avoiding state with infinite contexts

Freek Wiedijk
Radboud University Nijmegen
The Netherlands

10th Asian Logic Conference
special session: model theory and proof theory
Kobe, Japan

2008 09 05, 15:40

trusting a proof checker

proof checking of mathematics

fully verified for correctness by a computer:

- Gödel's first incompleteness theorem

  nqthm (1986), Coq (2003), HOL Light (2005)

- Jordan curve theorem

  HOL Light (2005), Mizar (2005)

- prime number theorem

  Isabelle (2004), HOL Light (2008)

- four color theorem

  Coq (2004)

the de Bruijn criterion

all software has bugs . . .

why trust a proof checker?

N.G. de Bruijn, 1968:

> [. . .] This is one of the reasons for keeping AUTOMATH as primitive as possible. [. . .]

two approaches:

- small independent checker(s)
  e.g.: Ivy system for Otter/Prover9

- small proof checking **kernel** inside the system
  LCF architecture

  Robin Milner, 1972

systems and kernels

source sizes in 10^3 lines of code

		kernel	system	thms
HOL Light	ocaml	0.7	30	69
Isabelle	sml	5	160	40
Twelf	sml	6	70	—
ProofPower	sml	7	90	42
Coq	ocaml	14	180	39
Mizar	pascal		80	45
ACL2	lisp		170	12
PVS	lisp		280	15

proving the kernel correct

- Coq in Coq

  Bruno Barras, 1997–1999

  executable Coq model of the Coq logic

  two different systems (the real system versus the extracted code)
  extracted code not yet used for serious proof development

- HOL in HOL

  John Harrison, 2006

  HOL model of the actual HOL Light kernel source

  not yet the full code (no type polymorphism, no definitions)
  no *systematic* relation between HOL model and executable code

kernels and state

to make proving a kernel feasible:

- code should be as ‘mathematical’ as possible
- for current technology:

  kernel should be programmed in a purely functional language

  Lisp, ML, Haskell, Coq

current practice:

- kernels always have a **state:**

  definitions from the formalization that already have been processed
- corresponds to a context in the formal treatment of the logic

$$\Gamma \vdash M : A$$

undo for HOL

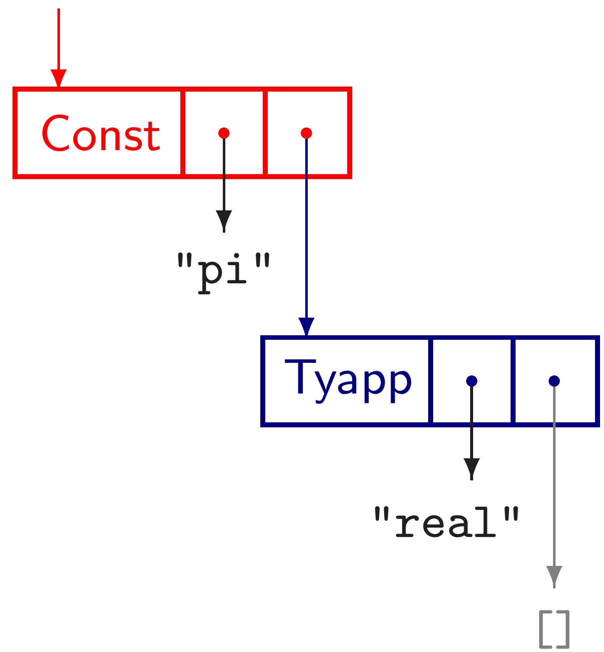
abstract datatypes of the HOL Light kernel

```
type hol_type = private
| Tyvar of string
| Tyapp of string * hol_type list

type term = private
| Var of string * hol_type
| Const of string * hol_type
| Comb of term * term
| Abs of term * term

type thm = private
| Sequent of term list * term
```

```
# `pi`;;
val it : term = `pi`
#
```



the problem with undoing definitions

hypothetical HOL Light session:

```
# let X0 = new_definition `X = 0`;;
val ( X0 ) : thm = |- X = 0
# undo_definition "X";;
val it : unit = ()
# let X1 = new_definition `X = 1`;;
val ( X1 ) : thm = |- X = 1
# TRANS (SYM X0) X1;;
val it : thm = |- 0 = 1
#
```

undoing definitions can change the meaning of existing `thms`

inconsistent!

$\Rightarrow$ *HOL Light does not support 'undo'*

putting the definitions in the names

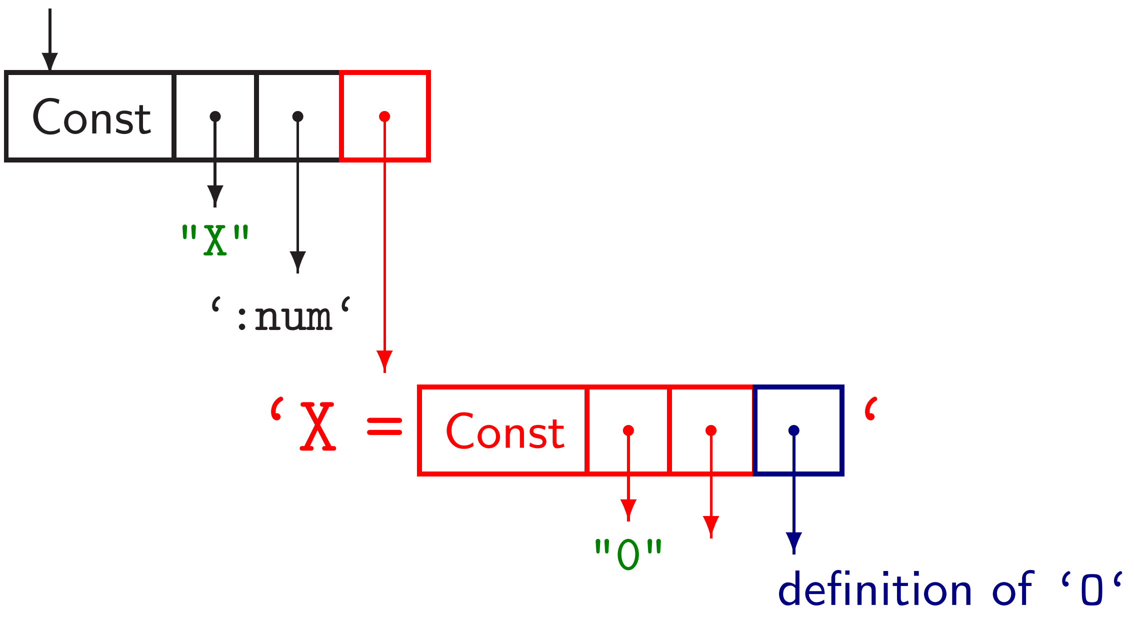
- **current HOL Light**

  names of constants: pair of a string and a type

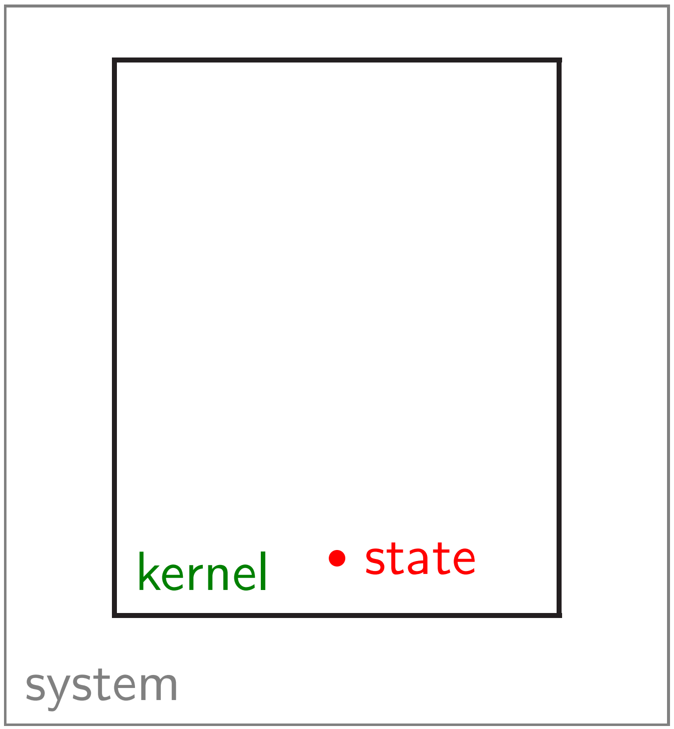
- **stateless HOL Light**

  names of constants: pair of the traditional name and the definition

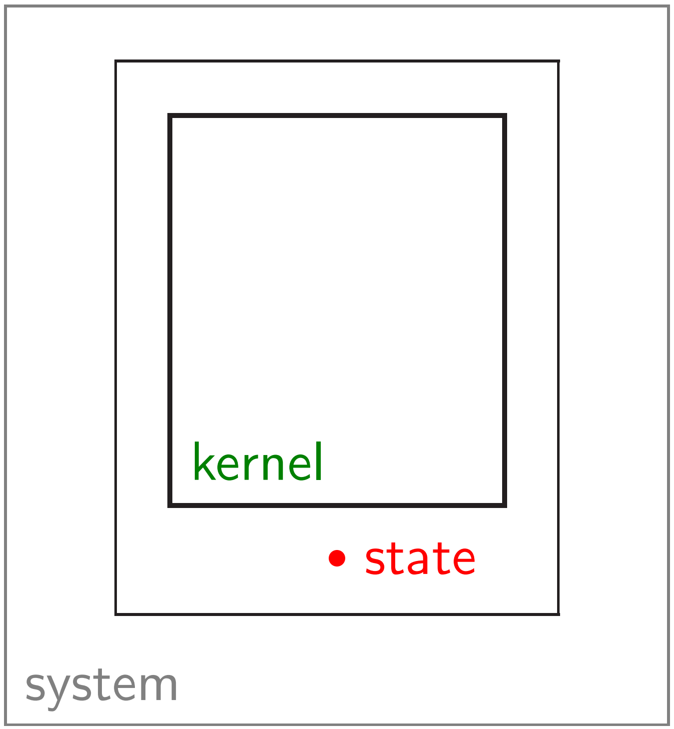
  comparing equal definitions by pointer comparison is cheap



current HOL Light



stateless HOL Light



about 10 % slower

type theory without explicit contexts

first order logic and contexts

first order logic

$$\dfrac{\dfrac{A \vdash P(x)}{A \vdash \forall x\, P(x)}\ \forall I}{\vdash A \rightarrow \forall x\, P(x)}\ {\rightarrow} I$$

'sea' of free variables

type theory

$$\dfrac{\dfrac{H : A\,,\ x : D \vdash M_1 : P(x)}{H : A \vdash M_2 : \Pi x : D.\, P(x)}\ \lambda}{\vdash M_3 : A \rightarrow \Pi x : D.\, P(x)}\ \lambda$$

'free' variables in the context

$$\dfrac{\dfrac{A \vdash_{\{x\}} P(x)}{A \vdash_{\{\}} \forall x\, P(x)}\ \forall I}{\vdash_{\{\}} A \rightarrow \forall x\, P(x)}\ {\rightarrow} I$$

$\vdash_{\{\}} \big(\forall x\, P(x)\big) \rightarrow \big(\exists x\, P(x)\big)$?

merging all contexts into an infinite context

category of contexts for a given type theory

- **objects:**

$$\Gamma$$

  Γ is a finite or countably infinite context

- **morphisms:**

$$\Gamma \xrightarrow{f} \Gamma'$$

  f is an injection mapping the variables from Γ to variables from Γ'

this category has pushouts:

every two contexts can be combined into a bigger context

direct limit of all contexts:

$$\Gamma_\infty$$

the system Γ_∞

Γ_∞: system equivalent to the PTS rules but without explicit contexts
(we reuse the name of the infinite context for the system)

PTS $=$ Pure Type System

we write

$$M : A$$

to morally mean

$$\Gamma_\infty \vdash M : A$$

Γ_∞ **preterms:**

$$A ::= s \mid x_i^A \mid x_i \mid AA \mid \lambda x_i{:}A.A \mid \Pi x_i{:}A.A$$

two kind of variables: free variables x_i^A and bound variables x_i
superscript A of x_i^A may be *any* preterm

two of the six Γ_∞ rules

PTS rules	**equivalent Γ_∞ rules**
$\dfrac{\Gamma \vdash A : s}{\Gamma,\, x_i : A \vdash x_i : A}$	$\dfrac{A : s}{x_i^A : A}$
$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma,\, x_i : A \vdash B : s_2}{\Gamma \vdash \Pi x_i{:}A.B : s_3}$	$\dfrac{A : s_1 \quad B : s_2}{\Pi x_i{:}A.B[x_j^A := x_i] : s_3}$

binding a variable in Γ_∞: replace a free variable by a bound variable derivable PTS judgment $\longleftrightarrow$ derivable Γ_∞ judgment

from left to right:

alpha convert the judgement to separate free from bound variables
then: remove the context

from right to left:

topological sort of the free variables in the Γ_∞ judgement
then: put them in that order in the context

implementing Γ_∞ for LF

the datatypes of the kernel

```
type preterm =
| Star
| Ref of int
| Var of string * preterm
| Const of string * preterm * preterm list
| App of preterm * preterm
| Bind of int * preterm * preterm
```

axioms used

$0 = \lambda,\ 1 = \Pi$

```
type term = private
| Box
| In of preterm * term

type red = private
| Red of preterm * preterm
```

difference of this approach with other kernels

most purely functional kernels:

```
        App : preterm * preterm -> preterm
  typecheck : state -> (preterm -> term)
extend_state : string * preterm -> state -> state
```

mutually inconsistent terms (from mutually inconsistent states) possible
internally inconsistent state *not* possible

our approach:
function application in LCF style

```
app : term * term -> term
```

mutually inconsistent terms *not* possible

comparing kernel sizes

current HOL Light		all lines	content
kernel	`fusion.ml`	669	394

stateless HOL Light			
kernel	`core.ml`	404	330
state	`state.ml`	95	71

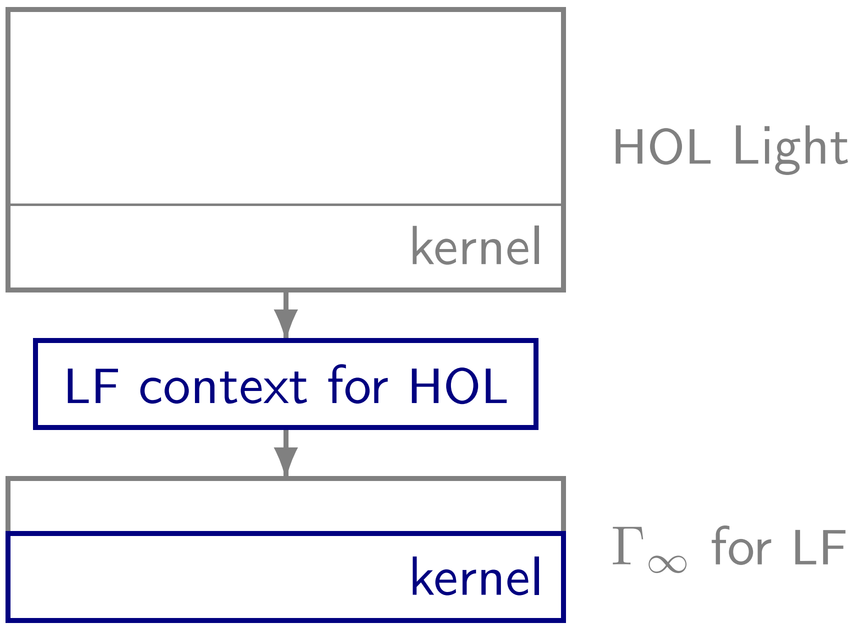
Γ_∞ for LF		
kernel	214	166
convertibility	64	49
typechecker	29	25

outlook

future work

but does this scale?

experiment:



how much slower than current HOL Light? 100 times? ∞ times?