## The axioms and rules of HOL (2)

Additionally, there is:

- universal $\alpha, \beta$, and $\eta$ congruence on terms (implicitly),
- the axiom of infinity, and
- the axiom of choice (Hilbert operator).
- This is the entire basis!





## Properties of HOL

### Theorem 1 (Soundness of HOL)

HOL is sound:

$\vdash \phi$ implies $\phi$ is valid in the general/standard sense

### Theorem 2 (Incompleteness of HOL)

HOL is incomplete w.r.t. standard models:

There exist $\phi$ that are valid in the standard sense, but $\nvdash \phi$

### Remark

[And86, Chap. 5-7] presents proofs for these theorems. Note, however, that [And86] does not restrict the semantics to models where $\mathcal{D}_{ind}$ is infinite.





## Section 3.3

**Conservative Extension of Theories**





## Basic ideas

- Theories are stepwise extension of the core theory of HOL
- Extensions may introduce new constants and new types
- Inconsistencies are avoided by construction
- Syntactical mechanisms are used to make extensions more convenient

### Remark

Extensions only introduce names for “things” that already exist in the core theory.

## Basic definitions

Terminology and basic definitions (cf. [GordonMelham93]):

### Definition (Theory)

A (syntactic) *theory* T is a triple $(\chi, \Sigma, A)$ where

- $\chi$ is a signature for type names
- $\Sigma$ is a signature for function/constant names using types of $\chi$
- $A$ is a set of axioms over $\Sigma$

### Definition (Consistent)

A theory T is *consistent* iff *False* is not provable in T: $A \nvdash \mathit{False}$

### Definition (Theory extension)

A theory $T' = (\chi', \Sigma', A')$ is an extension of a theory $T = (\chi, \Sigma, A)$ iff $\chi \subseteq \chi'$ and $\Sigma \subseteq \Sigma'$ and $A \subseteq A'$.





## Basic definitions (cont.)

### Definition (Conservative extension)

Let $T = (\chi, \Sigma, A)$ and $Th(T) = \{\phi \mid A \vdash \phi\}$;
a theory extension $T' = (\chi', \Sigma', A')$ of $T$ is *conservative* iff

$$Th(T) = (Th(T') \mid_\Sigma)$$

where $\mid_\Sigma$ restricts sets formulas to those containing only names in $\Sigma$.

### Lemma (Consistency)

*If $T'$ is a conservative extension of a consistent theory T, then*

$$\mathit{False} \notin Th(T')$$





## Syntactic schemata for conservative extensions

Not every extension is conservative:

### Counterexample

Let $T = (\chi, \Sigma, A)$ with $nat \in \chi$.
$T' = (\chi, \Sigma, A \cup \{\forall f_{nat \Rightarrow nat}.x = f\ x\})$ is not a conservative extension of $T$.

We consider conservative extensions by:

- constant definitions
- type definitions

### Remark

Cf. [GordonMelham93] for other extension schemata





## Constant definitions

### Definition (Constant definition)

A theory extension $T' = (\chi', \Sigma', A')$ of $T = (\chi, \Sigma, A)$ is called a constant definition iff

- $\chi' = \chi$ and $\Sigma' = \Sigma \cup \{c :: \alpha\}$ with $\alpha \in \chi$ and $c \notin \Sigma$
- $A' = A \cup \{c = E\}$
- $E$ does not contain $c$ (no recursion)
- $E$ is closed (no free variables)
- (no subterm of $E$ has a type containing a type variable that is not contained in the type of c)

# Why side conditions?

- no recursion and closedness guarantee well-definedness
- Consider the following definition with a free type variable:

$$c = (\exists\, x :: 'a.\ \exists\, y :: 'a.\ x \neq y)$$

If the language allows to instantiate the type variables:

$$\begin{array}{ll} & c = c \qquad\qquad \text{(by refl)} \\ \Longrightarrow & (\exists\, x :: \mathit{bool}.\ \exists\, y :: \mathit{bool}.\ x \neq y) = (\exists\, x :: \mathit{Unit}.\ \exists\, y :: \mathit{Unit}.\ x \neq y) \\ \Longrightarrow & \mathit{True} = \mathit{False} \\ \Longrightarrow & \mathit{False} \end{array}$$





# Constant definitions are conservative

**Lemma (Constant definition)**

*A constant definition is a conservative extension.*

**Proof.**

Proof sketch:

- $Th(T) \subseteq (Th(T')\,|_\Sigma)$ : from definition of $Th$
- $(Th(T')\,|_\Sigma) \subseteq Th(T)$ : let $\pi'$ be a proof for $\phi \in (Th(T')\,|_\Sigma)$. We unfold any subterm in $\pi'$ that contains $c$ by $c = E$ into $\pi$. $\pi$ is a proof in $T$, i.e., $\phi \in Th(T)$.

□





# Constant definitions in Isabelle/HOL

Definitions of *True*, *False*, *All*, *Ex*, ¬, ∧, ∨, *if*, *let*:

| | | | |
|---|---|---|---|
| True | :: | bool | |
| False | :: | bool | |
| Not | :: | bool ⇒ bool | ("‘¬_"' [40] 40) |
| If | :: | [bool, 'a, 'a ] ⇒ 'a | ("‘if _ then _ else _)"') |
| Let | :: | ['a, 'a ⇒ 'b] ⇒ 'b | |
| The | :: | ('a ⇒ bool) ⇒ 'a | (binder "‘THE"' 10) |
| All | :: | ('a ⇒ bool) ⇒ bool | (binder "‘∀"' 10) |
| Ex | :: | ('a ⇒ bool) ⇒ bool | (binder "‘∃"' 10) |
| = | :: | ['a,'a] ⇒ bool | (infixl 50) |
| ∧ | :: | [bool, bool] ⇒ bool | (infixr 35) |
| ∨ | :: | [bool, bool] ⇒ bool | (infixr 30) |
| ⟶ | :: | [bool, bool] ⇒ bool | (infixr 25) |





# Constant definitions in Isabelle/HOL (2)

| | | | |
|---|---|---|---|
| True_def: | True | ≡ | $((\lambda x :: \text{bool}.x) = (\lambda x.x))$ |
| All_def: | All(P) | ≡ | $(P = (\lambda x.\ \text{True}))$ |
| Ex_def: | Ex(P) | ≡ | $\forall Q.(\forall x.Px \longrightarrow Q) \longrightarrow Q$ |
| False_def: | False | ≡ | $(\forall P.P)$ |
| not_def: | $\neg P$ | ≡ | $P \longrightarrow \text{False}$ |
| and_def: | $P \wedge Q$ | ≡ | $\forall R.(P \longrightarrow Q \longrightarrow R) \longrightarrow R$ |
| or_def: | $P \vee Q$ | ≡ | $\forall R.(P \longrightarrow R) \longrightarrow (Q \longrightarrow R) \longrightarrow R$ |
| if_def: | If $P \times y$ | ≡ | $\text{THE } z :: 'a.(P = \text{True} \longrightarrow z = x) \wedge (P = \text{False} \longrightarrow z = y)$ |
| Let_def: | Let $s$ $f$ | ≡ | $f(s)$ |

# Approaching type definitions

## Idea

- Specify a subset of the elements of an existing type $r$
- "Copy" the subset and use the copy as value set of the new type $t$
- Link old and new type by two functions

More precisely, a type definition is based on:

- an existing type $r$
- a predicate $S :: r \Rightarrow bool$, defining a non-empty "subset" of $r$;
- an abstraction function $Abs_t :: r \Rightarrow t$
- a representation function $Rep_t :: t \Rightarrow r$
- axioms stating an isomorphism between S and the new type t.





# Type definitions as theory extensions

## Definition (Type definition)

Let $T = (\chi, \Sigma, A)$ be a theory and $r \in \chi$ and $S$ a term of type $r \Rightarrow bool$.
A theory extension $T' = (\chi', \Sigma', A')$ of $T$ is a type definition for $t$ with $t \notin \chi$ iff

- $\chi' = \chi \cup \{t\}$
- $\Sigma' = \Sigma \cup \{\ Abs_t :: r \Rightarrow t,\ Rep_t :: t \Rightarrow r\ \}$
- $A' = A \cup \{\ \forall x.\ Abs_t(Rep_t\ x) = x,\ \ \forall y.\ S\ y \longrightarrow Rep_t(Abs_t\ y) = y\ \}$
- One has to prove $T \vdash\ \exists x.\ S\ x$ (using Isabelle/HOL)

## Lemma (Type definition)

*A type definition is a conservative extension.*

For a proof see [GordonMelham93]





# The nature of extensions

## Remark

This may seem strange: if a new type is always isomorphic to a subset of an existing type, how is this construction going to lead to a "rich" collection of types for large-scale applications?

- But in fact, due to *ind* and $\Rightarrow$, the types in HOL are already very rich.
- Thus, extensions essentially give names to values and types that have already been "expressible" in the "old" theory.
- Extensions allow to formulate theorems in a more compact and readable way.

We now give three examples revealing the power of type definitions:

- Typed sets
- Pairs





# Types for sets

We define the new type *natset* containing all sets of natural numbers:

- existing type: $(nat \Rightarrow bool)$
- predicate $S :: (nat \Rightarrow bool) \Rightarrow bool, S \equiv\ \lambda f.True$
- $\chi' = \chi \cup \{natset\}$
- $\Sigma' = \Sigma \cup \{\ Abs_{natset} :: (nat \Rightarrow bool) \Rightarrow natset,$
  $Rep_{natset} :: natset \Rightarrow (nat \Rightarrow bool)\ \}$
- $A' = A \cup \{\ \forall x.\ Abs_{natset}(Rep_{natset}\ x) = x,$
  $\forall y.\ True \longrightarrow Rep_{natset}(Abs_{natset}\ y) = y\ \}$
- One has to prove $T \vdash\ \exists x.(\lambda f.True)\ x$ (using Isabelle/HOL)

# Remarks on the set type

## Remarks

- Isabelle/HOL allows to define a parametric type $\alpha$ *set* where $\alpha$ is a type variable.
- Functions of type $\alpha \Rightarrow$ *bool* are used to represent sets, i.e., sets are represented by their characteristic function.
- In $(Abs_{\alpha\ set}\ f)$, the abstraction function $Abs_{\alpha\ set}$ can thus be read as "interpret $f$ as a set".
- Here, sets are just an example to demonstrate type definitions. Later we study them for their own sake.





# Approaching the types for pairs

Given some types $\alpha$ and $\beta$.
How can we represent pairs, i.e., define the type $\alpha \times \beta$?

## Idea:

- Existing type: $\alpha \Rightarrow \beta \Rightarrow$ *bool*
- Represent pairs as functions of type $\alpha \Rightarrow \beta \Rightarrow$ *bool*)
- Use function $\lambda x :: \alpha.\ \lambda y :: \beta.\ x = a \wedge y = b$ to represent the pair $(a, b)$
- It is clear that there is exactly one function for each pair.
- There are also functions of type $\alpha \Rightarrow \beta \Rightarrow$ *bool* that do not represent a pair, i.e., we have to define a nontrivial $S$.





# Types for pairs

We define the new type $\alpha \times \beta$:

- existing type: $\alpha \Rightarrow \beta \Rightarrow$ *bool*
- predicate $S \equiv \lambda f :: \alpha \Rightarrow \beta \Rightarrow$ *bool*.
  $$\exists a.\ \exists b.\ f = \lambda x :: \alpha.\ \lambda y :: \beta.\ x = a \wedge y = b$$
- $\chi' = \chi \cup \{\alpha \times \beta\}$

## Remark

Isabelle/HOL provides a special syntax for type definitions.





# Type definitions in Isabelle/HOL

## Syntax for type definitions

```
typedef (T) (typevars) T' = "{x. A(x)}"
```

Relation with explained schema:

- The new type is $T'$
- $r$ is the type of $x$ (inferred)
- $S$ is $\lambda x.\ A\ x$
- Constants $Abs_{T'}$ and $Rep_{T'}$ are automatically generated.

## Conservative extensions: Summary

- We have presented a method to safely build up larger theories:
  - Constant definitions
  - Type definitions
- Subtle side conditions
- New types must be isomorphic to a “subset” of an existing type.
- Isabelle/HOL uses these conservative extensions to
  - build up the theory `Main` from the core definitions of HOL
  - provide more convenient specialized syntax for conservative extensions (datatype, primrec, function, ...)





## Conclusions of Chap. 3

- HOL generalizes semantics of FOL
  - *bool* serves as type of propositions
  - Syntax/semantics allows for higher-order functions
- Logic is rather minimal: 8 rules, more-or-less obvious
- Logic is very powerful in terms of what we can represent/derive.
  - Other “logical” syntax
  - Rich theories via conservative extensions





## Questions

1. What is the foundational reason that HOL is typed? Are there other reasons w.r.t. an application in computer science?
2. What does “higher-order” mean?
3. Why is predicate logic not sufficient? Give an example?
4. What are the types in HOL?
5. What are the terms in HOL? Give examples of constants.
6. Explain the description operator.
7. What is a frame? What is an interpretation?
8. How is satisfiability defined?





## Questions (2)

9. What is a standard model?
10. Give and explain one of the axioms of HOL?
11. Can the constants True and False be defined in HOL?
12. What does it mean that HOL+infinity is incomplete wrt. standard models?
13. What is a conservative extension?
14. What is the advantage of conservative extensions over axiomatic definitions?
15. Which syntactic schemata for conservative extensions were treated in the lecture?
16. Give examples of constant definitions.
17. Explain the definitions of new types?
18. Does a data type definition in Isabelle/HOL lead to a new type?