Twenty Years of Theorem Proving for HOLs
Past, Present and Future

- *Past*

- *Present*

- *Future*

Twenty Years of Theorem Proving for HOLs
Past, Present and Future

- *Past*
  - see “From 1988 to 2008” in my abstract in the Proceedings
- *Present*
  - see tutorials on ACL2, Coq, HOL4, Isabelle and PVS
- *Future*
  - what I’ll concentrate on!

Twenty Years of Theorem Proving for HOLs
Past, Present and Future

- *Past*
  - see “From 1988 to 2008” in my abstract in the Proceedings
- *Present*
  - see tutorials on ACL2, Coq, HOL4, Isabelle and PVS
- *Future*
  - what I’ll concentrate on!

Twenty Years of Theorem Proving for HOLs
Past, Present and Future

- *Past*
  - see “From 1988 to 2008” in my abstract in the Proceedings
- *Present*
  - see tutorials on ACL2, Coq, HOL4, Isabelle and PVS
- *Future*
  - what I’ll concentrate on!

Twenty Years of Theorem Proving for HOLs
Past, Present and Future

- *Past*
  - see “From 1988 to 2008” in my abstract in the Proceedings
- *Present*
  - see tutorials on ACL2, Coq, HOL4, Isabelle and PVS
- *Future*
  - what I’ll concentrate on!

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let’s celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let’s celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

First ... let's celebrate some **Amazing Achievements**!

- Powerful automatic theorem proving
  - SAT, decision procedures, SMT, first-order reasoners
- Logic extensions for modelling
  - type classes, locales, nominal logic, reflection, HOL-Omega
- New interactive proof methodologies
  - declarative proof, Quickcheck, SAT refutate
- Impressive theorems
  - four colour, Jordan curve, fundamental theorem of calculus
  - multivariate analysis, measure theory
- Applications
  - Java, Ada, C, C++, compilers, OS fragments, Z, OWR, FEF
  - floating point, security protocols, air traffic control
- Theorem prover as implementation platform
  - executable logic, verifiers as derived rules
  - links to external tools, e.g. Vampire, Simulink, LabVIEW

The future

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

The future

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

1973: programming in logic was just a dream

- Robert Kowalski
  Predicate Logic as Programming Language

  > *“... predicate logic is a useful and practical, high-level, non-deterministic programming language with sound theoretical foundations.”*

- P.J. Hayes
  Computation and Deduction

  > *“An interpreter for a programming language, and a theorem-proving program for a logical language, are structurally indistinguishable. “*

Logic and functional programming

- Logic Programming
  - Kowalski has relational vision of programming as deduction
  - execution by a resolution theorem prover
  - Colmerauer develops Prolog
- Functional Programming
  - Hayes has functional vision of computation as deduction
  - execution by resolution and paramodulation
  - rewriting-based languages (OBJ, Maude, ASF+SDF)
- 2008: easily programmed in a modern theorem prover

Logic and functional programming

- Logic Programming
  - Kowalski has relational vision of programming as deduction
  - execution by a resolution theorem prover
  - Colmerauer develops Prolog
- Functional Programming
  - Hayes has functional vision of computation as deduction
  - execution by resolution and paramodulation
  - rewriting-based languages (OBJ, Maude, ASF+SDF)
- 2008: easily programmed in a modern theorem prover

Logic and functional programming

- Logic Programming
  - Kowalski has relational vision of programming as deduction
  - execution by a resolution theorem prover
  - Colmerauer develops Prolog
- Functional Programming
  - Hayes has functional vision of computation as deduction
  - execution by resolution and paramodulation
  - rewriting-based languages (OBJ, Maude, ASF+SDF)
- 2008: easily programmed in a modern theorem prover

Logic and functional programming

- Logic Programming
  - Kowalski has relational vision of programming as deduction
  - execution by a resolution theorem prover
  - Colmerauer develops Prolog
- Functional Programming
  - Hayes has functional vision of computation as deduction
  - execution by resolution and paramodulation
  - rewriting-based languages (OBJ, Maude, ASF+SDF)
- 2008: easily programmed in a modern theorem prover

Computation = Logic + Control

- Classic functional and logic programming
  - programming is writing logic formulas
  - control of execution implicit in form of formula
  - execution using a "uniform proof procedure"
- Programming in logic using a theorem prover
  - programming is still writing logic formulas
  - execution by user-customised proof procedure
  - efficiency requires ingenuity by proof script programmer

Computation = Logic + Control

- Classic functional and logic programming
  - programming is writing logic formulas
  - control of execution implicit in form of formula
  - execution using a “uniform proof procedure”
  - efficient formulas might lose declarative clarity
- Programming in logic using a theorem prover
  - programming is still writing logic formulas
  - execution by user-customised proof procedure
  - efficiency requires ingenuity by proof script programmer
  - formulas optimised for declarative clarity not efficiency

Computation = Logic + Control

- Classic functional and logic programming
  - programming is writing logic formulas
  - control of execution implicit in form of formula
  - execution using a “uniform proof procedure”
  - efficient formulas might lose declarative clarity
- Programming in logic using a theorem prover
  - programming is still writing logic formulas
  - execution by user-customised proof procedure
  - efficiency requires ingenuity by proof script programmer
  - formulas optimised for declarative clarity not efficiency

Computation = Logic + Control

- Classic functional and logic programming
  - programming is writing logic formulas
  - control of execution implicit in form of formula
  - execution using a “uniform proof procedure”
  - efficient formulas might lose declarative clarity
- Programming in logic using a theorem prover
  - programming is still writing logic formulas
  - execution by user-customised proof procedure
  - efficiency requires ingenuity by proof script programmer
  - formulas optimised for declarative clarity not efficiency

Computation = Logic + Control

- Classic functional and logic programming
  - programming is writing logic formulas
  - control of execution implicit in form of formula
  - execution using a “uniform proof procedure”
  - efficient formulas might lose declarative clarity
- Programming in logic using a theorem prover
  - programming is still writing logic formulas
  - execution by user-customised proof procedure
  - efficiency requires ingenuity by proof script programmer
  - formulas optimised for declarative clarity not efficiency

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

A theorem prover is a programming environment

- Modern provers *all* support programming in logic
  - ACL2, Coq, Isabelle, HOL (various), PVS
- Good efficiency
  - especially ACL2, Coq, PVS
- Programs as logic terms have a tractable semantics
  - unlike modern logic and functional programming languages
- Already substantial examples of programs written in logic
  - Compcert Clight compiler
  - processor models (ARM, Rockwell Collins)
- Can interface to external solvers
  - BDD, SAT, SMT, FOL
  - special purpose tools (e.g. CIL, GNU assembler)

Moving to next generation software verifiers

- Now: shallow properties of real code
  or
  deep properties of toy code
- Future: shallow properties of real code
  and
  deep properties of real code
- Extend shallow analysis to full functional correctness
  - shape analysis: result is a list
  - full correctness: result is sorted permutation of input
- Future verifiers programmed in a theorem prover
  - long-term idealism: everything programmed by deduction
  - short-term pragmatism: trust external oracles

Moving to next generation software verifiers

- Now: shallow properties of real code
  or
  deep properties of toy code
- Future: shallow properties of real code
  and
  deep properties of real code
- Extend shallow analysis to full functional correctness
  - shape analysis: result is a list
  - full correctness: result is sorted permutation of input
- Future verifiers programmed in a theorem prover
  - long-term idealism: everything programmed by deduction
  - short-term pragmatism: trust external oracles

Moving to next generation software verifiers

- Now: shallow properties of real code
  or
  deep properties of toy code
- Future: shallow properties of real code
  and
  deep properties of real code
- Extend shallow analysis to full functional correctness
  - shape analysis: result is a list
  - full correctness: result is sorted permutation of input
- Future verifiers programmed in a theorem prover
  - long-term idealism: everything programmed by deduction
  - short-term pragmatism: trust external oracles

Moving to next generation software verifiers

- Now: shallow properties of real code
  or
  deep properties of toy code
- Future: shallow properties of real code
  and
  deep properties of real code
- Extend shallow analysis to full functional correctness
  - shape analysis: result is a list
  - full correctness: result is sorted permutation of input
- Future verifiers programmed in a theorem prover
  - long-term idealism: everything programmed by deduction
  - short-term pragmatism: trust external oracles

Moving to next generation software verifiers

- Now: shallow properties of real code
  or
  deep properties of toy code
- Future: shallow properties of real code
  and
  deep properties of real code
- Extend shallow analysis to full functional correctness
  - shape analysis: result is a list
  - full correctness: result is sorted permutation of input
- Future verifiers programmed in a theorem prover
  - long-term idealism: everything programmed by deduction
  - short-term pragmatism: trust external oracles

Beyond Church

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

Beyond Church

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
  - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
  - impressive Boyer-Moore proofs (e.g. Gödel's theorem)
  - can't do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
  - sufficient for almost all mathematics
  - but not for functional programs (e.g. can't express monads)
- Fancier type theories plug some gaps
  - PVS significantly more expressive than HOL
  - Coq can express everything, but
- Why not set theory
  - no set theory system as good as today's HOL systems

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
  - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
  - impressive Boyer-Moore proofs (e.g. Gödel's theorem)
  - can't do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
  - sufficient for almost all mathematics
  - but not for functional programs (e.g. can't express monads)
- Fancier type theories plug some gaps
  - PVS significantly more expressive than HOL
  - Coq can express everything, but . . .
- Why not set theory
  - no set theory system as good as today's HOL systems

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
  - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
  - impressive Boyer-Moore proofs (e.g. Gödel's theorem)
  - can't do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
  - sufficient for almost all mathematics
  - but not for functional programs (e.g. can't express monads)
- Fancier type theories plug some gaps
  - PVS significantly more expressive than HOL
  - Coq can express everything, but . . .
- Why not set theory
  - no set theory system as good as today's HOL systems

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
    - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
    - impressive Boyer-Moore proofs (e.g. Gödel's theorem)
    - can't do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
    - sufficient for almost all mathematics
    - but not for functional programs (e.g. can't express monads)
- Fancier type theories plug some gaps
    - PVS significantly more expressive than HOL
    - Coq can express everything, but . . .
- Why not set theory
    - no set theory system as good as today's HOL systems

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
  - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
  - impressive Boyer-Moore proofs (e.g. Gödel's theorem)
  - can't do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
  - sufficient for almost all mathematics
  - but not for functional programs (e.g. can't express monads)
- Fancier type theories plug some gaps
  - PVS significantly more expressive than HOL
  - Coq can express everything, but . . .
- Why not set theory
  - no set theory system as good as today's HOL systems

Is HOL powerful enough?

- Amazing what one can due even with propositional logic
  - e.g. bit-blasting then SAT
- First order logic (FOL) might seem enough
  - impressive Boyer-Moore proofs (e.g. Gödel’s theorem)
  - can’t do standard mathematics *directly* (need set theory)
- Simple type theory (HOL) is almost enough
  - sufficient for almost all mathematics
  - but not for functional programs (e.g. can’t express monads)
- Fancier type theories plug some gaps
  - PVS significantly more expressive than HOL
  - Coq can express everything, but . . .
- Why not set theory
  - no set theory system as good as today’s HOL systems

A significant step: Peter Homeier's HOL-Omega

- HOL-Omega is an extension of HOL4
  - inspired by and extends Norbert Völker's HOL2P
  - but doesn't stop at second order types
- Metatheory still undergoing certification!
  - intuitively plausible, but needs formal soundness proof
  - intented to have set-theoretic model
- Handles functional programming idioms impossible in HOL
  - monads
  - differently typed instances of a variable:
    $\forall\phi.\ \mathit{functor}(\phi) = \forall f\ g.\ \phi(f \circ g) = (\phi\ f) \circ (\phi\ g)$
    (example from Norbert Völker's TPHOLs 2007 paper)
- Available now!
  - fully compatible with existing HOL4 system
  - `svn checkout https://hol.svn.sf.net/svnroot/hol/branches/HOL-Omega`

A significant step: Peter Homeier’s HOL-Omega

- HOL-Omega is an extension of HOL4
  - inspired by and extends Norbert Völker’s HOL2P
  - but doesn’t stop at second order types
- Metatheory still undergoing certification!
  - intuitively plausible, but needs formal soundness proof
  - intented to have set-theoretic model
- Handles functional programming idioms impossible in HOL
  - monads
  - differently typed instances of a variable:
    $\forall\phi.\ \mathit{functor}(\phi) = \forall f\ g.\ \phi(f \circ g) = (\phi\ f) \circ (\phi\ g)$
    (example from Norbert Völker’s TPHOLs 2007 paper)
- Available now!
  - fully compatible with existing HOL4 system
  - `svn checkout https://hol.svn.sf.net/svnroot/hol/branches/HOL-Omega`

A significant step: Peter Homeier's HOL-Omega

- HOL-Omega is an extension of HOL4
  - inspired by and extends Norbert Völker's HOL2P
  - but doesn't stop at second order types
- Metatheory still undergoing certification!
  - intuitively plausible, but needs formal soundness proof
  - intented to have set-theoretic model
- Handles functional programming idioms impossible in HOL
  - monads
  - differently typed instances of a variable:
    $\forall\phi.\ \mathit{functor}(\phi) = \forall f\ g.\ \phi(f \circ g) = (\phi\ f) \circ (\phi\ g)$
    (example from Norbert Völker's TPHOLs 2007 paper)
- Available now!
  - fully compatible with existing HOL4 system
  - `svn checkout https://hol.svn.sf.net/svnroot/hol/branches/HOL-Omega`

A significant step: Peter Homeier's HOL-Omega

- HOL-Omega is an extension of HOL4
  - inspired by and extends Norbert Völker's HOL2P
  - but doesn't stop at second order types
- Metatheory still undergoing certification!
  - intuitively plausible, but needs formal soundness proof
  - intented to have set-theoretic model
- Handles functional programming idioms impossible in HOL
  - monads
  - differently typed instances of a variable:
    $\forall\phi.\ \mathit{functor}(\phi) = \forall f\ g.\ \phi(f \circ g) = (\phi\ f) \circ (\phi\ g)$
    (example from Norbert Völker's TPHOLs 2007 paper)
- Available now!
  - fully compatible with existing HOL4 system
  - `svn checkout https://hol.svn.sf.net/svnroot/hol/branches/HOL-Omega`

A significant step: Peter Homeier's HOL-Omega

- HOL-Omega is an extension of HOL4
  - inspired by and extends Norbert Völker's HOL2P
  - but doesn't stop at second order types
- Metatheory still undergoing certification!
  - intuitively plausible, but needs formal soundness proof
  - intented to have set-theoretic model
- Handles functional programming idioms impossible in HOL
  - monads
  - differently typed instances of a variable:
    $\forall \phi .\ \mathit{functor}(\phi) = \forall f\ g .\ \phi(f \circ g) = (\phi\ f) \circ (\phi\ g)$
    (example from Norbert Völker's TPHOLs 2007 paper)
- Available now!
  - fully compatible with existing HOL4 system
  - `svn checkout https://hol.svn.sf.net/svnroot/hol/branches/HOL-Omega`

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

The lure of Set Theory

- Standard
  - widely taught in schools and university
  - what mathematicians view as foundation
- Underlies popular specification methods
  - Z, VDM, TLA+
- Well understood axiomatisations (e.g. ZFC)
  - stable compared with type theory
- More expressive than HOL
  - Scott domains (D_∞), category theory (monads)
- Potential *lingua franca*
  - classical HOL logics can be embedded in set theory

An old dream

- Best of both worlds: type theory on top of set theory
- Soft types defined as sets
  - typechecking becomes ordinary theorem proving
  - types are first class (quantified, passed as parameters etc.)
  - higher order types (HOL-Omega) just definable
- Functions are sets
  - define λ-notation: $(\lambda x.\ E[x]) = \{(x, y) \mid y = E[x]\}$
  - define function application: $f \diamond x = \varepsilon y.\ \langle x, y \rangle \in f$
  - HOL logic proof infrastructure derived

An old dream

- Best of both worlds: type theory on top of set theory
- Soft types defined as sets
    - typechecking becomes ordinary theorem proving
    - types are first class (quantified, passed as parameters etc.)
    - higher order types (HOL-Omega) just definable
- Functions are sets
    - define λ-notation: $(\lambda x.\, E[x]) = \{(x, y) \mid y = E[x]\}$
    - define function application: $f \diamond x \;=\; \varepsilon y.\, \langle x, y\rangle \in f$
    - HOL logic proof infrastructure derived

An old dream

- Best of both worlds: type theory on top of set theory
- Soft types defined as sets
  - typechecking becomes ordinary theorem proving
  - types are first class (quantified, passed as parameters etc.)
  - higher order types (HOL-Omega) just definable
- Functions are sets
  - define λ-notation: $(\lambda x.\, E[x]) = \{(x, y) \mid y = E[x]\}$
  - define function application: $f \diamond x \;=\; \varepsilon y.\, \langle x, y\rangle \in f$
  - HOL logic proof infrastructure derived

An old dream

- Best of both worlds: type theory on top of set theory
- Soft types defined as sets
  - typechecking becomes ordinary theorem proving
  - types are first class (quantified, passed as parameters etc.)
  - higher order types (HOL-Omega) just definable
- Functions are sets
  - define λ-notation: $(\lambda x.\, E[x]) = \{(x, y) \mid y = E[x]\}$
  - define function application: $f \diamond x \;=\; \varepsilon y.\, \langle x, y\rangle \in f$
  - HOL logic proof infrastructure derived

I tried to implement set theory but failed!

- Tried to engineer too slick HOL shallow embedding
  - needed to trust complex ML encoding polymorphism
  - e.g. $I : \alpha \to \alpha$ vs. $(\lambda x : ST.\ x)$ vs. $\{(x,x) \mid x \in \alpha\} : ST$
- Tempted by elegance of ZF axioms in HOL
  - e.g. replacement: $\forall f\ s.\ \exists t.\ \forall y.\ y \in t \Leftrightarrow \exists x \in s.\ y = f\,x$
  - worried about relation to standard ZF in FOL
- But I still think set theory should be the long term aim!

I tried to implement set theory but failed!

- Tried to engineer too slick HOL shallow embedding
  - needed to trust complex ML encoding polymorphism
  - e.g. $I : \alpha \to \alpha$ vs. $(\lambda x : \mathrm{ST}.\ x)$ vs. $\{(x, x) \mid x \in \alpha\} : \mathrm{ST}$
- Tempted by elegance of ZF axioms in HOL
  - e.g. replacement: $\forall f\ s.\ \exists t.\ \forall y.\ y \in t \ = \ \exists x \in s.\ y = f\ x$
  - worried about relation to standard ZF in FOL
- But I still think set theory should be the long term aim!

I tried to implement set theory but failed!

- Tried to engineer too slick HOL shallow embedding
  - needed to trust complex ML encoding polymorphism
  - e.g. $I : \alpha \rightarrow \alpha$ vs. $(\lambda x : \mathrm{ST}.\ x)$ vs. $\{(x, x) \mid x \in \alpha\} : \mathrm{ST}$
- Tempted by elegance of ZF axioms in HOL
  - e.g. replacement: $\forall f\ s.\ \exists t.\ \forall y.\ y \in t\ =\ \exists x \in s.\ y = f\ x$
  - worried about relation to standard ZF in FOL
- But I still think set theory should be the long term aim!

I tried to implement set theory but failed!

- Tried to engineer too slick HOL shallow embedding
  - needed to trust complex ML encoding polymorphism
  - e.g. $I : \alpha \to \alpha$ vs. $(\lambda x : \mathrm{ST}.\, x)$ vs. $\{(x,x) \mid x \in \alpha\} : \mathrm{ST}$
- Tempted by elegance of ZF axioms in HOL
  - e.g. replacement: $\forall f\ s.\ \exists t.\ \forall y.\ y \in t \;=\; \exists x \in s.\ y = f\ x$
  - worried about relation to standard ZF in FOL
- But I still think set theory should be the long term aim!

One mathematics, many tools

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

One mathematics, many tools

- Logic programming reborn
  - theorems provers are the the new IDE
  - ACL2 almost as fast as C, others not far behind
  - immediate application to new generation of verifiers
- Beyond Church
  - HOL2P, HOL-Omega
  - set theory
- One mathematics, many tools
  - provers linked
  - most ordinary mathematics machine checked
  - tool-independent library of trustable theorems (QED)

Towards compatible proof systems

- Most of the world does mathematics in set theory
  - that's the official story
  - reality unclear
- Coq might appear an exception
  - but used for classical Four Color theorem, elliptic curves
  - Coq + axioms handles classical non-constructive theorems
- Slurping theorems from tool A into tool B impossible today!
  - even moving between Isabelle/HOL and other HOLs is hard
  - worse: sharing developments in HOL, PVS, Coq, ACL2
- Need a *lingua franca* for formalised mathematics
  - set theorem seems to me the only reasonable choice!

Towards compatible proof systems

- Most of the world does mathematics in set theory
  - that's the official story
  - reality unclear
- Coq might appear an exception
  - but used for classical Four Color theorem, elliptic curves
  - Coq + axioms handles classical non-constructive theorems
- Slurping theorems from tool A into tool B impossible today!
  - even moving between Isabelle/HOL and other HOLs is hard
  - worse: sharing developments in HOL, PVS, Coq, ACL2
- Need a *lingua franca* for formalised mathematics
  - set theorem seems to me the only reasonable choice!

Towards compatible proof systems

- Most of the world does mathematics in set theory
  - that’s the official story
  - reality unclear
- Coq might appear an exception
  - but used for classical Four Color theorem, elliptic curves
  - Coq + axioms handles classical non-constructive theorems
- Slurping theorems from tool A into tool B impossible today!
  - even moving between Isabelle/HOL and other HOLs is hard
  - worse: sharing developments in HOL, PVS, Coq, ACL2
- Need a *lingua franca* for formalised mathematics
  - set theorem seems to me the only reasonable choice!

Towards compatible proof systems

- Most of the world does mathematics in set theory
  - that's the official story
  - reality unclear
- Coq might appear an exception
  - but used for classical Four Color theorem, elliptic curves
  - Coq + axioms handles classical non-constructive theorems
- Slurping theorems from tool A into tool B impossible today!
  - even moving between Isabelle/HOL and other HOLs is hard
  - worse: sharing developments in HOL, PVS, Coq, ACL2
- Need a *lingua franca* for formalised mathematics
  - set theorem seems to me the only reasonable choice!

Towards compatible proof systems

- Most of the world does mathematics in set theory
  - that's the official story
  - reality unclear
- Coq might appear an exception
  - but used for classical Four Color theorem, elliptic curves
  - Coq + axioms handles classical non-constructive theorems
- Slurping theorems from tool A into tool B impossible today!
  - even moving between Isabelle/HOL and other HOLs is hard
  - worse: sharing developments in HOL, PVS, Coq, ACL2
- Need a *lingua franca* for formalised mathematics
  - set theorem seems to me the only reasonable choice!

Challenges for a mathematical lingua franca

- Need a concrete sytax
  - easy to parse and print
- Need method of storing proofs
  - hard to make this tool independent
- Need proof of concept
  - maybe one pioneer enough!
- Eventually need "buy in" from main tool developers
  - the hardest challenge of all!

Challenges for a mathematical lingua franca

- Need a concrete sytax
    - easy to parse and print
- Need method of storing proofs
    - hard to make this tool independent
- Need proof of concept
    - maybe one pioneer enough!
- Eventually need “buy in” from main tool developers
    - the hardest challenge of all!

Challenges for a mathematical lingua franca

- Need a concrete sytax
  - easy to parse and print
- Need method of storing proofs
  - hard to make this tool independent
- Need proof of concept
  - maybe one pioneer enough!
- Eventually need “buy in” from main tool developers
  - the hardest challenge of all!

Challenges for a mathematical lingua franca

- Need a concrete sytax
  - easy to parse and print
- Need method of storing proofs
  - hard to make this tool independent
- Need proof of concept
  - maybe one pioneer enough!
- Eventually need “buy in” from main tool developers
  - the hardest challenge of all!

Challenges for a mathematical lingua franca

- Need a concrete sytax
  - easy to parse and print
- Need method of storing proofs
  - hard to make this tool independent
- Need proof of concept
  - maybe one pioneer enough!
- Eventually need “buy in” from main tool developers
  - the hardest challenge of all!

Summary

- Logic programming reborn
  - proof scripting in theorem provers
- Beyond Church
  - HOL2P, HOL-Omega, ST
- One mathematics, many tools
  - provers linked via universal library of theories

Summary

- Beyond Church
  - HOL2P, HOL-Omega, ST

Summary

- One mathematics, many tools
  - provers linked via universal library of theories

Summary

- Logic programming reborn
    - proof scripting in theorem provers
- Beyond Church
    - HOL2P, HOL-Omega, ST
- One mathematics, many tools
    - provers linked via universal library of theories

Summary

- Logic programming reborn
  - proof scripting in theorem provers
- Beyond Church
  - HOL2P, HOL-Omega, ST
- One mathematics, many tools
  - provers linked via universal library of theories

THE END