# HOL4: State of the System

[Trindemossen-1]

Michael Norrish

June 2024



# Outline

# “Defining a Niche for HOL4”

HOL4 Workshop in Berlin at CADE’2015.

I got to talk about “niches”.

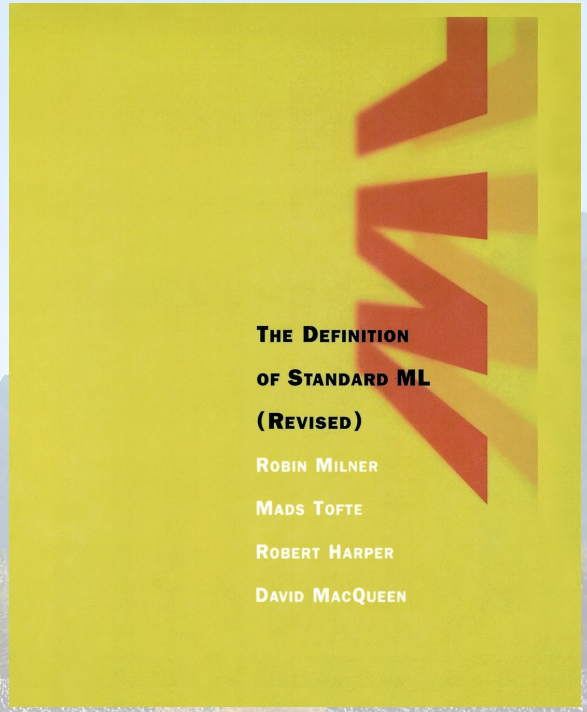
(More?) Importantly, also identified strengths and weaknesses.

[Lake Kananaskis by davebloggs007@hotmail.com *via* Flickr]

# SML as a Strength

Well-defined language.

Clean semantics.




Has the features the implementor wants:

- type system
- exceptions
- even concurrency (in Poly/ML)

# HOL as a Strength

Well-understood logical *lingua franca*:

- for users;
- for systems (*e.g.*, OpenTheory)

Also: a Lowest Common Denominator

# Strengths (Accidental/Historical)

Existing Formalisations

- CakeML, hardware models, ...

Existing Users

Documentation

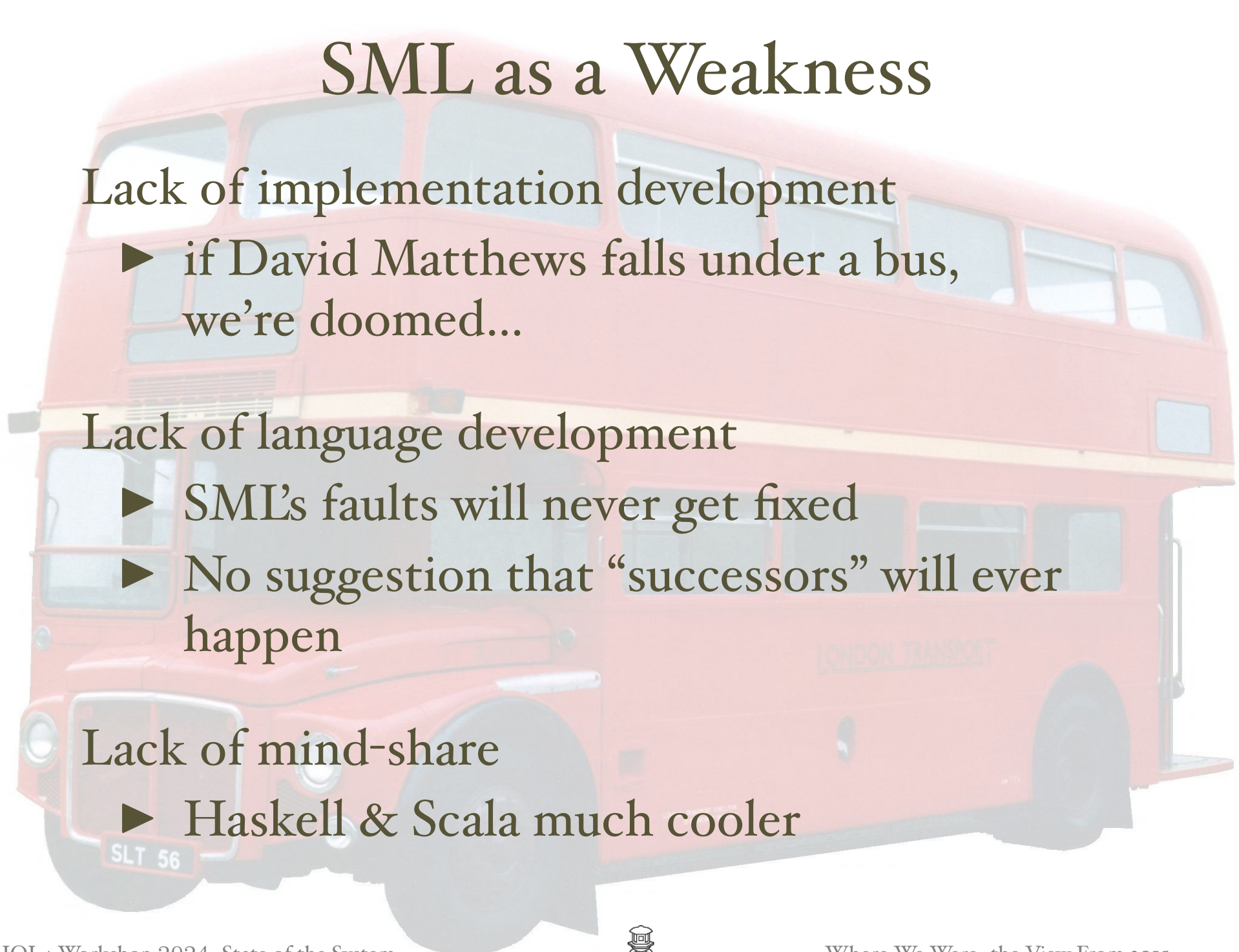
Minimal code churn

- caused by slow development...

# Weaknesses

- SML
- HOL
- Windows
- User Interface
- Persistent Theories as Code
- Script-files as Code
- Lack of Concurrency

# SML as a Weakness




Lack of implementation development

- if David Matthews falls under a bus, we're doomed...

Lack of language development

- SML's faults will never get fixed
- No suggestion that "successors" will ever happen

Lack of mind-share

- Haskell & Scala much cooler

# HOL as a Weakness

HOL doesn't have cool types.

Not even Isabelle/HOL's type-classes.

- And lacking constants with different definitions on different types fundamentally blocks some constructions

# User Interface

The `emacs` mode is hobbled by script files as code.

- Some would swear by `emacs` as an IDE
- ...but probably not for SML

Maybe proofs need different editing tools compared to code.

# HOL Now

Where We Were: the View From 2015

Where We Are

- Updates Since 2015
- Kernels, Tools, Theories, UI
- Community

Where We Might Go

- Kernel
- Tools
- UI

Conclusion

# Strengths Remain

**SML**:

- Language is just as well-defined now as it was 9 years ago
- Continued successes of CakeML mean we have excellent extraction story

**HOL**:

- Candle means our self-"verification" story is better than ever;
- principled `cv` story in Trindemossen

# Strengths Remain

**CakeML**:

- Clearly an ongoing success;
- Attracts students, collaborators, new variations, projects

**Hardware**:

- Historic strength; continues to be built upon

**Documentation/Lack of Churn**:

- Lack of drastic change keeps old docs. good

# Weaknesses Addressed

**Code** *vs.* **Data**:

- Theory files now better: `.dat` format; `LoadableThyData` abstraction
- Script files improved *via* "modern syntax"

**Concurrency**:

- `Holmake` got increasingly parallel from 2016 onwards

**Windows**:

- WSL reliable enough for Poly/ML; no longer an issue IMO

# Weaknesses Addressed

**User Interface**:

- Better.
- Emacs and `vim` modes have both received love
- Experiments with VSCode

- More yet to say here...

# Outline





# Kernels

HOL4 has 3 kernels:

`stdknl` The "standard" kernel (de Bruijn indices); good support for `EVAL`

# Kernels

HOL4 has 3 kernels:

`stdknl` The "standard" kernel (de Bruijn indices); good support for `EVAL`

`expknl` The "experimental" kernel (name-carrying); constant time `dest_abs`; slower `EVAL`; simpler



# Kernels

HOL4 has 3 kernels:

`stdknl` The "standard" kernel (de Bruijn indices); good support for `EVAL`

`expknl` The "experimental" kernel (name-carrying); constant time `dest_abs`; slower `EVAL`; simpler

`otknl` The "logging", OpenTheory kernel. Slow build caused by calls to `opentheory` tool.

# Trindemossen's Headline Feature:

## `cv_compute`

Has reduced build time for the in-logic compilation/bootstrapping from

# Trindemossen's Headline Feature:

## `cv_compute`

Has reduced build time for the in-logic compilation/bootstrapping from >12 hours down to

# Trindemossen's Headline Feature:

## `cv_compute`

Has reduced build time for the in-logic compilation/bootstrapping from

>12 hours down to

<1 hour

# `cv_compute` Conceptually

Builds on ground data ("`cv` terms") constructed of

- numbers
- pairs of `cv` terms

S-Expressions, in other words.

# `cv_compute` Conceptually

Some primitives:

- arithmetic: $+$, $-$, $<$, ...
- pairs: ispair, fst, snd
- if

User provides set of HOL function definitions:

$$f_i\ vs_i \quad = \quad rhs_i$$

Possibly mutually recursive ($rhs_i$ may also feature `let`s).

# `cv_compute` Conceptually

With extra tools (`cv_translate`)

- can encode "normal" HOL functions over many flavours of types (encoded as S-expressions) and get high-speed execution

# `cv_compute` and the HOL Kernels

Technology works in standard and experimental kernels.

Less clear how to log for `--otknl`'s benefit.

Could:

- calculate and emit (huge) rewrites as logging is done;
- assert OpenTheory axioms and let someone else figure it out;
- change OpenTheory to support `cv`



# Terms No Longer an Equality Type

Clearly appropriate in presence of

- $\alpha$-equivalence, and
- `stdknl`'s suspended substitutions

Added $t_1 \sim\!\!\sim t_2$ for `aconv` $t_1$ $t_2$

May affect old code.

Types can safely remain an equality type.

# Theory Format on Disk Changed

Bulk of theory information now uniformly stored as S-expressions

API exists to allow users to store "arbitrary" data there in standardised way

Done by system for:

- Grammars
- Simplifier changes
- TypeBase changes
- Recording induction principles
- `cv_translate` data, and much else

# HOL Tools 2024: `Holmake`

- Concurrent execution across all scrips and directories of a project
  - Scheduling is non-deterministic
    (*i.e.*, not optimised)
  - Needs careful (but fairly obvious) specification of `INCLUDES` lines
- Theory files (`.uo`, `.ui`, `.dat`) stashed out of sight


# HOL Tools 2024: Simplifier

Simplifier is fundamentally the same as ever.

Important new features over 9 years:

- `gv`, `gvs`: Almost always better than `fs`
- `SF` form adds fragments inside argument lists, e.g., `simp[SF DNF_ss]` (= `dsimp[]`).
- More and more ways to exclude things from simplification

# HOL Tools 2024: Simplifier Exclusions

**Tactic level:** `gs[Excl "APPEND_ASSOC"]`

**Proof level:** `Proof[exclude_simps = FACT]`

**Script level:** `temp_delsimps["FACT"]`

Script-level exclusion can be made permanent/inherited.



# HOL Tools 2024: Simplifier Normalisation

Multiplicative terms over $\mathbb{R}$, and (in)equalities between are aggressively simplified.

For example `x/2 = y/z` simplifies to

```
x * z = 2 * y
```

when simplifier can show `z≠0`



# HOL Tools 2024: Tactics

- General purpose resolution/unification tactic with back-tracking
- Encouragement of good name hygiene: renaming `rename[‘f x + _’]` and sub-goal selection by pattern `>~`

# HOL Tools 2024: Modern Syntax

```
Theorem P0_11[local,simp]:
  P0 c d = P0 e f <=> c = e /\ d = f
Proof
  simp[FUN_EQ_THM, P0_def, EQ_IMP_THM] >> rpt strip_tac
  >- (Q.RENAME_TAC [‘c x = e x’] >>
      pop_assum $ Q.SPEC_THEN ‘2 * x + 1’ mp_tac >>
      simp[ADD_EQ_0, ADD_CLAUSES, ODD_ADD, ODD_MULT]) >>
  Q.RENAME_TAC [‘d x = f x’] >>
  pop_assum $ Q.SPEC_THEN ‘2 * x + 2’ mp_tac >>
  simp[ADD_EQ_0, ADD_CLAUSES, ODD_ADD, ODD_MULT, MULT_EQ_0, ADD_SUB]
QED

Inductive iscv:
[~num:] (!m. iscv (N0 m))
[~pair:] (!c d. iscv c /\ iscv d ==> iscv (P0 c d))
End
```

Hides `store_thm` code smell;
pretends that only SML comes in the tactics

- Still under-documented

# HOL Theories '24: New Examples

By Chun Tian:

- revives & extends Monica Nesi's HOL88 CCS example
- ports and develops probability work

Hing-Lun (Joseph) Chan:

- abstract algebra up to/including finite fields
- correctness of AKS primality test

and Elliot Catt on Kolmogorov Complexity

# HOL Theories '24: Reorganisation

This is barely underway, but

I am keen to see examples interlock and build a common foundation.

Some steps towards this under `examples/logic`



# Continuing to Attract Contributions

Rate of incoming work slow enough
that editorial oversight is possible;
(particularly of student work)

# HOL UI

Modern syntax and learning more about our editor capabilities means our editor modes (`vim` and `emacs`) are better.

- *e.g.*, efficient "sexp" movement/selection in `emacs`.

Experiments with VSCode

# Community

CakeML Discord is a great resource

Seems reasonably discoverable

Github clearly also indispensable

# Outline

# Little Theories Reborn

It is easy to track the theorems that depend on a call to `new_axiom`

If the axiom has $1+$ models, why shouldn't we allow discharge/instantiation of axioms to derive new (axiom-less) theorems?

Needs validation in Candle or similar.



# Rewriting Primitives Generate Garbage

Rewriting `f`$^n$`(x + 0)` generates theorem

$$\vdash \texttt{f}^n\texttt{(x + 0) = f}^n\texttt{(x)}$$

Term on left of theorem is *not* the same as input term; it's a redundant copy.

`EVAL` does this right and shares structure.

# Lazy Theorems and Isabelle Style Contexts

Hasan Amjad's lazy theorems (2005) could be used to speed interactive evaluation of script files.

But proofs running concurrently in background need to capture part of their "context" (simplifier and grammar state at very least) and not depend on global references.

- Think change to type of `tactic` (!!!) can happen without breaking everything

# Really Exploiting `cv`

`cv_compute` gives us something that "smells" quite a bit like reflection in Coq.
Some decision-procedures might really improve (Cooper)

- (Others (`metis`) less so...)

# Better Datatypes

I am confident I can implement bounded natural functor style algebraic datatypes (as available in Isabelle/HOL).

This should make a number of things much nicer.

Co-algebraic types not so clear in my head yet.

# OpenTheory

Maybe someone will figure out prover interoperability for real.

# OpenTheory

Maybe someone will figure out prover interoperability for real.

[*Cue mad laughter*]

# User Interface

Classic SML tactics give great deal of "purely textual" power

- *e.g.*, `using` infix, `>~`, ...

I will continue to work on `emacs` mode for forseeable future because it's what I use myself.

I don't have time or expertise to put serious work into other modalities

# Conclusion

HOL is going from strength to strength.

It's far from perfect, but adds value to the user community's lives as researchers and FM engineers.

I hope we can collectively continue to make it yet more awesome (however slowly).