



Test de Systèmes Informatiques

Mise à niveau – Partie II

Pr. Burkhardt Wolff

What does Test do in Tests in Software Engineering?

- *« We know less about the theory of testing, which we do often, than about the theory of program proving, which we do seldom »*

Goodenough J. B.,
Gerhart S.,
IEEE Transactions on
Software Engineering, 1975





Testing : Conclusions Partie - I

- The core problem of Testing is the **automated generation of test-data**
- Test-Data Generation must be based:
 - on **specifications** (a model what a program should do)
 - on **programs** (a model on how a program does behave)
 - on **symbolic computations** of both



Testing : Conclusions Partie - I

- For all three, we need a foundation.
We chose Higher-Order Logic (HOL)
 - Foundation λ -Calculus
 - Foundation λ^α -Calculus
- Meta Logics
- HOL Core
- HOL Library



Testing : Conclusions Partie - I

- Brief Revision:
 - Foundation λ -Calculus
 - Foundation λ^α -Calculus
 - [Meta Logics]
 - [HOL Core]
 - [HOL Library]



Foundations: HOL / λ -Calculus

- Thus, the syntactic expressions E of the λ -calculus are for short:

$$E ::= C \mid V \mid \lambda x. E \mid E E$$

- Examples $C = \{_ + _, 0, 1, 2\}$, $V = \{x_1, x_2, \dots\}$:

$$\begin{aligned} & (_ + _) 1 2 \\ & (\lambda x_1. (_ + _) 2 x_1) \end{aligned}$$



Foundations: HOL / λ -Calculus

- Thus, the syntactic expressions E of the λ -calculus are for short:

$$E := C \mid V \mid \lambda x. E \mid E E$$

- Examples $C = \{_ + _, 0, 1, 2\}$, $V = \{x_1, x_2, \dots\}$:

$(_ + _) 1 2$

$(\lambda x_1. (_ + _) 2 x_1)$

we write: $1 + 2$

we write: $\lambda x_1. 2 + x_1$



Foundations: HOL / λ^α -Calculus

- Core of HOL: the λ^α -calculus, a simple typed language for functions and logics

- Type Expressions

$$\tau := \alpha \mid \Xi_0 \mid \Xi_1(\tau) \mid \dots \mid \Xi_k(\tau_1, \dots, \tau_k) \mid \dots$$

- Well-Typedness-Statement:

$$\Sigma, \Gamma \vdash E :: \tau$$



Foundations: HOL / λ^α -Calculus

- Core of HOL: the λ^α -calculus, a simple typed language for functions and logics
 - Constant Environment $\Sigma = C \rightarrow \tau$
 - Variable Environment $\Gamma = V \rightarrow \tau$
 - Well-Typedness-Statement:

$$\Sigma, \Gamma \vdash E :: \tau$$



Foundations: HOL / λ^α -Calculus

- Examples:

Are the following expressions well-typed:

$$\Sigma, ? \quad \vdash \{1\} :: ?$$

$$\Sigma, ? \quad \vdash \{1\} \in \{1\} :: ?$$

$$\Sigma, ? \quad \vdash \{1\} \subseteq \{1\} :: ?$$

$$\Sigma, ?$$

$$\vdash \{1\} \cup \{2\} \in \text{Pow } A :: ?$$

where we assume Σ to typed as in Part I!!

Foundations: HOL / λ^α -Calculus

■ Examples:

Are the following expressions well-typed:

$\Sigma, \emptyset \quad \vdash \{1\} :: \text{set int}$ 

$\Sigma, \emptyset \quad \vdash \{1\} \in \{1\} :: -$ 

$\Sigma, \emptyset \quad \vdash \{1\} \subseteq \{1\} :: \text{bool}$ 

$\Sigma, \{A \mapsto \text{set int}\}$
 $\vdash \{1\} \cup \{2\} \in \text{Pow } A :: \text{bool}$ 



Testing : Conclusions Partie - I

- TODO in the rest of this lecture:
 - Foundation λ -Calculus
 - Foundation λ^α -Calculus
 - Meta Logics
 - HOL Core
 - HOL Library



Foundations: HOL / Meta-Logics

- In order to represent RULES of a Logics, we need a little language providing syntax (and semantics) for this.
- This is a *Meta-Logic* (in Isabelle, it is called *Pure* ; strictly speaking is Isabelle just an implementation of Pure plus mechanisms to safely extend it.)



Foundations: HOL / Meta-Logics

- Our *Meta-Logic* has:
 - ... λ^α -calculus with function type $_ \Rightarrow _ \in \Xi_2$
 - The type-constructor *prop* (i.e. $\text{prop} \in \Xi_0$)
 - The meta-equality: $(_ \equiv _ \mapsto \alpha \Rightarrow \alpha \Rightarrow \text{prop}) \in \Sigma$
 - The meta-implication:
$$(_ \Longrightarrow _ \mapsto \text{prop} \Rightarrow \text{prop} \Rightarrow \text{prop}) \in \Sigma$$
 - The meta-quantifier (“this is a fresh free variable”) $(\Lambda _ . _ \mapsto (\alpha \Rightarrow \text{prop}) \Rightarrow \text{prop}) \in \Sigma$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Informal Mathematic Textbook Notation:

From assumption A_1 to A_n ,
we infer A_{n+1} .



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Formal Textbook Notation:

$$\frac{A_1 \quad \dots \quad A_n}{A_{n+1}}$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Formal Notation in our Meta-Logic:

$$A_1 \Longrightarrow \dots \Longrightarrow A_n \Longrightarrow A_{n+1}$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Alternative Notation in our Meta-Logic:

$$\llbracket A_1 ; \dots ; A_n \rrbracket \Longrightarrow A_{n+1}$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Informal Mathematic Textbook Notation:

If we can infer Q under the assumption P ,
we can infer R .



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Formal Mathematic
Textbook Notation:

$$\begin{array}{c} [P] \\ \vdots \\ Q \\ \hline R \end{array}$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Formal Notation in Meta-Logic:

$$(P \implies Q) \implies R$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Informal Mathematic Textbook Notation:

If we can infer $Q\ x$ under the assumption $P\ x$ (where x does not occur free in the assumptions), we can infer R .



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Informal Mathematic Textbook Notation:

What ???

Oh gee, natural deduction is not all that natural, after all (Larry Paulson).



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Informal Mathematic
Textbook Notation:

$$\begin{array}{c} [P]_a \\ \vdots \\ \vdots \\ Q \\ \hline R \end{array}$$



Foundations: HOL / Meta-Logics

- In our *Meta-Logic* we describe *rules* as follows:

Formal Notation in Meta-Logic:

$$(\Lambda a. P a \implies Q a) \implies R$$



Foundations: HOL / Meta-Logics

- Having cleared what *rules* are (in natural deduction), we would like to combine them to *proofs*. The basics are straight-forward:

Foundations: HOL / Meta-Logics

- Having cleared what *rules* are (in natural deduction), we would like to combine them to *proofs*. The basics are straight-forward:

$$\frac{\frac{\frac{A \quad B}{C} \quad D}{F} \quad \frac{C \quad D}{F}}{H}}{J}$$

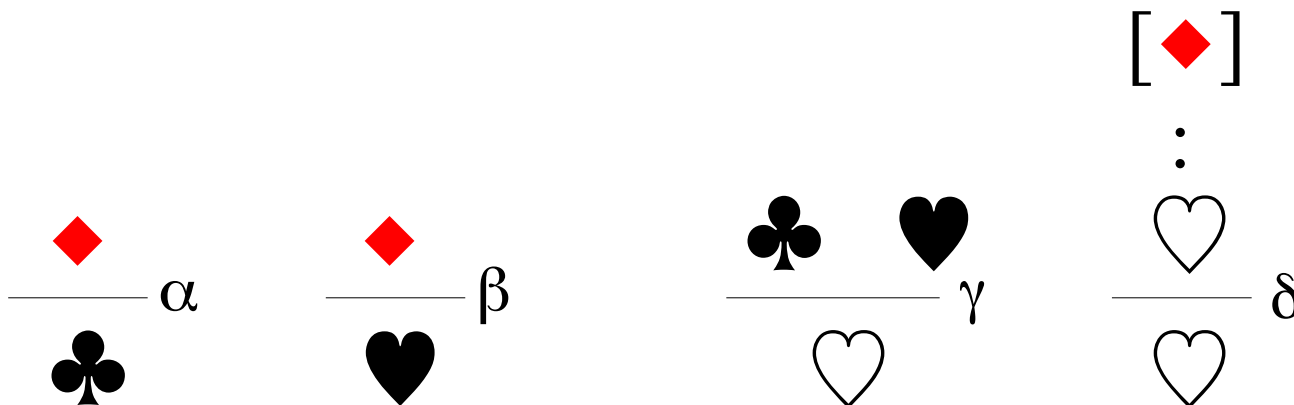


Foundations: HOL / Meta-Logics

- Having cleared what *rules* are (in natural deduction), we would like to combine them to *proofs*. The basics are straight-forward:
 - i.e conclusions have to match assumptions
 - the leaves of the tree are the “assumptions of the proof” Γ establishing the conclusion φ (also written : $\Gamma \vdash \varphi$)
 - in our example: $\{A,B,C,D\} \vdash J$

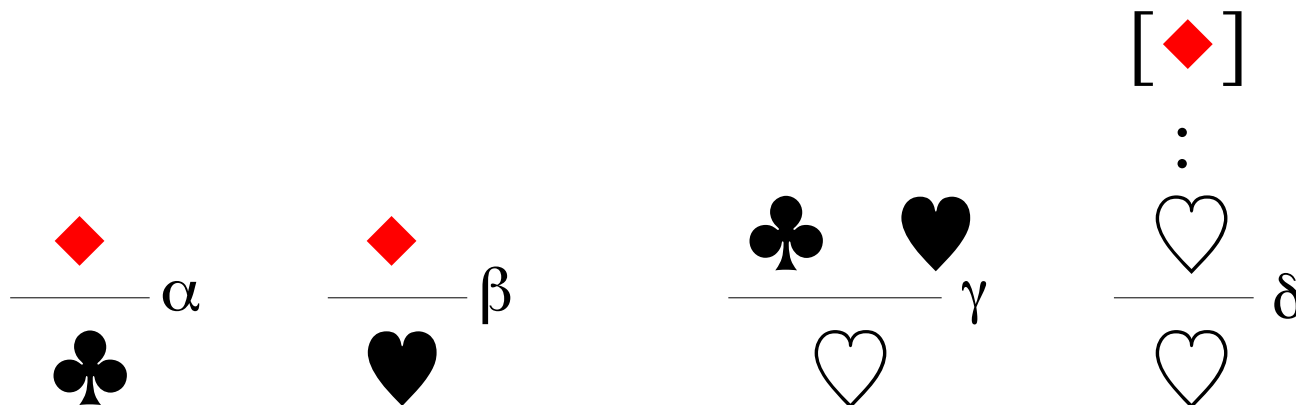
Foundations: HOL / Meta-Logics

- A bit tricky in *proofs*: Discharging assumptions. Consider the rule set $\{\alpha, \beta, \gamma, \delta\}$:



Foundations: HOL / Meta-Logics

- A bit tricky in proofs: *Discharging* assumptions. Consider the rule set $\{\alpha, \beta, \gamma, \delta\}$:

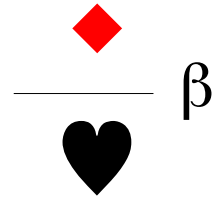
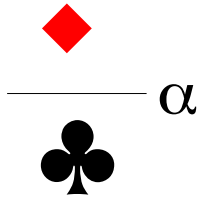


Can we prove  ?



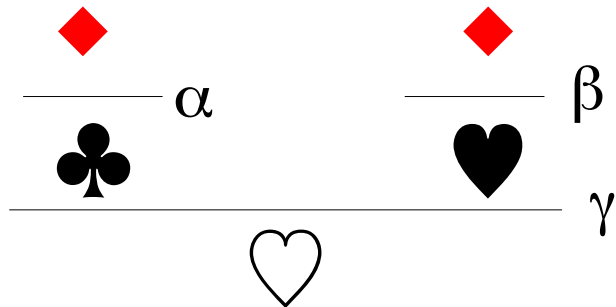
Foundations: HOL / Meta-Logics

- Yes !



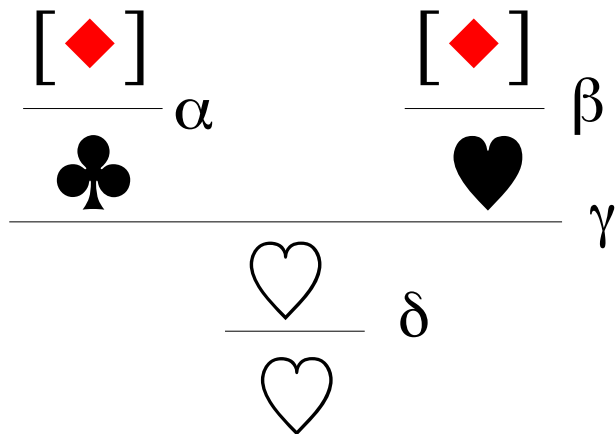
Foundations: HOL / Meta-Logics

- Yes !



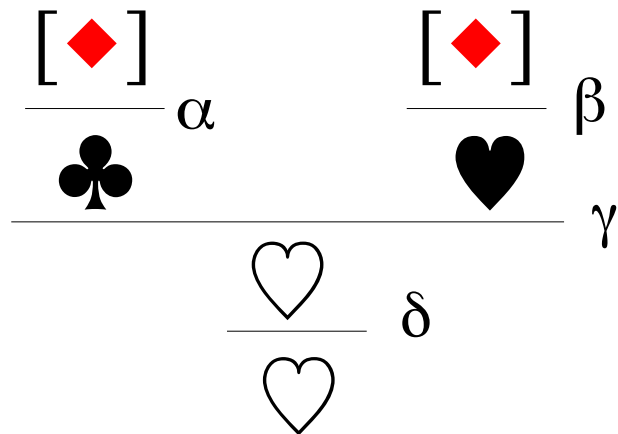
Foundations: HOL / Meta-Logics

- Yes !



Foundations: HOL / Meta-Logics

- Yes !





Foundations: HOL / Meta-Logics

- We ignore the foundational (= semantic) aspects of the Meta-Language here, the interested reader is referred to:

L. C. Paulson. The foundation of a generic theorem prover.
J. Automated Reasoning **5** (1989), 363–397.
<http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-130.html>

L. C. Paulson. Isabelle: the next 700 theorem provers.
In: P. Odifreddi (editor), Logic and Computer Science
(Academic Press, 1990), 361–386.
<http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-143.html>



Foundations: HOL / Core

- ... built inside our Meta-Language
- Defines the logic operators of HOL.

- Core-Type: $\text{bool} \in \mathbf{E}_0$

- Connectives:

$$\Sigma \supseteq \left\{ \begin{array}{l} \neg_ \mapsto \text{bool} \Rightarrow \text{bool}, \\ _ \wedge _ \mapsto \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}, \\ _ \vee _ \mapsto \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}, \\ _ \rightarrow _ \mapsto \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}, \\ _ = _ \mapsto \alpha \Rightarrow \alpha \Rightarrow \text{bool} \end{array} \right\}$$



Foundations: HOL / Core

- ... built inside our Meta-Language
- Defines the logic operators of HOL.
 - Core-Type: $\text{bool} \in \Xi_0$
 - Quantifiers:
$$\Sigma \supseteq \left\{ \begin{array}{l} \forall _ . _ \mapsto (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool}, \\ \exists _ . _ \mapsto (\alpha \Rightarrow \text{bool}) \Rightarrow \text{bool} \end{array} \right\}$$
 - Notation:
we write “ $\forall x. P$ ” instead of $(\forall _ . _)(\lambda x. P)$



Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$
$$\frac{A \quad B}{A \wedge B}$$
$$\frac{A \wedge B \quad \begin{array}{c} [A, B] \\ \vdots \\ Q \end{array}}{Q}$$



Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\frac{B}{A \vee B}$$

$$\frac{B}{A \vee B}$$

$$\frac{A \vee B \quad \begin{array}{c} [A] \\ \vdots \\ Q \end{array} \quad \begin{array}{c} [B] \\ \vdots \\ Q \end{array}}{Q}$$

Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\frac{A \quad \neg A}{Q}$$

$$\frac{\neg\neg Q}{Q}$$

$$\frac{Q}{\neg\neg Q}$$

$$\frac{[\neg Q] \quad \vdots \quad \textit{False}}{Q}$$

$$\frac{[\neg Q] \quad \vdots \quad Q}{Q}$$



Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \rightarrow B} \qquad \frac{A \rightarrow B \quad A}{B}$$



Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\frac{}{s = s}$$

$$\frac{s = t}{t = s}$$

$$\frac{r = s \quad s = t}{r = t}$$

$$\frac{s = t \quad P \ s}{P \ t}$$

$$\frac{\wedge x. s \ x = t \ x}{s = t}$$

Foundations: HOL / Core

- ... built on this syntactic material, we can give the rules (as axioms) of HOL:

$$\begin{array}{c}
 \frac{P \ ?t}{\exists x.P \ x} \\
 \\
 \frac{\exists x.P \ x \quad \begin{array}{c} [P \ ?t]_x \\ \vdots \\ Q \end{array}}{Q} \\
 \\
 \frac{\wedge x.P \ x}{\forall x.P \ x} \quad \frac{\forall x.P \ x \quad \begin{array}{c} [P \ ?t] \\ \vdots \\ Q \end{array}}{Q} \quad \frac{\forall x.P \ x \quad \begin{array}{c} [P \ ?t; \forall x.P \ x] \\ \vdots \\ Q \end{array}}{Q}
 \end{array}$$



Foundations: HOL / Core

- Theorem: HOL/Core is sound wrt. to standard and non-standard (Henkin-) models
- Theorem: HOL/Core is complete wrt. standard models!
- Theorem: HOL/Core + axiom of infinity is incomplete wrt. standard models (Gödel)
- Theorem: HOL/Core + axiom of infinity is complete wrt. (Henkin-) models (Andrews)



Foundations: HOL / Library

- Just adding axioms is extremely dangerous:
just consider $Y(F) = F(Y F) !!!$
- ... we need methods to extend logical systems
safely !



Foundations: HOL / Library

- ... we need **conservative** theory extensions:

$$(\Xi, \Sigma, \text{Rules}) \mapsto (\Xi', \Sigma', \text{Rules}')$$

with $\Xi \subseteq \Xi'$, $\Sigma \subseteq \Sigma'$, $\text{Rules} \subseteq \text{Rules}'$ and

$$\text{Models}(\Xi, \Sigma, R) = \text{Models}(\Xi', \Sigma', R')|_{\Sigma}$$

(modulo signature adaption, we get the same class of semantic interpretations as before ...)



Foundations: HOL / Library

- A) conservative theory extension

constant definition:

$$(\Xi, \Sigma, R) \mapsto (\Xi, \Sigma \cup \{c \mapsto \tau\}, R \cup \{c = E\})$$

- where c is fresh
- where E is closed and does not contain c
- where no free type-variables occur in E that do not occur in the type of c



Foundations: HOL / Library

- B) conservative theory extension *type definition* for $\tau' = \chi(\alpha_1 \dots \alpha_k)$ from set $E :: \tau(\alpha_1 \dots \alpha_k) \Rightarrow \text{bool}$

$$(\mathbb{E}, \Sigma, R) \mapsto (\mathbb{E} [k := \mathbb{E}_k \cup \{\chi_k\}], \\ \Sigma \cup \{\text{Abs}_\chi \mapsto \tau \Rightarrow \tau', \text{Rep}_\chi \mapsto \tau' \Rightarrow \tau\}, \\ R \cup \{A, B, C\})$$

- where χ_k , Abs_χ , Rep_χ are fresh and where
- A, B, C state an isomorphism between E and τ'

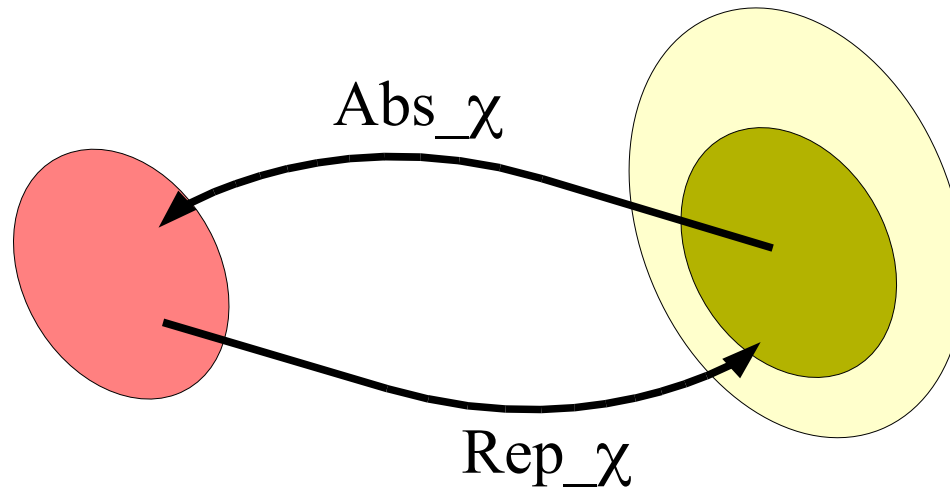


Foundations: HOL / Library

- B) conservative theory extension *type definition*
for $\tau' = \chi(\alpha_1.. \alpha_k)$ from set $E :: \tau(\alpha_1.. \alpha_k) \Rightarrow \text{bool}$
 - A: $\exists x. E x$ -- type consistency
 - B: $\text{Abs}_{\chi}(\text{Rep}_{\chi} x) = x$
 - C: $E x \Rightarrow \text{Rep}_{\chi}(\text{Abs}_{\chi} x) = x$

Foundations: HOL / Library

- B) conservative theory extension *type definition*
for $\tau' = \chi(\alpha_1.. \alpha_k)$ from set $E :: \tau(\alpha_1.. \alpha_k) \Rightarrow \text{bool}$





Foundations: HOL / Library

- With these two kinds of conservative extensions, the entire Library of Isabelle/HOL is built including:
 - set theory, inductive sets
 - wellfounded orders, well-founded recursion
 - arithmetic (nat,int,real,hyperreal, IEE754 floats)
 - data types, option, list, tree, ...
 - partial maps, updates, ...
 - programming language semantics (IMP, JAVA, JVM, ...)

Foundations: HOL / Library

■ Examples:

- type synonym (not even a type definition!)

`types` α set = " $\alpha \Rightarrow \text{bool}$ "

- constant definitions

`constdefs` Collect $::$ " $(\alpha \Rightarrow \text{bool}) \Rightarrow \alpha$ set"

Collect S \equiv S"

member $::$ " $\alpha \Rightarrow \alpha$ set $\Rightarrow \text{bool}$ "

member s S \equiv S x"

- syntactic paraphrasing (not shown here!):

Collect(λ x. A) \triangleq {x . A}, member s S \triangleq s \in S



Foundations: HOL / Library

- Examples:
 - type definitions (syntax simplified)

```
typedef ( $\alpha$ ,  $\beta$ ) "_ $\times$ _"  
= "{f.  $\exists a::\alpha b::\beta. f = \lambda x y. x = a \wedge y = b$ }"
```





Foundations: HOL / Library

- Examples:

- data type definitions
(automatically compiled to type definitions)

`datatype` α option = None | Some α

`datatype` α list = Nil | Cons α " α list"

- syntax:

$\text{Nil} \triangleq []$, $\text{Cons } a \ l \triangleq a \ \# \ l$



Foundations: HOL / Library

- Examples:

- primitive recursions

(automatically compiled to constant definitions)

consts ins :: "[α ::linorder, List α] \Rightarrow List α "

primrec

ins x [] = [x]

ins x (y#ys) = if x < y then x#(ins y ys) else y#(ins x ys)

consts sort :: "List(a::linorder) \Rightarrow List a"

primrec

sort [] = []

sort (x#xs) = ins x (sort xs)



Testing : Conclusions Partie - II

- HOL is a universal foundation for:
 - on **specifications** (a model what a program should do)
 - on **programs** (a model on how a program does behave)
 - on **symbolic computations** of both
- Thus, it is a good foundation for Model-based Testing and Tools like HOL-TestGen
<http://www.brucker.ch/projects/hol-testgen/>