# Verification with Automated Reasoning

Simon Foster

Monday $6^{\text{th}}$ March, 2017

INTO-CPS

into-cps.au.dk

Horizon 2020 Programme

# Outline

INTO-CPS

Motivation

Automated Reasoning and Isabelle

Verification by Unifying Theories of Programming

# Outline

Motivation

Automated Reasoning and Isabelle

Verification by Unifying Theories of Programming

# What is automated reasoning?

INTO-CPS

- natural language can have ambiguities and imprecision
- formal logic: a branch of mathematics that explores construction of propositions, theorems, and proofs

# What is automated reasoning?

INTO-CPS

- natural language can have ambiguities and imprecision
- formal logic: a branch of mathematics that explores construction of propositions, theorems, and proofs

$$time \in \{23{:}30 \ldots 04{:}00\} \Rightarrow dark$$

$$\forall\, x : \mathbb{N} \bullet \exists\, y : \mathbb{N} \bullet y > x$$

$$\forall\, x, y : \mathbb{Q} \bullet x < y \Rightarrow (\exists\, z : \mathbb{Q} \bullet x < z \wedge z < y)$$

# What is automated reasoning?

INTO-CPS

- natural language can have ambiguities and imprecision
- formal logic: a branch of mathematics that explores construction of propositions, theorems, and proofs

$$time \in \{23{:}30 \ldots 04{:}00\} \Rightarrow dark$$

$$\forall\, x : \mathbb{N} \bullet \exists\, y : \mathbb{N} \bullet y > x$$

$$\forall\, x, y : \mathbb{Q} \bullet x < y \Rightarrow (\exists\, z : \mathbb{Q} \bullet x < z \wedge z < y)$$

- allow to precisely form properties, as in Z and Circus
- prove (or falsify) properties using formal deduction rules
- theorem provers allow to (partially) automate this process
- apply to formally verify models and programs

# Why theorem proving?

- model checkers like FDR4 struggle with data structures and infinite state systems
- it is thus difficult to model check Circus
- state explosion problem – limit on the number of states

# Why theorem proving?

- model checkers like FDR4 struggle with data structures and infinite state systems
- it is thus difficult to model check Circus
- state explosion problem – limit on the number of states
- theorem provers allow to tackle problems symbolically
- no explicit representation of the state
- e.g. "the current state has $x > 5$ and $x < 10$"
- complementary to model checking: not quite "push button"

# Cyber-Physical Systems (CPSs)

- current "hot topic" in computer science research
- combine discrete computation (cyber-) with physical world
- interract with environment using sensors and actuators
- a controller makes decisions about behaviour
- can communicate with other systems via a network
- e.g. automated driverless cars
- INTO-CPS explores modelling and verification of CPS

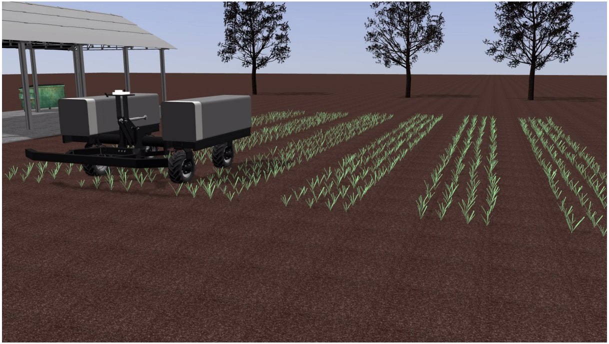
# Agricultural Robot

- example: Robotti agricultural robot (http://agrointelli.com/)

# Agricultural Robot

INTO-CPS

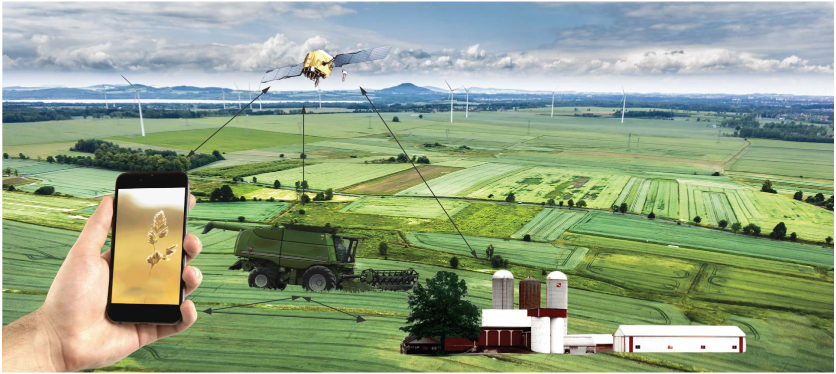
- example: Robotti agricultural robot (http://agrointelli.com/)



- immersive simulation and design space exploration

# Agricultural Robot

- example: Robotti agricultural robot (http://agrointelli.com/)



- integrated and automated farming processes

# Verifying CPSs



- such systems are complex to model and verify
- controller specified using a discrete notation like Circus
- environment modelled by differential equations
- very large state-space
- complex reasoning about real-numbers ($\mathbb{R}$)
- not simply infinite state, but uncountably infinite
- theorem proving thus an essential verification technique

# Outline

INTO-CPS

Motivation

Automated Reasoning and Isabelle

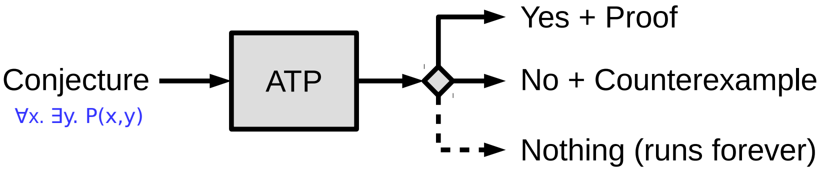
Verification by Unifying Theories of Programming

Horizon 2020 Programme

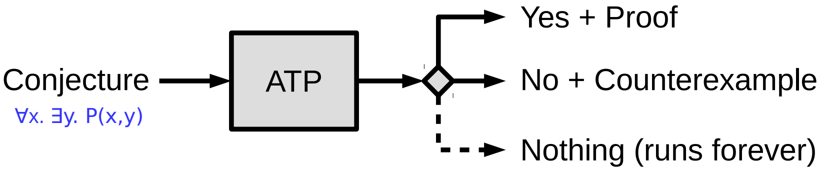
# Formal Proof

- conjecture: under some assumptions, a formula is true
- e.g. “assuming $x > 0$ then $x$ is a natural number”
- proof shows how to derive conclusion from assumptions
- by application of existing theorems and deduction rules
- analogy with function mapping inputs to outputs
- turns a conjecture into a theorem (or lemma)
- theorem provers and proof assistants aid us in this process

# Automated Theorem Provers



# Automated Theorem Provers




- can also use SMT solvers to prove arithmetic theorems etc.
- usually limited to first-order logic
- e.g. in general cannot handle induction
- induction required for proofs about failures-divergences
- thus we also need Interactive Theorem Proving

# Isabelle/HOL

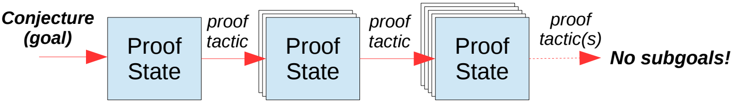
http://isabelle.in.tum.de

- an interactive theorem prover for Higher Order Logic (HOL)
- HOL = a functional specification language
- similarities to both Z and Haskell
- supports data structures, recursive functions, relations etc.
- allows readable proofs in “natural deduction” style
- large online library of formalised mathematics[1]
- support for verified code generation
- verification tools for Circus in progress

[1] Archive of Formal Proofs. http://afp.sf.net

# Proof in Isabelle

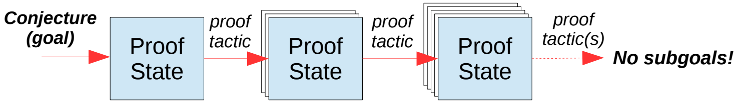
- an Isabelle proof is a script that acts on a proof state




- “divide and conquer” approach to proof
- uses proof tactics to subdivide and eliminate proof goals

# Proof in Isabelle

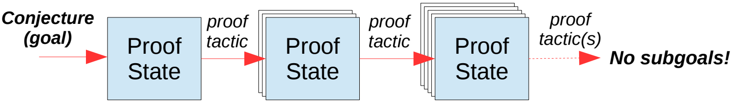
- an Isabelle proof is a script that acts on a proof state




- "divide and conquer" approach to proof
- uses proof tactics to subdivide and eliminate proof goals
  - simp – perform equational simplification ($1 + 2 \leadsto 3$)
  - blast and auto – automated deduction
  - sledgehammer – call external ATPs to find a proof
  - nitpick – try to find a counterexample

# Proof in Isabelle

- an Isabelle proof is a script that acts on a proof state




- "divide and conquer" approach to proof
- uses proof tactics to subdivide and eliminate proof goals
  - simp – perform equational simplification ($1 + 2 \rightsquigarrow 3$)
  - blast and auto – automated deduction
  - sledgehammer – call external ATPs to find a proof
  - nitpick – try to find a counterexample
- proof as a game where the winning condition is QED

# An aside


[2] Tobias Nipkow. Teaching Semantics with a Proof Assistant

# Demo 1: Isabelle proof goals

```
theorem ex1: "(1::int) + 2 = 3"
  by simp

theorem ex2:
  assumes "P ∧ R" "P ⟶ Q"
  shows "Q"
  using assms by simp

theorem ex3: "∀ x::nat. ∃y. y > x"
  oops

theorem ex4: "∃ x::nat. ∀y. y > x"
  oops
```

# Demo 2: Isabelle functions and theorems

INTO-CPS

```
datatype 'a seq = Nil | Cons 'a "'a seq"

fun length :: "'a seq ⇒ nat" ("#_" [999] 999) where
"#(Nil) = 0" |
"#(Cons x xs) = #xs + 1"

fun append :: "'a seq ⇒ 'a seq ⇒ 'a seq" (infixr "@" 65) where
"Nil @ xs = xs" |
"(Cons x xs) @ ys = Cons x (xs @ ys)"

theorem length_append: "#(xs @ ys) = #xs + #ys"
proof (induct xs)
  case Nil
  then show ?case by simp
next
  case (Cons x1 xs)
  then show ?case by simp
qed
```

# Outline

Motivation

Automated Reasoning and Isabelle

Verification by Unifying Theories of Programming

Horizon 2020 Programme

# Programs-as-predicates and the UTP

INTO-CPS

- how do we apply tools like Isabelle to program verification?

# Programs-as-predicates and the UTP

- how do we apply tools like Isabelle to program verification?
- UTP: encode programs as logical predicates
- allows to combine specifications and programs (as in Z)

# Programs-as-predicates and the UTP

INTO-CPS

- how do we apply tools like Isabelle to program verification?
- UTP: encode programs as logical predicates
- allows to combine specifications and programs (as in Z)

$$
\begin{aligned}
x := v \quad &\triangleq \quad x' = v \wedge y' = y \\
P \,;\, Q \quad &\triangleq \quad \exists\, x_0 \bullet P[x_0/x'] \wedge Q[x_0/x] \\
P \lhd b \rhd Q \quad &\triangleq \quad (b \wedge P) \vee (\neg b \wedge Q) \\
\textbf{while}\; b \;\textbf{do}\; P \quad &\triangleq \quad \mu\, X \bullet ((P \,;\, X) \lhd b \rhd I\!I)
\end{aligned}
$$

# Programs-as-predicates and the UTP

INTO-CPS

- how do we apply tools like Isabelle to program verification?
- UTP: encode programs as logical predicates
- allows to combine specifications and programs (as in Z)

$$
\begin{aligned}
x := v &\triangleq x' = v \land y' = y \\
P \,;\, Q &\triangleq \exists\, x_0 \bullet P[x_0/x'] \land Q[x_0/x] \\
P \lhd b \rhd Q &\triangleq (b \land P) \lor (\neg b \land Q) \\
\textbf{while}\; b\; \textbf{do}\; P &\triangleq \mu\, X \bullet ((P \,;\, X) \lhd b \rhd I\!I)
\end{aligned}
$$

- encoding programs in this way allows us to verify them
- program refinement: $\mathit{Spec} \sqsubseteq \mathit{Impl} \;\Leftrightarrow\; (\forall\, v \bullet \mathit{Impl} \Rightarrow \mathit{Spec})$
- Isabelle/UTP – automated reasoning for UTP

# Demo 3: Library in UTP

INTO-CPS

```
type_synonym book = string

alphabet library =
  books :: "book set"
  loans :: "book set"

abbreviation "Books ≡ {''War and Peace''
                      ,''Pride and Prejudice''
                      ,''Les Miserables''}"

definition InitLibrary :: "library prog" where
[upred_defs]: "InitLibrary = true ⊢ₙ books, loans := «Books», {}ᵤ"

definition InitLibraryAlt :: "library prog" where
[upred_defs]: "InitLibraryAlt = true ⊢ₙ ($books´ =ᵤ «Books» ∧ $loans´ =ᵤ {}ᵤ)"

lemma InitLibrary_alt_same: "InitLibrary = InitLibraryAlt"
  by (fast_rel_auto)

definition LibraryInvariant :: "library upred" where
[upred_defs]: "LibraryInvariant = (&loans ⊆ᵤ &books)"

definition BorrowBook :: "book ⇒ library prog" where
[upred_defs]: "BorrowBook(b) = («b» ∉ᵤ &loans ∧ «b» ∈ᵤ &books) ⊢ₙ loans := &loans ∪ᵤ {«b»}ᵤ"
```

# Demo 4: CSP in Isabelle

INTO-CPS

```
lemma ExtChoice_comm:
  "P □ Q = Q □ P"
  by (unfold extChoice_def, simp add: insert_commute)

lemma ExtChoice_idem:
  "P is CSP ⟹ P □ P = P"
  by (unfold extChoice_def, simp add: ExtChoice_single)

lemma ExtChoice_assoc:
  assumes "P is CSP" "Q is CSP" "R is CSP"
  shows "P □ Q □ R = P □ (Q □ R)"
proof -
  have "P □ Q □ R = Rs(preR(P) ⊢ cmtR(P)) □ Rs(preR(Q) ⊢ cmtR(Q)) □ Rs(preR(R) ⊢ cmtR(R))"
    by (simp add: SRD_reactive_design_alt assms(1) assms(2) assms(3))
  also have "... =
    Rs (((preR P ∧ preR Q) ∧ preR R) ⊢
         (((cmtR P ∧ cmtR Q) ◁ $tr´ =u $tr ∧ $wait´ ▷ (cmtR P ∨ cmtR Q)) ∧ cmtR R)
             ◁ $tr´ =u $tr ∧ $wait´ ▷
          ((cmtR P ∧ cmtR Q) ◁ $tr´ =u $tr ∧ $wait´ ▷ (cmtR P ∨ cmtR Q)) ∨ cmtR R)))"
    by (simp add: extChoice_rdes unrest)
  also have "... =
```

# Formal Semantics



- failures-divergences is a particular “semantic model”
- but it is just one of many theories of concurrency
- what about other models of concurrency? (e.g. mobility)
- object-orientation?
- real-time systems?
- hybrid systems and differential equations?
- and all combinations of the above?
- multi-paradigm languages are semantically heterogeneous

Horizon 2020 Programme

# Unifying Theories of Programming

- treat all the different theories as building blocks
- isolate them and study their fundamental laws
- construct foundations for heterogeneous languages
- CyPhyCircus – Circus + support for differential equations
- will enable formal modelling of examples like Robotti

# Conclusion

- theorem proving is an essential verification technique
- can be used to verify infinite state systems
- requires more input from the user
- however automation is improving all the time
- goal of the UTP is to formalise core computational theories
- Isabelle/UTP – mechanised programming laws
- we are applying it to verifying Cyber-Physical Systems

# Interested?

INTO-CPS

- Isabelle/UTP: https://github.com/isabelle-utp/utp-main
- Projects:
  - Integrating Theorem Proving and Computer Algebra Systems (simonf.isabelle-cas)
  - Mechanising the refinement calculus in Isabelle/UTP (simonf.refine-calc)
  - Automatic Translation from CSPm into Isabelle/UTP (zeyda.01)
  - Compositional analysis of interacting state machines for robotic applications (ahm504.02)
  - Formal refinement for a state-rich process algebra in Isabelle/HOL (ahm504.03)
  - Refinement support for a state-rich process algebra in Eclipse (ahm504.04)