DTU

Towards user-friendly proof mechanization

PhD defense

Frederik Krogsdal Jacobsen

October 11, 2024

What’s logic, anyway?

Today is October 11, 2024. ✓
Today is a Monday. ✗

What’s logic, anyway?

Today is October 11, 2024. ✓
Today is a Monday. ✗

If it is raining, then the ground is wet. It is raining. So the ground is wet. ✓

Patterns

If it is raining, then the ground is wet. It is raining. So the ground is wet.

If X, then Y. X. So Y.

$$(X \to Y) \implies X \implies Y.$$

Interpretations and Validity

Interpretations assign truth values to variables:

X is true	X is true	X is false	X is false
Y is true	Y is false	Y is true	Y is false

Valid formulas are true no matter which interpretation we choose:

$X \lor \neg X$ ✓

$X \land Y$ ✗

Deduction

$$\frac{}{\vDash X, \neg X}$$

$$\frac{\vDash X \qquad \vDash Y}{\vDash X \wedge Y}$$

$$\frac{\vDash X, Y}{\vDash X \vee Y}$$

How do we know our system works?

Two problems:

1. Maybe we can prove false things
2. Maybe we can't prove true things

Desirable properties:

1. Soundness
2. Completeness

Higher-order logic

How do we prove that our system works?

Functional programming + logic

Isabelle/HOL

Proof assistant for higher-order logic

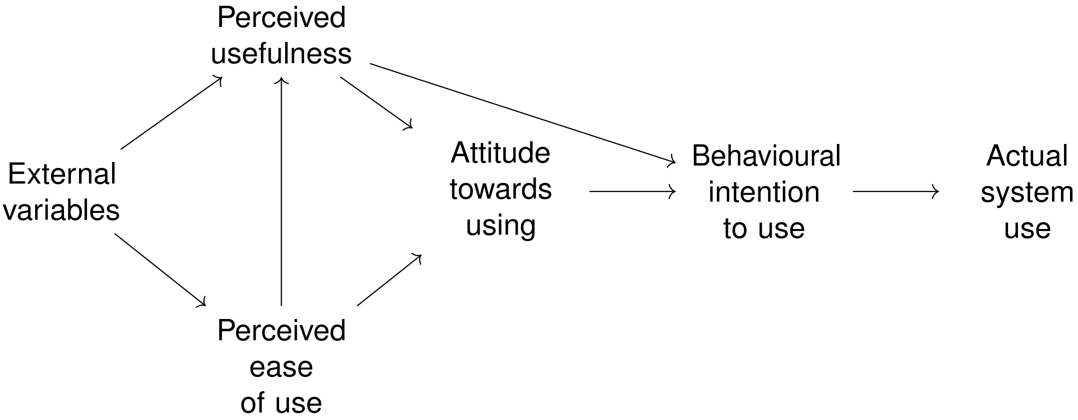
Isabelle/HOL

Proof assistant for higher-order logic

Automatic search for proofs
Automatic search for counterexamples
Export of definitions

Use cases

. . . so why not?


Perceived usefulness
External variables
Attitude towards using
Behavioural intention to use
Actual system use
Perceived ease of use

Themes

1. SeCaV: A Sequent Calculus Verifier in Isabelle/HOL
   Using Isabelle in Two Courses on Logic and Automated Reasoning
   Teaching Functional Programmers Logic and Metatheory
   On Exams with the Isabelle Proof Assistant
   ProofBuddy: A Proof Assistant for Learning and Monitoring
2. Verifying a Sequent Calculus Prover for First-Order Logic with Functions in Isabelle/HOL
3. The Concurrent Calculi Formalisation Benchmark

Learning to prove with Isabelle

Joint work with:
Jørgen Villadsen, Asta Halkjær From, Nadine Karsten, Kim Jana Eiken and Uwe Nestmann

Overview

1. Sequent Calculus Verifier
2. Learning with computer assistance
3. Enabling future research

Sequent Calculus Verifier

$$\frac{\text{Neg } p \in z}{\Vdash p, z}\ \textsc{Basic} \qquad \frac{\Vdash z \qquad z \subseteq y}{\Vdash y}\ \textsc{Ext} \qquad \frac{\Vdash p, z}{\Vdash \text{Neg } (\text{Neg } p), z}\ \textsc{NegNeg}$$

$$\frac{\Vdash p, q, z}{\Vdash \text{Dis } p\ q, z}\ \textsc{AlphaDis} \qquad \frac{\Vdash \text{Neg } p, z \qquad \Vdash \text{Neg } q, z}{\Vdash \text{Neg } (\text{Dis } p\ q), z}\ \textsc{BetaDis}$$

$$\frac{\Vdash p\,[\text{Var } 0/t], z}{\Vdash \text{Exi } p, z}\ \textsc{GammaExi}$$

$$\frac{\Vdash \text{Neg } (p\,[\text{Var } 0/\text{Fun } i\ []]), z \qquad i \text{ fresh}}{\Vdash \text{Neg } (\text{Exi } p), z}\ \textsc{DeltaExi}$$

Web interface

Sequent Calculus Verifier | Help and Input Examples | 27:6 | Copy Output to Clipboard | SeCaV Unshortener 1.A

```
Imp (Con (Uni p[0]) q) (Dis r (Exi p[0]))

AlphaImp
  Neg (Con (Uni p[0]) q)
  Dis r (Exi p[0])
AlphaCon
  Neg (Uni p[0])
  Neg q
  Dis r (Exi p[0])
GammaUni
  Neg p[a]
  Neg q
  Dis r (Exi p[0])
Ext
  Dis r (Exi p[0])
  Neg p[a]
AlphaDis
  r
  Exi p[0]
  Neg p[a]
Ext
  Exi p[0]
  Neg p[a]
GammaExi
  p[a]
  Neg p[a]
Basic
```

```
proposition ‹((∀x. (p x)) ∧ q) ⟶ (r ∨ (∃x. (p x)))› by metis

text ‹
  Predicate numbers
    0 = p
    1 = q
    2 = r
  Function numbers
    0 = a
  ›

lemma ‹⊩
  [
    Imp (Con (Uni (Pre 0 [Var 0])) (Pre 1 [])) (Dis (Pre 2 []) (Exi (Pre 0 [Var 0])))
  ]
  ›
proof -
  from AlphaImp have ?thesis if ‹⊩
    [
      Neg (Con (Uni (Pre 0 [Var 0])) (Pre 1 [])),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with AlphaCon have ?thesis if ‹⊩
    [
      Neg (Uni (Pre 0 [Var 0])),
      Neg (Pre 1 []),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with GammaUni[where t=‹Fun 0 []›] have ?thesis if ‹⊩
    [
      Neg (Pre 0 [Fun 0 []]),
      Neg (Pre 1 []),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with Ext have ?thesis if ‹⊩
```

Natural Deduction Assistant

Natural Deduction Assistant

Load Code Help ProofJudge 36/36 Stop Undo ☺

1	Imp_I	$[\,]\ ((A \to B) \to A) \to A$
2	Boole	$[(A \to B) \to A]\ A$
3	Imp_E	$[A \to \bot, (A \to B) \to A]\ \bot$
4	Assume	$[A \to \bot, (A \to B) \to A]\ A \to \bot$
5	Imp_E	$[A \to \bot, (A \to B) \to A]\ A$
6	Assume	$[A \to \bot, (A \to B) \to A]\ (A \to B) \to A$
7	Imp_I	$[A \to \bot, (A \to B) \to A]\ A \to B$
8	Boole	$[A, A \to \bot, (A \to B) \to A]\ B$
9	Imp_E	$[B \to \bot, A, A \to \bot, (A \to B) \to A]\ \bot$
10	Assume	$[B \to \bot, A, A \to \bot, (A \to B) \to A]\ A \to \bot$
11	Assume	$[B \to \bot, A, A \to \bot, (A \to B) \to A]\ A$

Progression

NaDeA $\longrightarrow$ SeCaV $\longrightarrow$ Isabelle (manual) $\longrightarrow$ Isabelle

Does our approach work?

(✓) Concrete implementations in a programming language aid understanding of concepts in logic

✓ Students experiment with definitions to gain understanding

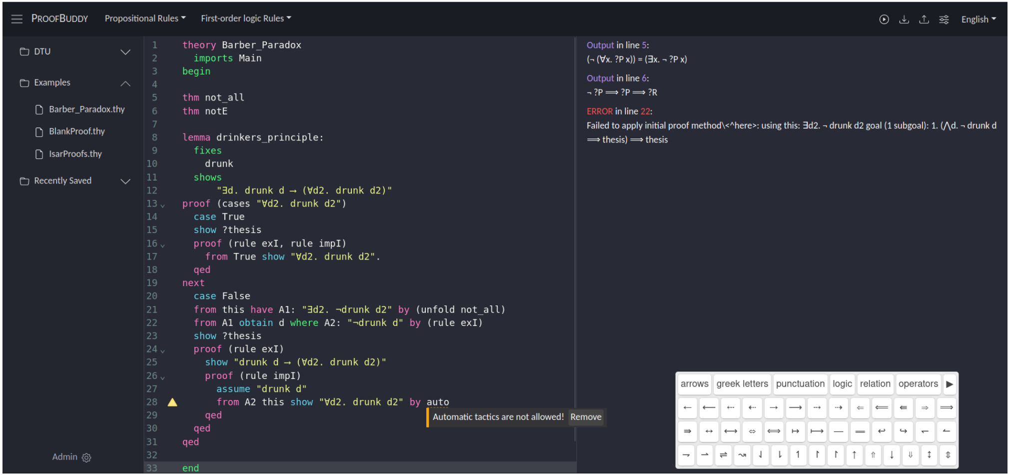
✓ Prior experience with functional programming is useful

✓ The approach gives students more confidence in their functional programming ability

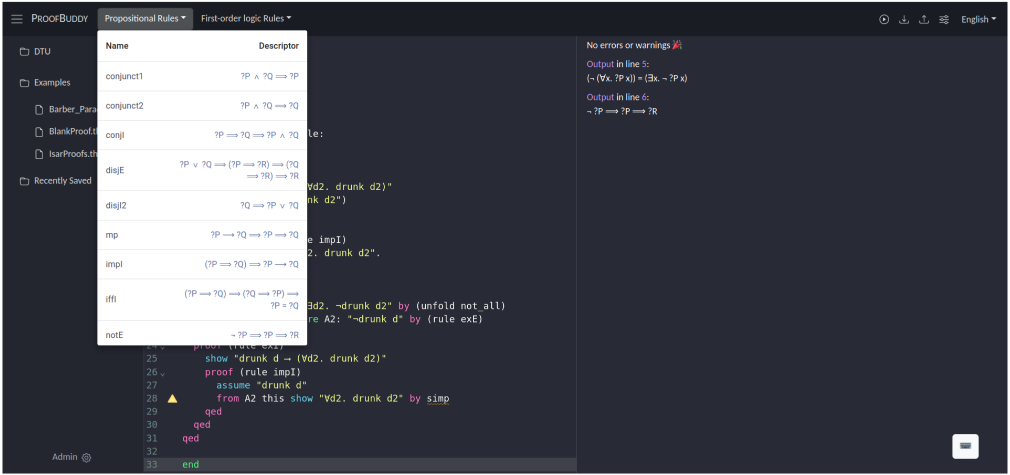
✗ Our formalizations make it clear how to implement concepts in practice

✗ Our course makes students able to design and implement their own systems

ProofBuddy: enabling future research



ProofBuddy: enabling future research



Verified and understandable automated reasoning

Joint work with Asta Halkjær From

Sequent Calculus Verifier

$$\frac{\text{Neg } p \in z}{\Vdash p, z}\ \textsc{Basic} \qquad \frac{\Vdash z \qquad z \subseteq y}{\Vdash y}\ \textsc{Ext} \qquad \frac{\Vdash p, z}{\Vdash \text{Neg } (\text{Neg } p), z}\ \textsc{NegNeg}$$

$$\frac{\Vdash p, q, z}{\Vdash \text{Dis } p\ q, z}\ \textsc{AlphaDis} \qquad \frac{\Vdash \text{Neg } p, z \qquad \Vdash \text{Neg } q, z}{\Vdash \text{Neg } (\text{Dis } p\ q), z}\ \textsc{BetaDis}$$

$$\frac{\Vdash p\,[\text{Var } 0/t], z}{\Vdash \text{Exi } p, z}\ \textsc{GammaExi}$$

$$\frac{\Vdash \text{Neg } (p\,[\text{Var } 0/\text{Fun } i\ []]), z \qquad i \text{ fresh}}{\Vdash \text{Neg } (\text{Exi } p), z}\ \textsc{DeltaExi}$$

Proof by programming

1. Come up with a method for applying rules to find a proof if one exists
2. Write a program that applies the rules
3. Prove that the program works

Some observations

$$\frac{\text{Neg}\ p \in z}{\Vdash p, z}\ \textsc{Basic} \qquad \frac{\Vdash z \qquad z \subseteq y}{\Vdash y}\ \textsc{Ext} \qquad \frac{\Vdash p, z}{\Vdash \text{Neg}\ (\text{Neg}\ p), z}\ \textsc{NegNeg}$$

$$\frac{\Vdash p, q, z}{\Vdash \text{Dis}\ p\ q, z}\ \textsc{AlphaDis} \qquad \frac{\Vdash \text{Neg}\ p, z \qquad \Vdash \text{Neg}\ q, z}{\Vdash \text{Neg}\ (\text{Dis}\ p\ q), z}\ \textsc{BetaDis}$$

$$\frac{\Vdash p\,[\text{Var}\ 0/t], z}{\Vdash \text{Exi}\ p, z}\ \textsc{GammaExi}$$

$$\frac{\Vdash \text{Neg}\ (p\,[\text{Var}\ 0/\text{Fun}\ i\ []]), z \qquad i\ \text{fresh}}{\Vdash \text{Neg}\ (\text{Exi}\ p), z}\ \textsc{DeltaExi}$$

A prover design

- Opportunistically check if BASIC applies
- Meta rules: apply to all matching formulas
- Remember all terms on the branch for GAMMA rules
- Keep trying all rules one by one
- If all branches are “done”, we have a proof!

Soundness

1. If the prover returns a proof, we can reconstruct a SeCaV proof
2. SeCaV is sound, so the prover is as well

Proof: by induction on the proof tree, reconstructing the SeCaV proof for each rule

Completeness

1. We either get a finite proof tree or one with an infinite (saturated) escape path
2. The root of a saturated escape path cannot be a valid formula
3. So valid formulas result in finite proof trees

The end result

- An automatic prover exported that can show its work
- Formally verified soundness and completeness of the prover in Isabelle/HOL

Formal proofs about concurrent systems

Joint work with:

Marco Carbone, David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, Alberto Momigliano, Luca Padovani, Alceste Scalas, Dawit Tirore, Martin Vassor, Nobuko Yoshida and Daniel Zackon

The Concurrent Calculi Formalisation Benchmark

Concurrent systems are hard!

Challenges:

1. Linearity and behavioural type systems
2. Name passing and scope extrusion
3. Coinduction and infinite processes

Linearity and behavioural type systems

Processes:

$$
\begin{array}{rcl}
v, w & ::= & a \quad | \quad l \\
P, Q & ::= & \mathbf{0} \quad | \quad x!v.P \quad | \quad x?(l).P \quad | \quad (P \mid Q) \quad | \quad (\nu xy)\ P
\end{array}
$$

Linearity and behavioural type systems

Processes:

$$
\begin{array}{lcl}
v, w & ::= & a \quad | \quad l \\
P, Q & ::= & \mathbf{0} \quad | \quad x!v.P \quad | \quad x?(l).P \quad | \quad (P \mid Q) \quad | \quad (\nu xy)\, P
\end{array}
$$

Semantics:

R-COM
$$\frac{}{(\nu xy)\,(x!a.P \mid y?(l).Q \mid R) \to (\nu xy)\,(P \mid Q\{a/l\} \mid R)}$$

R-RES
$$\frac{P \to Q}{(\nu xy)\, P \to (\nu xy)\, Q}$$

R-PAR
$$\frac{P \to Q}{P \mid R \to Q \mid R}$$

R-STRUCT
$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q \equiv Q'}{P \to Q}$$

Linearity and behavioural type systems

1. No endpoint is used simultaneously by parallel processes.
2. The two endpoints of the same channel are used dually.

Linearity and behavioural type systems

1. No endpoint is used simultaneously by parallel processes.
2. The two endpoints of the same channel are used dually.

Types:

$$
\begin{array}{rcl}
S, T & ::= & \textbf{end} \quad | \quad \textbf{base} \quad | \quad ?.S \quad | \quad !.S \\
\Gamma & ::= & \cdot \quad | \quad \Gamma, l \qquad \Delta \quad ::= \quad \cdot \quad | \quad \Delta, x : S
\end{array}
$$

Typing rules:

$$
\textsc{T-Inact}\ \frac{\textbf{end}(\Delta)}{\Gamma; \Delta \vdash \mathbf{0}}
\qquad
\textsc{T-Par}\ \frac{\Gamma; \boxed{\Delta_1} \vdash P \qquad \Gamma; \boxed{\Delta_2} \vdash Q}{\Gamma; \boxed{\Delta_1, \Delta_2} \vdash P \mid Q}
\qquad
\textsc{T-Res}\ \frac{\Gamma; (\Delta, x : \boxed{T}, y : \boxed{\overline{T}} \vdash P)}{\Gamma \vdash (\nu xy)\ P}
$$

$$
\textsc{T-Out}\ \frac{\Gamma \vdash_v v : \textbf{base} \qquad \Gamma; \Delta, x : T \vdash P}{\Gamma; (\Delta, x : !.T) \vdash x!v.P}
\qquad
\textsc{T-In}\ \frac{(\Gamma, l); (\Delta, x : T) \vdash P}{\Gamma; (\Delta, x : ?.T) \vdash x?(l).P}
$$

Name passing and scope extrusion

Processes:

$$P, Q := \mathbf{0} \quad | \quad (P \mid Q) \quad | \quad x!y.P \quad | \quad x?(y).P \quad | \quad (\nu x)\ P$$

One relevant example:

$$\big((\nu y)\ x!y.P\big) \mid (x?(z).Q)$$

Name passing and scope extrusion

First approach: structural congruence and reduction

$$
\begin{array}{ll}
((\nu y)\; x!y.P) \mid (x?(z).Q) & \equiv \\
(\nu y)\; (x!y.P \mid x?(z).Q) & \rightarrow \\
(\nu y)\; (P \mid Q\{y/z\}) &
\end{array}
$$

Name passing and scope extrusion

Second approach: labelled transition system

$$\dfrac{\dfrac{x!y.P \xrightarrow{x!y} P \qquad x \neq y}{(\nu y)\ x!y.P \xrightarrow{x!(y)} P} \qquad x?(z).Q \xrightarrow{x?y} Q\{y/z\} \qquad y \notin \mathrm{fn}(Q)}{((\nu y)\ x!y.P) \mid (x?(z).Q) \xrightarrow{\tau} (\nu y)\ (P \mid Q\{y/z\})}$$

Open

$$\dfrac{P \xrightarrow{x!z} P' \qquad z \neq x}{(\nu z)\ P \xrightarrow{x!(z)} P'}$$

Close-L

$$\dfrac{P \xrightarrow{x!(z)} P' \qquad Q \xrightarrow{x?z} Q' \qquad z \notin \mathrm{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\nu z)\ P' \mid Q'}$$

Coinduction and infinite processes

Describing the behaviour of recursive loops in programs.

$$
\begin{array}{lcl}
v, w & ::= & a \mid l \\
P, Q & ::= & \mathbf{0} \mid x!v.P \mid x?(l).P \mid (P \mid Q) \mid (\nu x)\, P \mid \,!P
\end{array}
$$

$$
\textsc{Rep}\quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid \,!P}
$$

Coinduction and infinite processes

Observability predicate:

$$P \downarrow_{x?} \quad \text{if } P \text{ can perform an input action via } x.$$
$$P \downarrow_{x!} \quad \text{if } P \text{ can perform an output action via } x.$$

Strong barbed bisimilarity:
the *largest* symmetric relation such that, whenever $P \overset{\bullet}{\sim} Q$:

$$P \downarrow_{\mu} \text{ implies } Q \downarrow_{\mu} \tag{1}$$
$$P \xrightarrow{\tau} P' \text{ implies } Q \xrightarrow{\tau} \overset{\bullet}{\sim} P' \tag{2}$$

Coinduction and infinite processes

Observability predicate:

$$P \downarrow_{x?} \quad \text{if } P \text{ can perform an input action via } x.$$
$$P \downarrow_{x!} \quad \text{if } P \text{ can perform an output action via } x.$$

Strong barbed bisimilarity:
the *largest* symmetric relation such that, whenever $P \stackrel{\bullet}{\sim} Q$:

$$P \downarrow_{\mu} \ \text{ implies } Q \downarrow_{\mu} \tag{1}$$
$$P \xrightarrow{\tau} P' \text{ implies } Q \xrightarrow{\tau} \stackrel{\bullet}{\sim} P' \tag{2}$$

Problem: not a congruence

Coinduction and infinite processes

Strong barbed congruence:
$P \simeq^c Q$, if $C[P] \stackrel{\bullet}{\sim} C[Q]$ for every context C.

Lemma

$\simeq^c$ is the largest congruence included in $\stackrel{\bullet}{\sim}$.

Coinduction and infinite processes

Strong barbed congruence:

$P \simeq^c Q$, if $C[P] \stackrel{\bullet}{\sim} C[Q]$ for every context C.

Lemma

$\simeq^c$ is the largest congruence included in $\stackrel{\bullet}{\sim}$.

Challenge:

Theorem

$P \simeq^c Q$ if, for any process R and substitution σ, $P\sigma \mid R \stackrel{\bullet}{\sim} Q\sigma \mid R$.

What are we going to do about it?

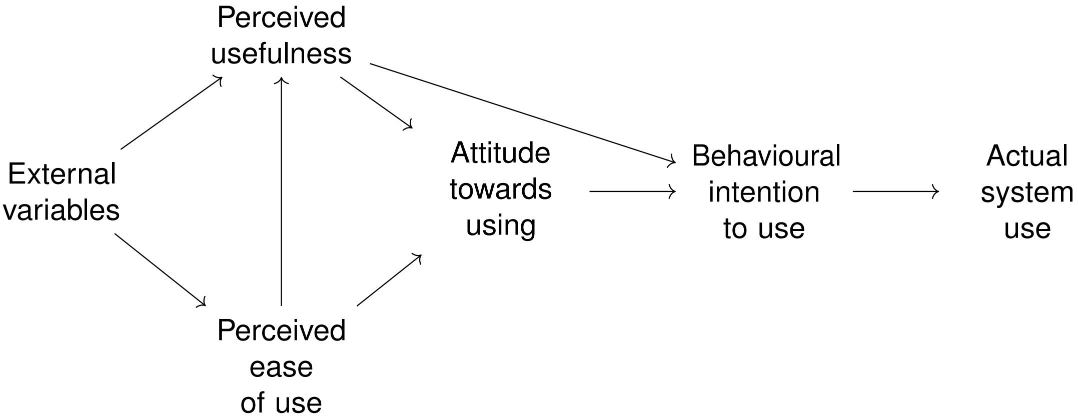
We want to encourage:

- Comparison of different approaches
- Development of guidelines, tutorials, techniques, libraries, . . .
- Reusable components

Conclusion

Conclusion

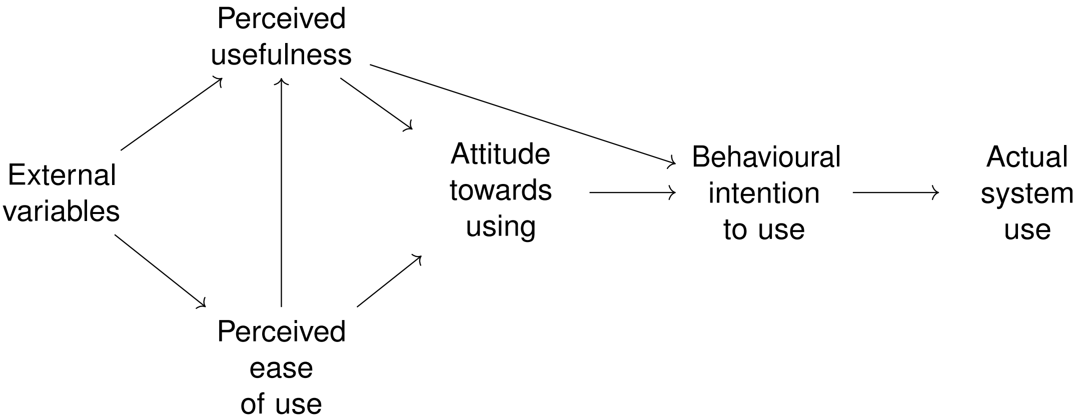
What have we accomplished?



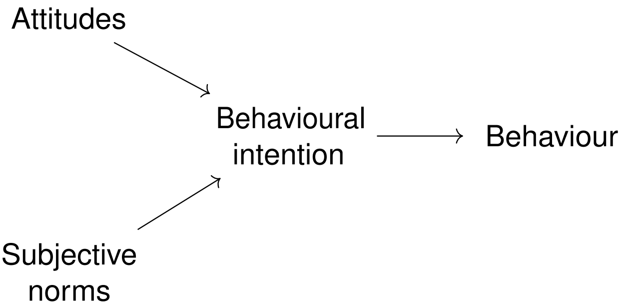
Bonus slides!

Models of technology adoption

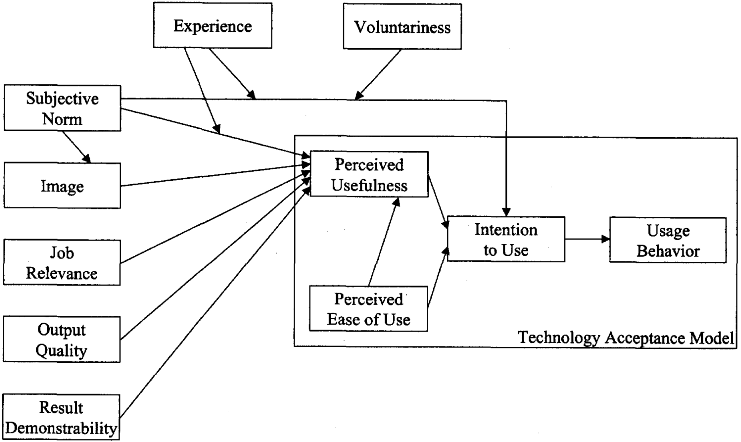
Technology Adoption Model



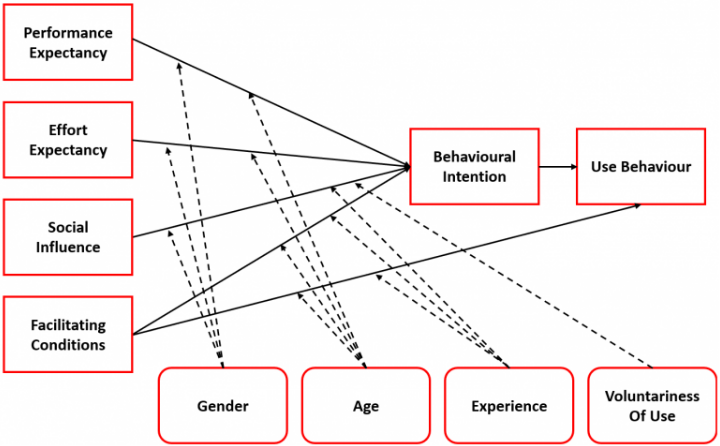
Theory of Reasoned Action



Technology Adoption Model 2



Unified Theory of Acceptance and Use of Technology



Others

- Lazy user model
- Matching Person and Technology model
- Hedonic-Motivation System Adoption Model

What are formal methods?

Some elements of the software development lifecycle

- Specification
- Development
- Verification
- Monitoring

Definitions

1. ISO 26262 (automotive safety)

Formal verification is the use of *any* method used to ensure correctness against a specification based on a notation with a completely defined syntax and semantics

2. ISO 24029 (assessment of the robustness of neural networks)

Formal methods are mathematical techniques for rigorous specification and verification of software and hardware systems with the goal to prove their correctness

3. Dines Bjørner and Klaus Havelund (in the paper 40 Years of Formal Methods)

By a formal method we shall understand a method whose techniques and tools can be explained in mathematics. If, for example, the method includes, as a tool, a specification language, then that language has a formal syntax, a formal semantics, and a formal proof system.

Perspectives

Are automated theorem provers formal methods?

1. ✓: they have completely defined syntax and semantics
2. ✓: they are mathematical and rigorous
3. ✓: they have a formal syntax, a formal semantics, and a formal proof system

Are interactive theorem provers formal methods?

1. ✓: they have completely defined syntax and semantics
2. ✓: they are mathematical and rigorous
3. ✓: they have a formal syntax, a formal semantics, and a formal proof system

Are model checkers formal methods?

1. ✓: they have completely defined syntax and semantics
2. ✓: they are mathematical and rigorous
3. ✓: they have a formal syntax, a formal semantics, and a formal proof system

Perspectives

Are handwritten proofs formal methods?

1. ✗: they do not have completely defined syntax and semantics
2. ✓: they are mathematical and rigorous
3. ✗: they do not have a formal syntax, a formal semantics, or a formal proof system

Are type checkers formal methods?

1. ✓: they have completely defined syntax and semantics
2. ✓: they are mathematical and rigorous
3. ✗: they (usually) do not have a formal proof system

Are tests formal methods?

1. ✓: they have completely defined syntax and semantics
2. ✗: they are not rigorous
3. ✗: they do not have a formal proof system

What does “user-friendly” mean?

Nobody agrees

(Non)-synonyms

- User-friendly
- Usable
- Accessible
- Good user experience
- Ease of use

Perspectives

Jakob Nielsen’s heuristics

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose and recover from errors
- Help and documentation

Perspectives

Laura Faulkner

It is a term that serves as a shortcut for a holistic concept of qualities and characteristics that cannot easily be captured in a few words of definitions.

A design that is the source of a simple experience after which a user visibly relaxes, with a moment of “knowing,” or the faint glow of a smile, before moving on to the next thing

My working definition

Useful and easy to use