Semantics approach for inductive definitions**

Three steps:

- Transform inductive definition \( ID \) into “normalized form”
- “Extract” a fixed point equation for a function \( F_{ID} :: \alpha \text{ set} \Rightarrow \alpha \text{ set} \)
- Take the least fixed point

**Assumption**

For every (well-formed) inductive definition, the least fixed point exists.

Transformation to “normalized form”

A “normalized” inductive definition has exactly one implication of the form:

\[
\text{inductive_set } S :: \alpha \text{ set} \text{ where } \\
\quad \text{"} m \in (F_{S} S) \Rightarrow m \in S \text{"}
\]

**Example:**

\[
\text{inductive_set } \text{even} :: \text{"nat set" where } \\
\quad \text{"} 0 \in \text{even} \mid \\
\quad \text{"} n \in \text{even} \Rightarrow n+2 \in \text{even} \text{"}
\]

has the normalized form:

\[
\text{inductive_set } \text{even} :: \text{"nat set" where } \\
\quad \text{"} m \in \{m. m=0 \lor (\exists n. n \in \text{even} \land m=n+2)\} \Rightarrow m \in \text{even} \text{"}
\]

That is, the function \( F_{\text{even}} \) is

\[
F_{\text{even}} \text{ nset} = \{m. m=0 \lor (\exists n. n \in \text{nset} \land m=n+2)\}
\]

Supremum and infimum

**Definition (Supremum/infimum)**

Let \((L, \leq)\) be partially ordered set and \(A \subseteq L\).

1. **Supremum:** \( y \in L \) is called a supremum of \( A \) if \( y \) is an upper bound of \( A \), i.e., \( b \leq y \) for all \( b \in A \) and 
   \( \forall y' \in L : ((y' \text{ upper bound of } A) \rightarrow y \leq y') \)
   
2. **Infimum:** analogously defined, greatest lower bound

Existence of fixed points:

Unique least and greatest fixed points exist if

1. \( F_{S} \) is monotone, i.e., \( F_{S} S \subseteq S \) for all \( S \).
2. Domain (and range) of \( F_{S} \) is a complete lattice (Knaster-Tarski theorem)

Prerequisites are satisfied for inductive definitions, because

1. In inductive definitions, occurrence of \( x \in S \) must be positive, and this allows to prove monotonicity.
2. Set of sets are a complete lattice with \( \subseteq \) as ordering.
**Existence and structure of fixed points**

**Theorem (Knaster-Tarski)**

Let \((L, \leq)\) be a complete lattice and let \(F : L \to L\) be a monotone function. Then the set of fixed points of \(F\) in \(L\) is also a complete lattice.

**Corollary (Knaster-Tarski)**

\(F\) has a (unique) least and greatest fixed point.

---

**Complete lattices**

**Definition (Complete lattice)**

A partially ordered set \((L, \leq)\) is a **complete lattice** if every subset \(A\) of \(L\) has both an infimum (also called the meet) and a supremum (also called the join) in \(L\).

The meet is denoted by \(\land A\), the join by \(\lor A\).

**Lemma**

Complete lattices are non empty.

**Lemma**

Let \(P(S)\) be the power set of a set \(S\). \((P(S), \subseteq)\) is a complete lattice.

---

**Proof of Knaster-Tarski Corollary**

We prove:

The set of all fixed points \(P\) of \(F\), \(P \subseteq L\), has the following properties:

1. \(\lor P = \lor \{ y \in L \mid y \leq F(y) \}\)
2. \((\lor P) \in P\)
3. \(\land P = \land \{ y \in L \mid F(y) \leq y \}\)
4. \((\land P) \in P\)

That is, \((\lor P)\) is the greatest and \((\land P)\) is the least fixed point.

**Proof:**

We show the first two properties. The proof of the third and forth property are analogous.

---

**Proof of Knaster-Tarski Corollary (2)**

**Show:** \(\lor P = \lor \{ y \in L \mid y \leq F(y) \}\) and \((\lor P) \in P\)

Let \(D = \{ y \in L \mid y \leq F(y) \}\) and \(u = \lor D\). We show:

\(u \in P\) and \(u = \lor P\), i.e., \(u\) is the greatest fixed point of \(F\).

For all \(x \in D\), also \(F(x) \in D\), because \(F\) is monotone and \(F(x) \leq F(F(x))\).

\(F(u)\) is an upper bound of \(D\), because for \(x \in D\), \(x \leq u\) and \(F(x) \leq F(u)\), i.e., \(x \leq F(x) \leq F(u)\).

As \(u\) is least upper bound, \(u \leq F(u)\). Thus, \(u \in D\).

As shown above, \(u \in D\) implies \(F(u) \in D\), thus \(F(u) \leq u\).

In summary, \(F(u) = u\), i.e., \(u\) is a fixed point, \(u \in P\).

Because \(P \subseteq D\), \(\lor P \leq \lor D\), hence \(u \leq \lor P \leq u\), i.e., \(u = \lor P\).
Lattices in Isabelle/HOL

Remark

Isabelle/HOL handles:

- lattices in Chapter 5 of theory Main
- complete lattices in Chapter 8 of theory Main
- inductive definitions and Knaster-Tarski in Chapter 9

The natural numbers are introduced in Chapter 15, using an inductive definition!

Some related definitions and lemmas in Isabelle/HOL

\[ \text{mono } f \equiv \forall A B. A \leq B \rightarrow f A \leq f B \]  \hspace{1cm} \text{(mono\_def)}

where \( A, B \) are often sets and “\( \leq \)” is “\( \subseteq \)”

\[ \text{lfp } f \equiv \inf \{ u | f u \leq u \} \]  \hspace{1cm} \text{(lfp\_def)}

\[ \text{mono } f \Rightarrow \text{lfp } f = f (\text{lfp } f) \]  \hspace{1cm} \text{(lfp\_unfold)}

\[ \llbracket \text{mono } f; f (\inf (\text{lfp } f) P) \leq P \rrbracket \Rightarrow \text{lfp } f \leq P \]  \hspace{1cm} \text{(lfp\_induct)}

\[ \text{gfp } f \equiv \sup \{ u | u \leq f u \} \]  \hspace{1cm} \text{(gfp\_def)}

\[ \text{mono } f \Rightarrow \text{gfp } f = f (\text{gfp } f) \]  \hspace{1cm} \text{(gfp\_unfold)}

\[ \llbracket \text{mono } f; X \leq f (\sup X (\text{gfp } f)) \rrbracket \Rightarrow X \leq \text{gfp } f \]  \hspace{1cm} \text{(coinduct)}

Motivation

Modeling

Behavior of software-controlled systems can be modeled

- by using a modeling language (UML, B, Z, ASM, ABS, Maude, ...)
- by formalizing the operational behavior as transition system

Transition systems

Transition systems are also a fundamental means for specifying

- the operational semantics of programming and modeling language (cf. Chap. 7)
- process calculi and concurrency
- computing architectures and hardware

Verification of transition systems cannot exploit program structure, but need other techniques.
6. Inductive Definitions and Fixed Points 6.3 Specifying and verifying transition systems

**Transition systems**

**Definition (Transition system)**

A **transition system** (TS) is a pair \((Q, T)\) consisting of

- a set \(Q\) of states;
- a binary relation \(T \subseteq Q \times Q\), usually called the **transition relation**. Notation: \(q \rightarrow q'\)

(Other names: state transition system, unlabeled transition system)

**Definition (Labeled transition system)**

A **labeled transition system** (LTS) over \(Act\) is a pair \((Q, T)\) consisting of

- a set \(Q\) of states;
- a ternary relation \(T \subseteq Q \times Act \times Q\), usually called the transition relation. Notation: \(q \xrightarrow{lab} q', \ lab \in Act\)

\(Act\) is called the set of **actions** or **labels**.

**Remark**

- The action labels express input, output, or an “explanation” of an internal state change.
- Finite automata are LTS.
- Often, transition systems are equipped with a set of initial states or sets of initial and final states.
- **Traces** are sequences \(\langle q_i \rangle\) of states with \((q_i, q_{i+1}) \in T\) or sequences of labels.
- **Behaviors** are sets of traces (beginning at initial states).
- **Properties** are often expressed in appropriate logics (PDL, CTL ...)

**Lemma**

Every LTS \((Q, T)\) over \(Act\) can be expressed by a TS \((Q', T')\) such that there is a mapping

\[ rep : Q \times Act \rightarrow Q' \]

with

\[ q_1 \xrightarrow{lab} q_2 \in T \iff \exists \ lab. \ rep(q_1, lab) \rightarrow rep(q_2, lab) \in T' \]

(Proof is a left as an exercise)

**Modeling: Case study Elevator control system**

**Requirements**

Design the control for an elevator serving 3 floors such that:

- **Model:**
  - Elevator has for each floor one button which, if pressed, causes it to visit that floor. Button is cancelled when the elevator visits the floor.
  - Each floor has a button to request the elevator. Button is cancelled when elevator visits the floor.
  - The elevator remains in the middle floor if no requests are pending.

- **Properties:**
  - All requests for floors from the elevator must be serviced eventually.
  - All requests from floors must be serviced eventually.
Modeling approach and motivation

- Direct modeling as a transition system:
  - without using a programming or modeling language
  - without using a library/theory

Motivation:
- Learn to construct models
- Deepen the knowledge about transition systems
- Understand the formalization of transition systems

Datatypes for facts and actions

datatype floor = F0 | F1 | F2 (* three floors *)
datatype action = Call floor (* input message *)
| GoTo floor (* input message *)
| Open (* output message *)
| Move (* internal message *)
datatype direction = UP | DW (* up | down *)
datatype door = CL | OP (* closed | open *)
type_synonym state =
  action × floor × direction × door × (floor set)
  (* what , where , where to , door state , requests *)

Datatypes and actions: Transition relation

\[
\begin{align*}
\text{inductive_set } \text{tr} : & \text{ (state } \times \text{ state) set where } \\
& \begin{cases}
  \text{g } \not\in \text{T; } \neg (\text{f } = \text{g } \land \text{d } = \text{OP}) \Rightarrow \\
  (\text{a,f,r,d,T}, (\text{Call g,f,r,d,TU}\{\text{g}\})) \in \text{tr} | \\
  \text{g } \not\in \text{T; } \neg (\text{f } = \text{g } \land \text{d } = \text{OP}) \Rightarrow \\
  (\text{a,f,r,d,T}, (\text{GoTo g,f,r,d,TU}\{\text{g}\})) \in \text{tr} | \\
  \text{f } \in \text{T} \Rightarrow (\text{a,f,r,d,T}, (\text{Open f,r,OP,T-}\{\text{f}\})) \in \text{tr} | \\
  (\text{a,F1,r,d,}\{\text{F0}\}), (\text{Move,F0,DW,CL,}\{\text{F0}\}) \in \text{tr} | \\
  (\text{a,F1,r,}\{\text{F2}\}), (\text{Move,F2,UP,CL,}\{\text{F2}\}) \in \text{tr} | \\
  \text{F0}\not\in \text{T} \Rightarrow (\text{a,F0,r,d,T}, (\text{Move,F1,UP,CL,T}) \in \text{tr} | \\
  \text{F2}\not\in \text{T} \Rightarrow (\text{a,F2,r,d,T}, (\text{Move,F1,DW,CL,T}) \in \text{tr} | \\
  (\text{F1}\not\in \text{T}; \text{F2}\not\in \text{T}) \Rightarrow \\
  (\text{a,F1,UP,d,T}, (\text{Move,F2,UP,CL,T}) \in \text{tr} | \\
  (\text{F1}\not\in \text{T}; \text{F0}\not\in \text{T}) \Rightarrow \\
  (\text{a,F1,DW,d,T}, (\text{Move,F0,DW,CL,T}) \in \text{tr}
\end{cases}
\end{align*}
\]

Traces

Defining sets of infinite traces

types trace = "nat ⇒ state"

coinductive_set traces :: "trace set" where
"[ t \in \text{traces}; (s, t 0) \in \text{tr} ] \Rightarrow \\
(\text{\lambda n. case n of 0 } \Rightarrow s | \text{ Suc x } \Rightarrow t x) \in \text{traces}"

(* Functions on traces *)
definition head :: "trace ⇒ state" where
"head t \equiv t 0"
definition drp :: "trace ⇒ nat ⇒ trace" where
"drp t n \equiv (\lambda m. t (n + m))"
Basic properties of traces

- **Lemma (iff):** \( \text{drp (drp } t \ n) \ m = \text{drp } t \ (n + m) \)
- **Lemma drp_traces:** \( t \in \text{traces} \implies \text{drp } t \ n \in \text{traces} \)

More interesting properties

Expressing temporal properties of traces

- For every floor \( f \): If \( f \) is a requested floor, the elevator will eventually reach the floor and open the door in \( f \):
  \[
  \text{Always } (\ll To f \gg \implies \text{Finally } (\ll Op \gg \text{ and } \ll At f \gg))
  \]
  Could be directly expressed over traces
- Alternative: Temporal logic, e.g., linear TL:
  - Formulas built with \( \text{Atoms, } \neg, \land, \Diamond \)
  - Interpretations: Kripke structures \( (Q, I, T, L) \)
  - A transition relation \( T \subseteq Q \times Q \) such that \( \forall q \in Q. \exists q' \in Q. (q, q') \in T \)
  - A labeling (or interpretation) function \( L :: Q \Rightarrow \mathcal{P}(\text{Atoms}) \)

Syntax for LTL

**LTL formulas:**

\[
\text{datatype formula} = \begin{cases} 
\text{Atom atom} & (\ll \_ \gg) \\
\text{Neg formula} & (\neg) \\
\text{And formula formula} & (\text{infixr } \land \land 80) \\
\text{Always formula} & (\Box) \\
\text{Finally formula} & (\Diamond)
\end{cases}
\]

As abbreviation:

**Definition Imp :: formula \Rightarrow formula \Rightarrow formula**

\[
\text{where } \begin{cases} 
a .\rightarrow b = \neg (a \land \neg b) \end{cases}
\]

Semantics for LTL

**Definition (Kripke structure)**

Let \( AP \) be a set of atomic propositions. A Kripke structure is a 4-tuple \( M = (Q, I, T, L) \) consisting of

- a finite set of states \( Q \)
- a set of initial states \( I \subseteq Q \)
- a relation \( T \subseteq Q \times Q \) such that \( \forall q \in Q. \exists q' \in Q. (q, q') \in T \)
- a labeling (or interpretation) function \( L :: Q \Rightarrow \mathcal{P}(\text{Atoms}) \)
Kripke structure of elevator example

- $Q$ as defined by type synonym "state" (UNIV state)
- $I$: some suitable set of initial states
- $T$ as defined by $tr$ (why is there always a successor state?), and
- define $AP \equiv$ atom and $L$ as follows:

  ```
  datatype atom = Up | Op | At floor | To floor
  fun L :: "state ⇒ atom set" where
  "L (_, g, dr, ds, fs) = \{ a . (dr=UP ∧ a=Up) ∨ (ds=OP ∧ a=Op) ∨ (a=At g) ∨ (∃ f∈fs.(a=To f)) \}"
  ```

Remarks and example

Remarks:
- Since $T$ is left-total, it is always possible to construct an infinite path through the Kripke structure. A deadlock state $qd$ can be expressed by a single outgoing edge back to $qd$ itself.
- The labeling function $L$ defines for each state $q$ in $Q$ the set $L(s)$ of all atomic propositions that are valid in $s$.
- Kripke structures are used to define the semantics of LTL (see next slide).

Example of formalized property:

```
definition liveness :: "floor ⇒ formula" where
"liveness f = □ (≪ To f ≫ . −→ ⋄ (≪ Op ≫ . ∧ ≪ At f ≫ ))"
```

Semantics for LTL

Let $M = (Q, I, T, L)$ be a Kripke structure and `trace` the type of traces defined by $T$:

```
primrec valid_in_trace :: "trace ⇒ formula ⇒ bool" ("(_ ⊨ _)") [80, 80] 80 where
"t ⊨ ≪ a≫ = ( a ∈ L (head t) )"
| "t ⊨ .¬f = ( ¬ (t ⊨ f) )"
| "t ⊨ f ∧ g = ( (t ⊨ f) ∧ (t ⊨ g) )"
| "t ⊨ □ f = ( ∀ n. ((drp t n) ⊨ f ) )"
| "t ⊨ ◇ f = ( ∃ n. ((drp t n) ⊨ f ) )"

definition valid :: "formula ⇒ bool" ("(≡ _)") [80] 80 where
"≡ f ≡ ( ∀ t ∈ traces. t ⊨ f )"
```

Reasoning about finite transition systems

Three options for reasoning:
1. In Isabelle/HOL using the rules obtained from the definitions (semantics-based, formalized mathematical reasoning):
   - Elevator.thy (see exercises)
2. In LTL using rules for temporal reasoning (rules not shown here)
3. Model checking (works for finite state systems)