The design of a programming language for provably correct programs: success and failure

Don Sannella

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh
http://homepages.inf.ed.ac.uk/dts

Outline

The Standard ML functional programming language

- Origins
- Design and features
- Semantics

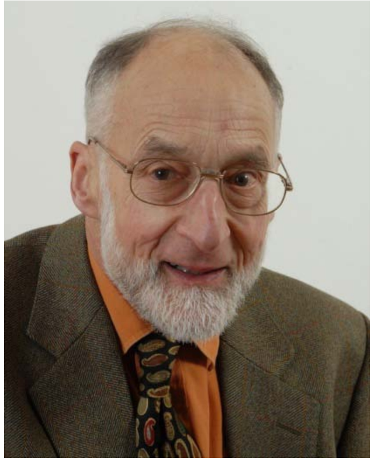
The Extended ML framework for specification and development of modular Standard ML software systems

An application of program proof: security certification

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

- For programming proof search strategies
  `s : goal -> (goal list * (thm list -> thm))`
- Higher order functions for strategy-building combinators
- Exception mechanism for backtracking
- Thm as an abstract data type, with the inference rules as its only constructors
- Polymorphism
- Interactive



Hope (Burstall, 1980)

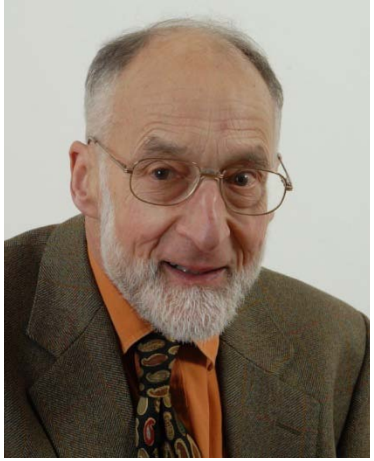
Algebraic specification (Burstall/Goguen)

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

- For programming proof search strategies
- Higher order functions for strategy-building combinators

  `s : goal -> (goal list * (thm list -> thm))`
- Exception mechanism for backtracking
- Thm as an abstract data type, with the inference rules as its only constructors
- Polymorphism
- Interactive



Hope (Burstall, 1980)

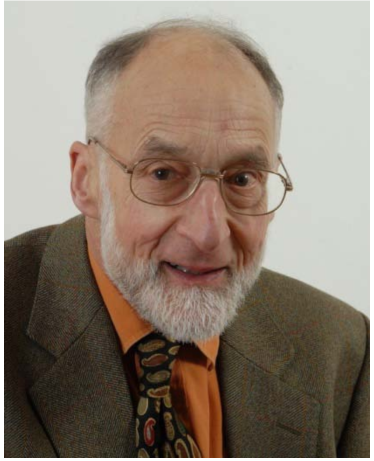
Algebraic specification (Burstall/Goguen)

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

- For programming proof search strategies
- Higher order functions for strategy-building combinators
- Exception mechanism for backtracking

  `REPEAT (x ORELSE y) THEN z`
- Thm as an abstract data type, with the inference rules as its only constructors
- Polymorphism
- Interactive



Hope (Burstall, 1980)

Algebraic specification (Burstall/Goguen)

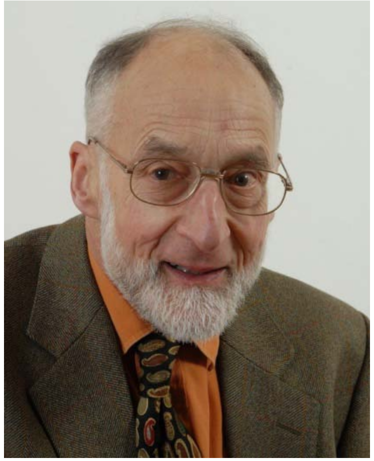
Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

- For programming proof search strategies
- Higher order functions for strategy-building combinators
- Exception mechanism for backtracking
- Thm as an abstract data type, with the inference rules as its only constructors

```
MP : thm * thm -> thm
```

- Polymorphism
- Interactive



Hope (Burstall, 1980)

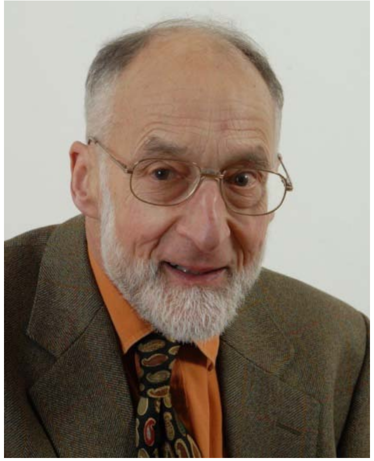
Algebraic specification (Burstall/Goguen)

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

- For programming proof search strategies
- Higher order functions for strategy-building combinators
- Exception mechanism for backtracking
- Thm as an abstract data type, with the inference rules as its only constructors
- Polymorphism

  `reverse : α list -> α list`
- Interactive



Hope (Burstall, 1980)

Algebraic specification (Burstall/Goguen)

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

Hope (Burstall, 1980)

```
datatype α tree = empty
                | node of α tree * α * α tree

fun flatten empty = []
  | flatten(node(t1,x,t2)) =
        (flatten t1) @ (x :: (flatten t2))
```

Algebraic specification (Burstall/Goguen)

Standard ML: Origins

Meta-language of LCF theorem prover (Milner, 1978)

Hope (Burstall, 1980)

Algebraic specification (Burstall/Goguen)

- Parameterised modules
- Interfaces and module bodies are separate
- Pushout-style application
- Stratification between code level and module level

Standard ML: Design and features

Design by committee with strong leadership (1983-1987)

- Mainly Edinburgh, plus Dave MacQueen
- Led by Robin Milner

Core language

Module language

Standard ML: Design and features

Design by committee with strong leadership (1983-1987)

Core language

- ML's features
- Hope's algebraic data types
- Cardelli's labelled records
- generalised exceptions
- generalised references
- call-by-value

Module language

Standard ML: Design and features

Design by committee with strong leadership (1983-1987)

Core language

Module language (Dave MacQueen)

- Explicit interfaces ("signatures")
- Software components ("structures")
- Generic components ("functors")
- Shared sub-components with explicit sharing declarations

Standard ML: Language definition (1990)

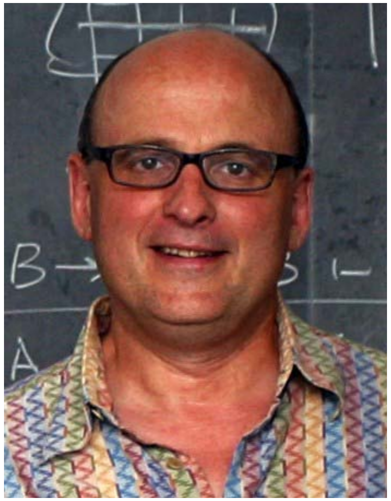
Syntax – 21 pages

- full “bare” syntax in 2.5 pages

Static semantics (type rules) – 30 pages

Dynamic semantics (evaluation rules) – 17 pages

Commentary (1991)



Standard ML: Language definition (1990)

Syntax – 21 pages

Static semantics (type rules) – 30 pages

$$\frac{C \vdash exp : \tau' \rightarrow \tau \qquad C \vdash exp' : \tau'}{C \vdash exp \;\; exp' : \tau}$$

Dynamic semantics (evaluation rules) – 17 pages

Commentary (1991)

Standard ML: Language definition (1990)

Syntax – 21 pages

Static semantics (type rules) – 30 pages

Dynamic semantics (evaluation rules) – 17 pages

$$\frac{E \vdash \mathrm{dec} \blacktriangleright E' \qquad E + E' \vdash \mathrm{exp} \blacktriangleright v}{E \vdash \mathbf{let}\ \mathrm{dec}\ \mathbf{in}\ \mathrm{exp}\ \mathbf{end} \blacktriangleright v}$$

Commentary (1991)

Standard ML: Language definition (1990)

Syntax – 21 pages

Static semantics (type rules) – 30 pages

Dynamic semantics (evaluation rules) – 17 pages

Commentary (1991)

- Explanation of the semantics
- Theorems about the language, e.g. deterministic evaluation, type soundness, existence of principal types

Outline

The Standard ML functional programming language

The Extended ML framework for specification and development of modular Standard ML software systems

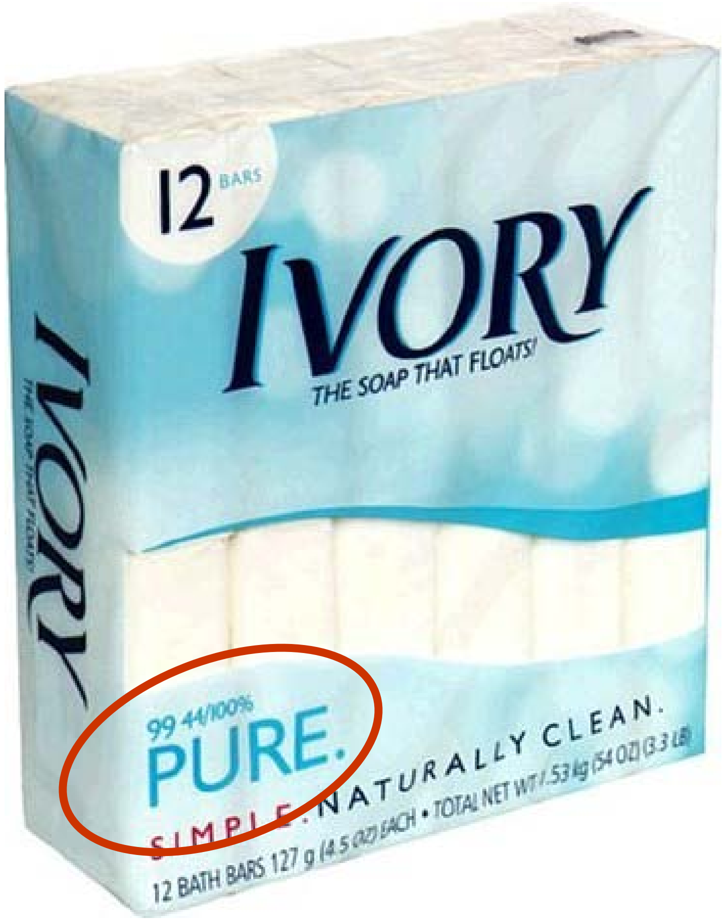
- Motivation
- Design
- Theory
- Semantics
- Proof
- Tools
- Failure
- Post mortem

An application of program proof: security certification

Extended ML: Motivation (1985)

Pure functional programming allows straightforward proofs of properties because of referential transparency

- Equational reasoning
- Structural induction
- Standard ML is not pure, but almost



Extended ML: Motivation (1985)

Pure functional programming allows straightforward proofs of properties because of referential transparency

Algebraic specification theory (Sannella/Tarlecki et al)

- algebraic models
- axiomatic specifications
- specification structure
- proof of consequences
- stepwise refinement
- information hiding
- parameterisation
- behavioural equivalence
- independence from logical system

Extended ML: Motivation (1985)

Pure functional programming allows straightforward proofs of properties because of referential transparency

Algebraic specification theory (Sannella/Tarlecki et al)

Standard ML language definition provides a basis for establishing soundness

Extended ML: Design (1985)

Minimal extension of Standard ML

- Axioms in first-order logic with equality
- Placeholder for expressions and types that haven't been written yet

Extended ML: Design (1985)

Minimal extension of Standard ML

A “wide spectrum” language

- Covering specifications, programs, and intermediate stages of development

Extended ML: Design (1985)

Minimal extension of Standard ML

A “wide spectrum” language

Simple and intuitive for ML programmers

Leave out references: too hard

Otherwise stick with full Standard ML

Extended ML: Design (1985)

Axioms are just boolean expressions containing extra constants

- **`forall`** x:t **=>** expr
- **`exists`** x:t **=>** expr
- expr **==** expr’
- expr **`terminates`**
- expr **`raises`** exn

Extended ML: Design (1986-1990)

Axioms are just boolean expressions containing extra constants

Hard problem: interactions between features

- polymorphism
- quantification
- equality
- abstraction boundaries
- exceptions and non-termination

Extended ML: Design (1986-1990)

Axioms are just boolean expressions containing extra constants

Hard problem: interactions between features

Looked for solution that is natural for ML programmers

Example: quantification over a polymorphic type

- **`forall (x,xs) => [x]@xs == xs@[x]`**
- ... looks like it should be false
- ... but it is polymorphic – with types we have

  `forall (x:α,xs:α list) => [x]@xs == xs@[x]`
- true if **α** is **`unit`**, false otherwise!
- ... so it is taken to have no meaning
- **`forall xs => exists ys => xs@ys == ys@xs`**

  is true, because y=**`[]`** satisfies it

Extended ML: Design (1986-1990)

Axioms are just boolean expressions containing extra constants

Hard problem: interactions between features

Looked for solution that is natural for ML programmers

Example: quantification over a polymorphic type

- Easy solution: require explicit quantification over type variables
- But ML has implicit polymorphism!

Extended ML: Theory (1985-1995)

Lots of very interesting problems to do with modules

Methodology for formal development of modular software systems by stepwise refinement and decomposition

Theory is independent of language used for axioms and language used for coding “in the small”

- Experiments with Prolog
- Experiments with knowledge representation language

Extended ML: Theory (1985-1995)

Lots of very interesting problems to do with modules

Methodology for formal development of modular software systems by stepwise refinement and decomposition

Theory is independent of language used for axioms and language used for coding “in the small”

Drove development of theory of algebraic specification

- behavioural equivalence
- stable constructions
- parameterisation
- implementation of specifications
- institution-independent language definitions

Extended ML: Theory (1985-1995)

Lots of very interesting problems to do with modules

Methodology for formal development of modular software systems by stepwise refinement and decomposition

Theory is independent of language used for axioms and language used for coding “in the small”

Drove development of theory of algebraic specification

Very productive synergy with algebraic specification work

Extended ML: Semantics (1992-1996)

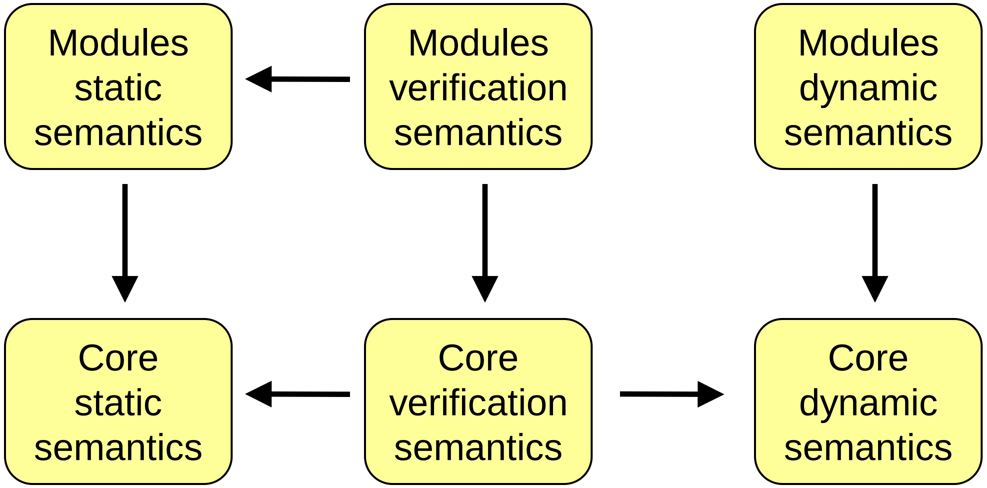
Determined to build on top of Standard ML semantics

- Static semantics
- Dynamic semantics
- Verification semantics
- Dependencies more complex than before

Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

- Static semantics
- Dynamic semantics
- Verification semantics
- Dependencies more complex than before



Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

- Static semantics
- Dynamic semantics
- Verification semantics
- Dependencies more complex than before
- Result was 140 pages
- Found some errors in the Standard ML semantics

Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

More hard problems

Example: domain of quantification for function types

Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

More hard problems

Example: domain of quantification for function types

- Set-theoretic functions?
- Computable functions?
- Function space in a model of parametric polymorphism?
- Expressible functions?
- But what does “expressible” mean exactly?

Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

More hard problems

Example: domain of quantification for function types

Very complex rules

$$\frac{\mathrm{Comp}(FE, s) = VE \qquad \dfrac{\begin{array}{c}\gamma \succ \gamma_1 = (C, \tau)\cdot\gamma_2 \quad s^{\#\#}(C) + \mathrm{Stat}\, VE \vdash_{\mathrm{STAT}} atexp^{\bullet} \Rightarrow \tau', \emptyset, \gamma_3 \\ s^{\#\#}(\tau) = \tau' \quad \mathrm{Dyn}(s, FE + VE) \vdash_{\mathrm{DYN}} atexp^{\bullet} \Rightarrow v_{\mathrm{DYN}}, (\top, ens)\end{array}}{\exists s'.s, (FE + VE, \gamma_1 \cdot \gamma_3) \vdash (\mathtt{fn}\, x \Rightarrow exp^{\bullet})\, atexp^{\bullet} \Rightarrow \mathtt{true}, s'}}{s, (FE, \gamma) \vdash \mathtt{forall}\, x \,\texttt{=>}\, exp^{\bullet} \Rightarrow \mathtt{true}, s}$$

Extended ML: Semantics (1992-1996)

Determined to build on top of Standard ML semantics

More hard problems

Example: domain of quantification for function types

Very complex rules

New version of Standard ML language definition (1997)

- ... time to start again?

Extended ML: Proof (1994-1996)

For first-order monomorphic functions that always terminate and never raise exceptions, everything is as expected

- Otherwise, it's a nightmare

Extended ML: Proof (1994-1996)

For first-order monomorphic functions that always terminate and never raise exceptions, everything is as expected

Multi-valued logic because we used boolean expressions as axioms, and boolean expressions can raise exceptions

- So logical connectives sometimes behave strangely

Extended ML: Proof (1994-1996)

For first-order monomorphic functions that always terminate and never raise exceptions, everything is as expected

Multi-valued logic because we used boolean expressions as axioms, and boolean expressions can raise exceptions

Reasoning about exceptions is intractable (Pitts/Stark 1993)

- So equality isn't even reflexive (expr **==** expr is not always true)

Extended ML: Proof (1994-1996)

For first-order monomorphic functions that always terminate and never raise exceptions, everything is as expected

Multi-valued logic because we used boolean expressions as axioms, and boolean expressions can raise exceptions

Reasoning about exceptions is intractable (Pitts/Stark 1993)

Specifying higher-order functions is messy: functional arguments typically need to be specified to always terminate and never raise exceptions

- Likewise for higher-order functional arguments, provided *their* functional arguments do so

Extended ML: Proof (1994-1996)

For first-order monomorphic functions that always terminate and never raise exceptions, everything is as expected

Multi-valued logic because we used boolean expressions as axioms, and boolean expressions can raise exceptions

Reasoning about exceptions is intractable (Pitts/Stark 1993)

Specifying higher-order functions is messy: functional arguments typically need to be specified to always terminate and never raise exceptions

We gave up

Extended ML: Tools (1992-2001)

Parsers and typecheckers

Proof obligation generator (prototype)

Limited proof support by translation into PVS (prototype)

Extended ML: Failure

Very good for teaching formal methods to students who know Standard ML already

Otherwise, not enough user interest

- Specifications are too hard to write
- Formal development of modular programs from specifications is possible, but a lot of work
- Proving correctness of single-threaded functional programs is too much work for too little payoff

Proof is intractable

Extended ML: Post mortem

We were too ambitious

There are features of ML that are hard to handle in isolation

- But nobody really knew that at the time

Extended ML: Post mortem

We were too ambitious

There are features of ML that are hard to handle in isolation

... and they are much harder to handle in combination

Doing it formally for a “real” language was very hard

- But I still believe in that goal

Extended ML: Post mortem

We were too ambitious

There are features of ML that are hard to handle in isolation

... and they are much harder to handle in combination

Doing it formally for a “real” language was very hard

Doing design and semantics long before proof and tools was a big mistake

Correctness of pure functional programs is not a problem in practice

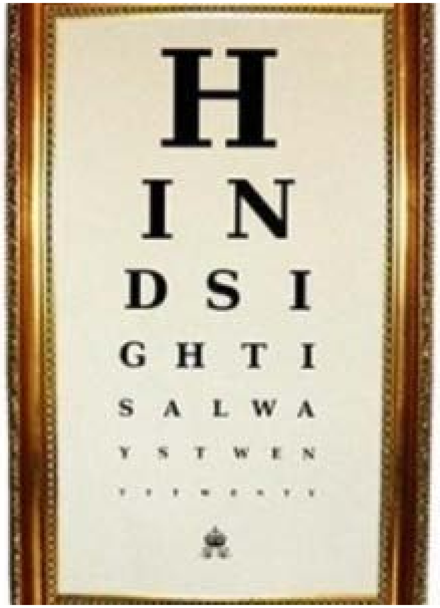
Extended ML: Alternatives

Start with a small subset, do semantics, proofs and tools for that

Add a feature and iterate

Stop when the next iteration is too hard

Attractive starting point: Moggi's computational lambda calculus (1989)



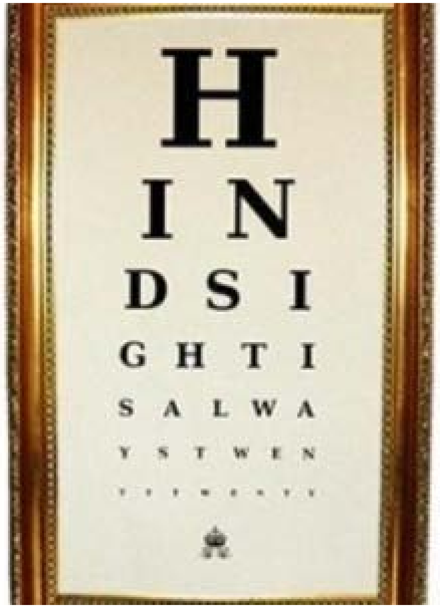
Extended ML: Alternatives

Forget proofs, focus on specification-based testing

Testing as a useful approximation to proof

Sometimes it is even as good as proof

Axioms as an aid to programming productivity



Extended ML: Alternatives

Be much less ambitious about the kinds of properties to be proved

Focus on properties that people care about

... and situations where having a proof of that property is valuable

Security certification!

Outline

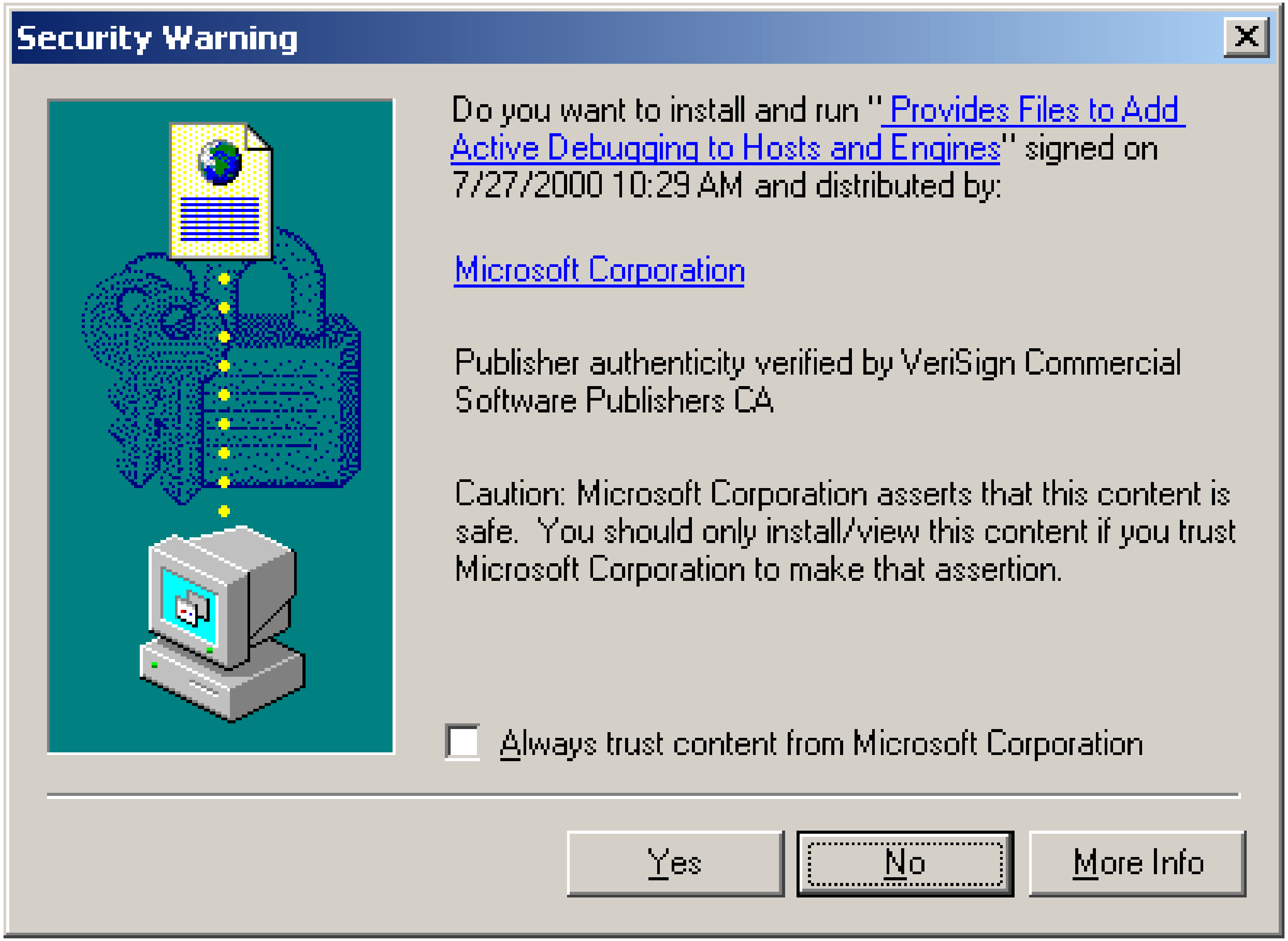
The Standard ML functional programming language

The Extended ML framework for specification and development of modular Standard ML software systems

An application of program proof: security certification

- Proof-carrying code
- Evidence-based certification

In Microsoft I trust



Microsoft security bulletin MS01-017

Who should read this bulletin: All customers using Microsoft® products.

Technical description: In mid-March 2001, VeriSign, Inc., advised Microsoft that on January 29 and 30, 2001, it issued two VeriSign Class 3 code-signing digital certificates to an individual who fraudulently claimed to be a Microsoft employee. …

Impact of vulnerability: Attacker could digitally sign code using the name "Microsoft Corporation".

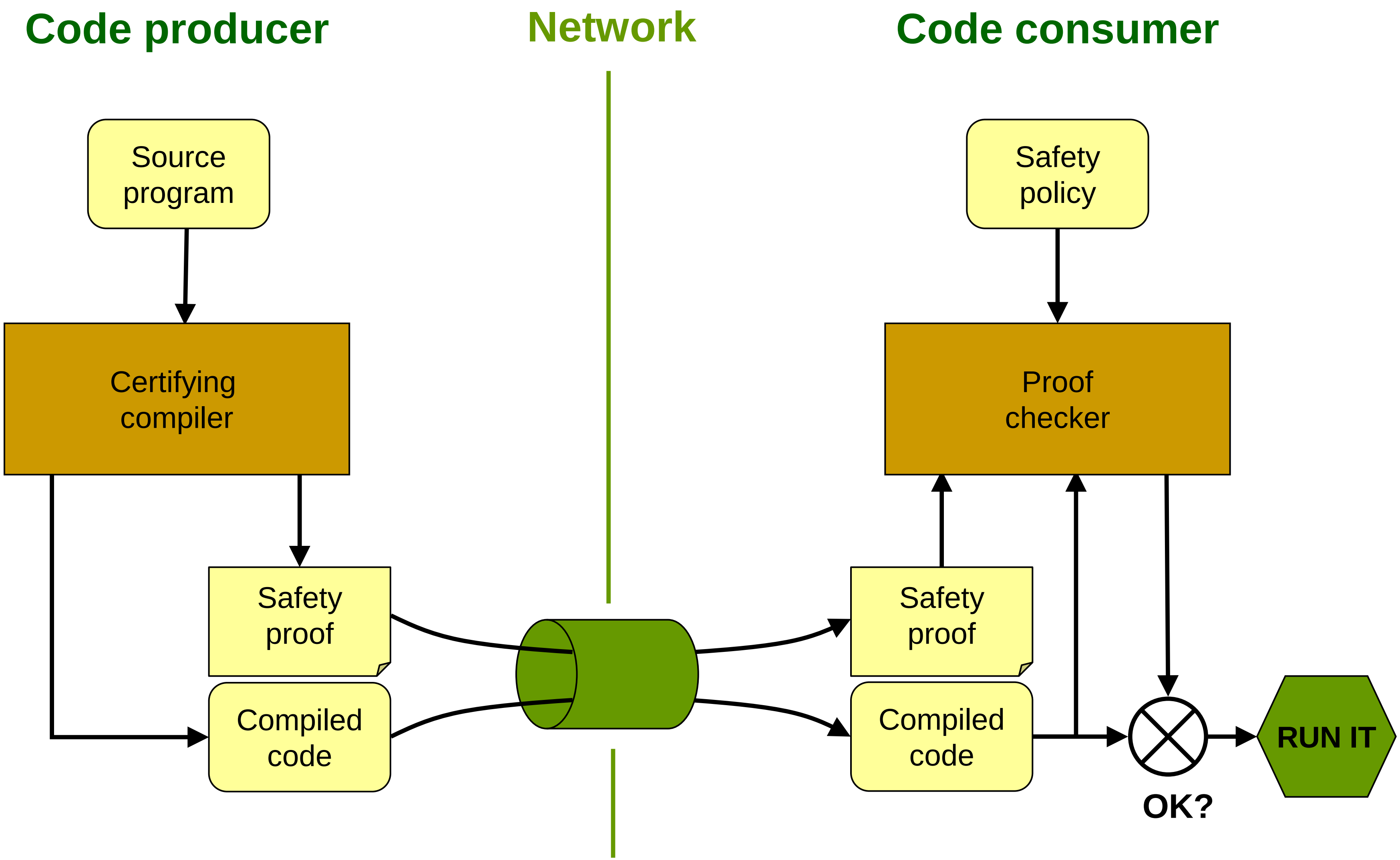
Proof-carrying code (Necula, 1997)

PCC certifies code with a condensed formal proof of a desired property.

- Checked by client before installation / execution
- Proofs may be hard to generate, but are easy to check
- Independent of trust networks: unforgeable, tamper-evident

A *certifying compiler* uses types and other high-level source information to create the necessary proof to accompany machine code.

PCC architecture



Space Types (MRG project, 2005)

```
let insert n l d =
  match l with Nil -> Cons(n,Nil)@d
        | Cons(h,t)@d' -> if n <= h
                          then Cons(n,Cons(h,t)@d')@d
                          else Cons(h,insert n t d)@d'
let sort l =
  match l with Nil -> Nil
        | Cons(h,t)@d -> insert h (sort t) d
```

```
insert: int * intlist * <> -> intlist
sort  : intlist –> intlist
```

Types and annotations can be inferred using a separate linear constraint solver, and proofs can be generated from type derivations

More generally: Evidence-based Security

PCC certifies code with a condensed formal proof of a desired property.

- Checked by client before installation / execution
- Proofs may be hard to generate, but are easy to check
- Independent of trust networks: unforgeable, tamper-evident

Evidence-based security is about certifying code with checkable evidence of a desired property.

Proof-carrying code is just one example.

Some forms of evidence provide weaker guarantees than proof.

Conclusion

Things are sometimes a lot harder than they appear

Doing theory and practice hand-in-hand is important for both

Times change and new applications can build on old work

This is a fruitful area for research and experimentation

Mobius

MRG Mobile Resource Guarantees