This document uses the slide template from the "Interactive Theorem Proving Course" by Thomas Tuerk (https://www.thomas-tuerk.de):
https://github.com/thtuerk/ITP-course

Pablo Buiras (https://people.kth.se/~buiras/), Arve Gengelbach (https://people.kth.se/~arveg/) and Karl Palmskog (https://setoid.com) are the document authors.

Part XXI

HOL4 and ITPs in Research: an Overview

HOL4 and ITPs in Perspective

- this course only looks at the **basics** of ITP and HOL4
- thousands of researchers and engineers around the world use ITPs, and use is growing (mostly in Europe)
- ITPs can serve as **platforms** that connect programs, hardware, mathematical definitions, mathematical results
- many CS conferences and journals encourage use of ITPs; may become **mandatory** in the future

Proof Engineering

- developing large trustworthy systems using ITPs requires:
  - knowledge of strengths and limitations of selected ITP
  - good choice of underlying mathematical theories
  - careful selection of libraries and other tools
  - effective encodings of specifications and implementation
  - adequate development infrastructure and processes
  - ...
- the emerging field that considers these and related concerns holistically is called **proof engineering**
- see recent survey of proof engineering for program verification: `https://arxiv.org/abs/2003.06458`
- many concerns and ideas from software engineering apply—but generally unknown to what extent
- ITP languages tend to be nicely behaved and designed in comparison to many traditional programming languages

Strengths and Limitations of HOL4

- \+ based on classical higher order logic, a sweet spot between expressivity and ease of automation
- \+ trustworthy thanks to LCF approach
- \+ simple enough to understand easily
- \+ very easy to write custom proof tools, i. e. , own automation
- \+ reasonably fast and efficient
- \+ good automation
- \+ comprehensive bundled theories
- − can't have types depend on term, e. g. , R^n
- − no organizational mechanisms such as type classes or modules
- − no user interface (besides SML toplevel)
- − no special proof language
- − no IDE, very modest editor support
- − modestly-sized ecosystem, hard to google questions

Strengths and Limitations of Coq

+ based on a constructive higher-order type theory
+ logic is highly expressive (dependent types, universes, ...)
+ trustworthy thanks to small type checker
+ allows verified computation inside proofs (reflection)
+ functions are computable by default
+ large ecosystem, relatively easy to google questions
+ supported by many graphical interfaces and IDEs
– tactic execution and proof checking can be slow (explicit proofs)
– low level of built-in automation (need many plugins)
– many separately developed incompatible "standard" libraries
– difficult to write custom proof tools besides using tactic language

See a more detailed comparison of Coq and HOL:
`https://coq.discourse.group/t/why-doesnt-coq-have-a-theorem-type-like-hol-light/532`

Other ITPs

- HOL Light — like HOL4, but implemented in OCaml
- Isabelle/HOL — HOL logic with comprehensive automation, math proof language, advanced IDE, Archive of Formal Proofs
- Lean — Coq-like type theory with classical logic and built-in automation
- Agda — constructive Coq-like type theory
- ACL2 — first-order logic with strong automation (idealized version formalised in HOL4, Milawa)
- NuPRL & RedPRL — constructive extensional type theory

Definitions and theorem statements can often be directly transferred between systems, but proofs must typically be manually ported.

Learning and Reusing Libraries and Projects

- due to cost of using ITPs, important to avoid reinventing the wheel
- many CS applications already have several formalisations for a given ITP, but can still be inconvenient to "fit into" existing formalisation
- proper reuse of libraries may require careful study and experimentation
- there is probably no substitute for looking at lots of ITP code
- compare learning APIs in traditional programming languages such as Java and C++

HOL4 Theories I

Besides the basic libraries and theories that are required and loaded by `hol`, there are many more developments in HOL4's source directory.

- `src/sort` – sorting lists
- `src/string` – **strings**
- `src/TeX` – exporting LaTeX code
- `src/res_quan` – restricted quantifiers
- `src/quotient` – quotient type package
- `src/finite_map` – **finite map theory**
- `src/bag` – bags a. k. a. multisets
- `src/n-bit` – **machine words**

HOL4 Theories II

- `src/ring` – reasoning about rings
- `src/integer` – integers
- `src/llists` – lazy lists
- `src/path` – finite and infinite paths through a transition system
- `src/patricia` – efficient finite map implementations using trees
- `src/emit` – emitting SML and OCaml code
- `src/search` – traversal of graphs that may contain cycles
- `src/relation` – relations, including transition system **bisimulations**

HOL4 Theories III

- `src/rational` – rational numbers
- `src/real` – real numbers
- `src/complex` – comples numbers
- `src/HolQbf` – quantified boolean formulas
- `src/HolSmt` – **support for external SMT solvers**
- `src/float` – IEEE floating point numbers
- `src/floating-point` – new version of IEEE floating point numbers
- `src/probability` – **probability theory**
- `src/temporal` – shallow embedding of temporal logic
- ...

HOL4 Selected Examples I

The directory examples hosts many theories and libraries as well. There is not always a clear distinction between an example and a development in `src`. However, in general examples are more specialised and often larger. They are not required to follow HOL4's coding style as much as developments in `src`.

- `examples/balanced_bst` – **finite maps via balanced trees**
- `examples/unification` – (nominal) unification
- `examples/Crypto` – various block ciphers
- `examples/elliptic` – elliptic curve cryptography
- `examples/formal-languages` – regular and context free formal languages
- `examples/computability` – basic computability theory

HOL4 Selected Examples II

- `examples/set-theory` – axiomatic formalisation of set theory
- `examples/lambda` – lambda calculus
- `examples/acl2` – connection to ACL2 prover
- `examples/theorem-prover` – soundness proof of Milawa prover
- `examples/PSL` – formalisation of PSL
- `examples/HolBdd` – Binary Decision Diagrams
- `examples/HolCheck` – basic model checker
- `examples/temporal_deep` – deep embedding of temporal logics and automata

HOL4 Selected Examples III

- `examples/pgcl` formalisation of pGCL (the Probabilistic Guarded Command Language)
- `examples/dev` – some hardware compilation
- `examples/STE` – symbolic trajectory evalutation
- `examples/separationLogic` – formalisation of separation logic
- `examples/ARM` – formalisation of ARM architecture
- `examples/l3-machine-code` – l3 language
- `examples/machine-code` – compilers and decompilers to machine-code
- . . .

Some Ongoing Research Projects Related to HOL4

- Rigorous Engineering of Mainstream Systems (REMS)
  - `https://www.cl.cam.ac.uk/~pes20/rems/`
  - NetSem, validated formalisation of the TCP/IP stack
  - SAIL language for defining Instruction-Set Architecture (ISA) models
- CakeML, including Candle, a verified HOL interactive theorem prover
- Formalisation of Network Interface Controllers
- **HolBA** and **SCAM-V**

HolBA

- Binary analysis platform in HOL4
- Toolkit to analyse and reason about low-level (assembly) code
- Relies on formal semantics of ISAs (ARM/Risc-V/etc)
- Binary Intermediate Representation (BIR)
  - Language designed to automate analysis
  - Formal semantics in HOL4
  - Similar to LLVM IR
- Program not in memory / Assertions
- Verified theories and proof producing analyses
  - Transpilation into BIR
  - Weakest precondition
  - Symbolic execution

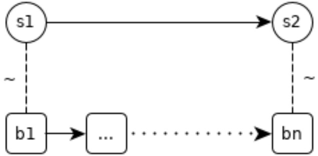
HolBA Lifter (transpiler)

```
0: pop R1
4: push R1
```

⇓

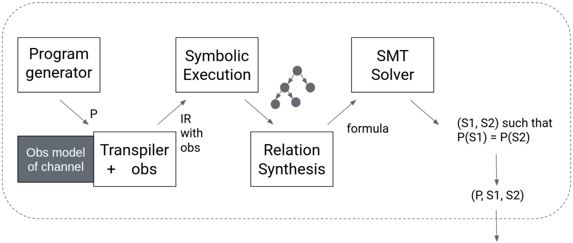
```
[0 {  R1 := MEM[SP];
      SP := SP-4;
      PC := PC+4;
      JMP 4}]
[4 {  MEM := MEM with [SP<-R1];
      SP := SP+4;
      PC := PC+4;
      JMP 8}]
```

Simulation theorem:



SCAM-V: Side-channel abstract model validator

- Modern architectures are too complex to directly analyse side-channels
- Abstract models based on system-state observations
- **Assumption:** States with equivalent observations in the model are indistinguishable to the attacker on real hardware
- SCAM-V **validates** this by generating and testing states that are supposed to be indistinguishable.



Verification vs validation

- **Verification:** formally proving that the program satisfies its formal specification
- **Validation:** testing that the program satisfies its formal specification
- Validation cannot prove the absence of errors
- In most cases, verification is preferred, but it is much more costly
- Sometimes it's not possible, e.g. soundness of side-channel models with respect to microarchitecture
- Final decision sometimes boils down to risk assessment and resource allocation
- It is possible to mix and match depending on requirements, i.e. critical modules can be verified, while less critical ones can be validated

Some Ongoing Research in Proof Engineering

- improvements and extensions to portability tools such as Ott and Lem
- improvement and verification of incremental proof checking — `https://setoid.com/chip`
- proof repair and proof transfer — `https://proofengineering.org`
- mutation analysis of ITP theories to find weak specifications — `http://cozy.ece.utexas.edu/mcoq/`
- learning and suggesting naming and formatting in ITP code

Trends in ITP for Research

- type theory ITPs heavily used programming languages research (POPL, PLDI)
- HOL-based ITPs often used in hardware-related research (FMCAD) and automated reasoning (IJCAR)
- lots of work on formalising mathematics, but ongoing debate which foundation and ITP should be used (CPP, ITP)
- ITP interface research may see a resurgence as ITPs go more mainstream
- bias towards "complete" formalisations for getting research published

Trends in Industrial Application of ITPs

- ITP verified cryptographic code included in Google Chrome
- seL4 and CompCert starting to get used in embedded systems
- Dune build system includes verified cycle checking code
- hardware designers and manufacturers look to apply formal verification (again?)

Challenges in Research Using ITPs

- working in a research group with knowledge of ITPs helps a lot, but documentation and resources for individual work are improving
- ITP experts may have unrealistically high expectations when reviewing applied work
- researchers without ITP expertise may underestimate effort and difference in trustworthiness
- personal experience: ITP community much friendlier to engineering research than the software engineering community is to ITP-based research