Evaluation Opportunities in Mechanized Theories

Joe Hurd

Galois, Inc.
joe@galois.com

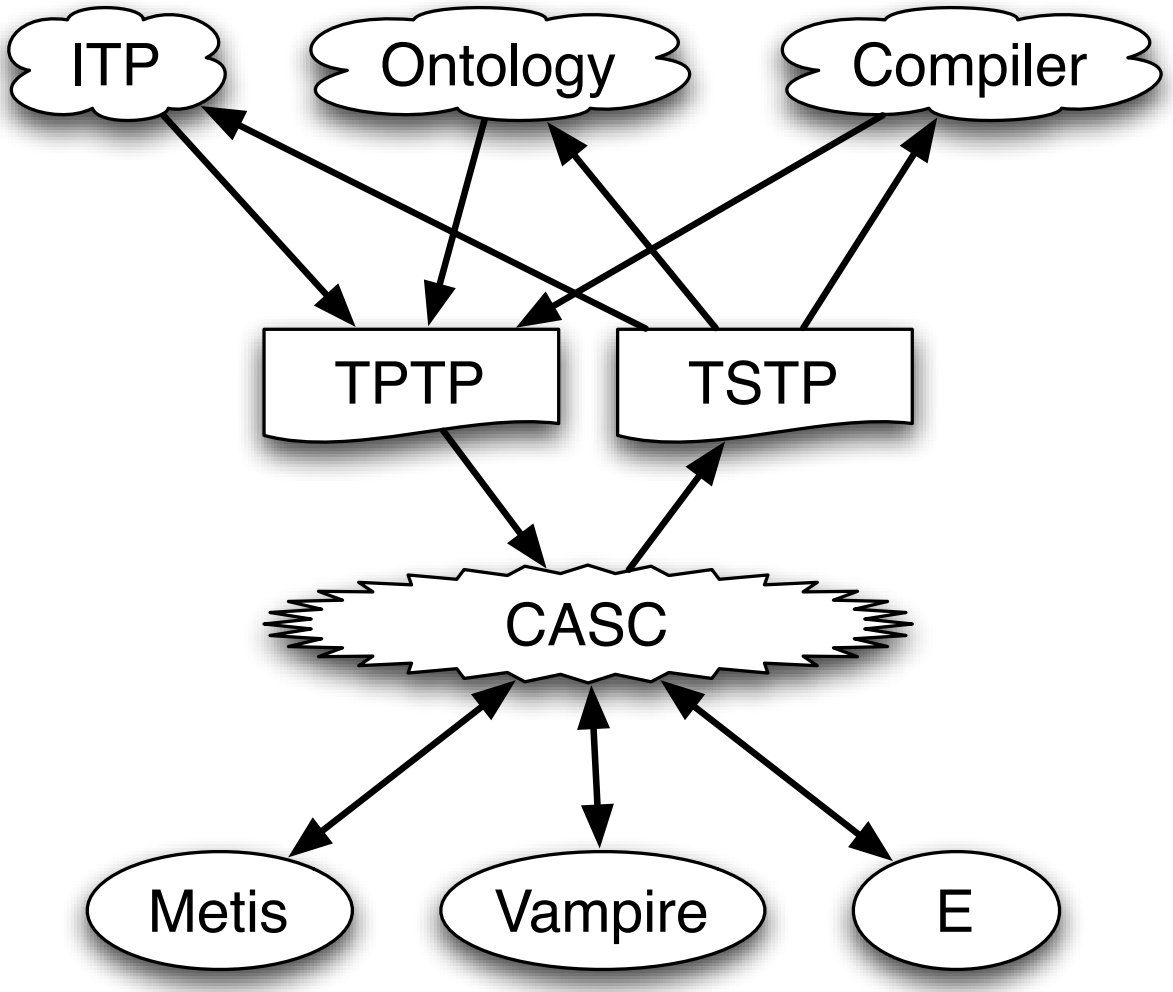
EMS+QMS Workshop
Tuesday 20 July 2010

galois

Talk Plan

galois

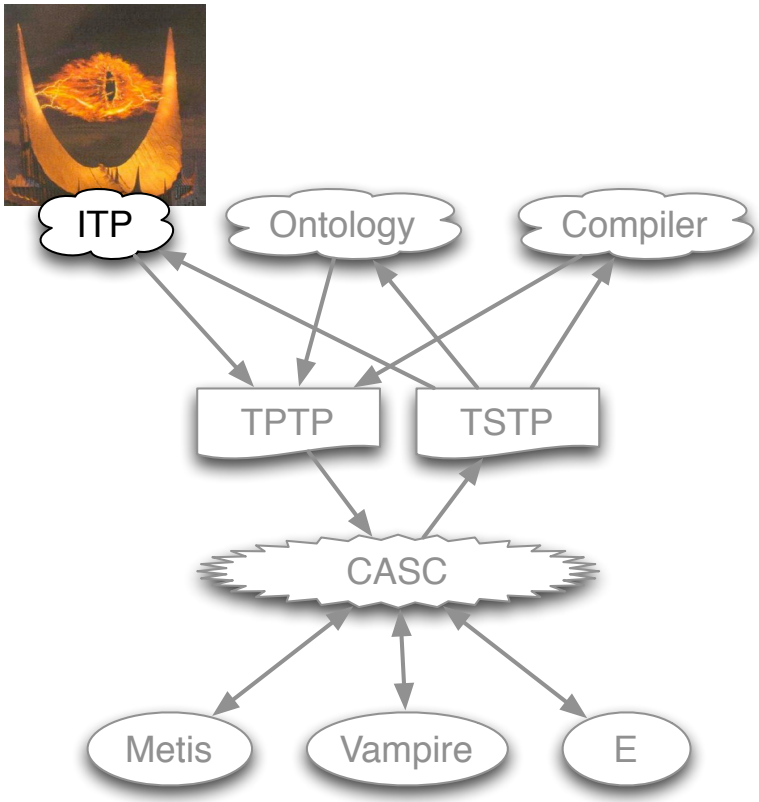
A Healthy Ecosystem



galois

The View from an Application Domain

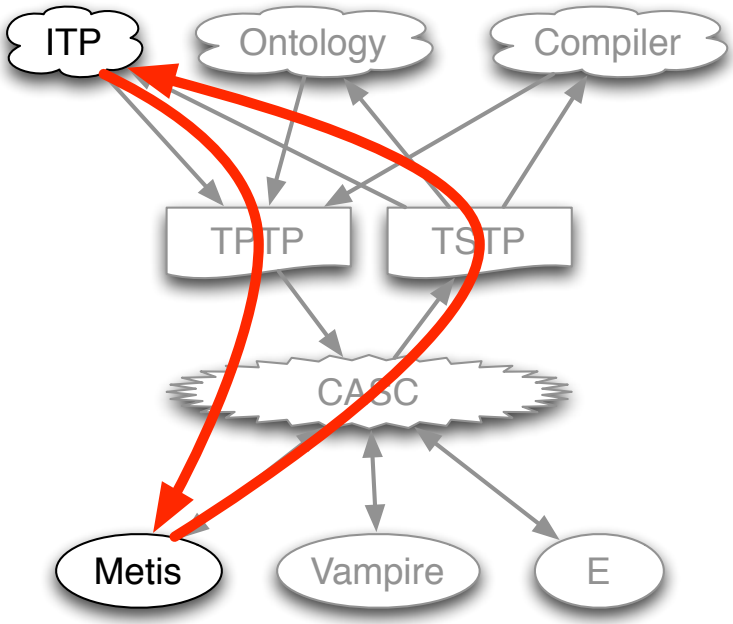
- Application domains are the source of requirements for solvers and solutions.
- This talk will survey the domain-specific requirements for ITPs.
- **Disclaimers:** HOL-centric, IMHO, YMMV, IANAL, ...



galois

Domain-Specific Solver Metrics

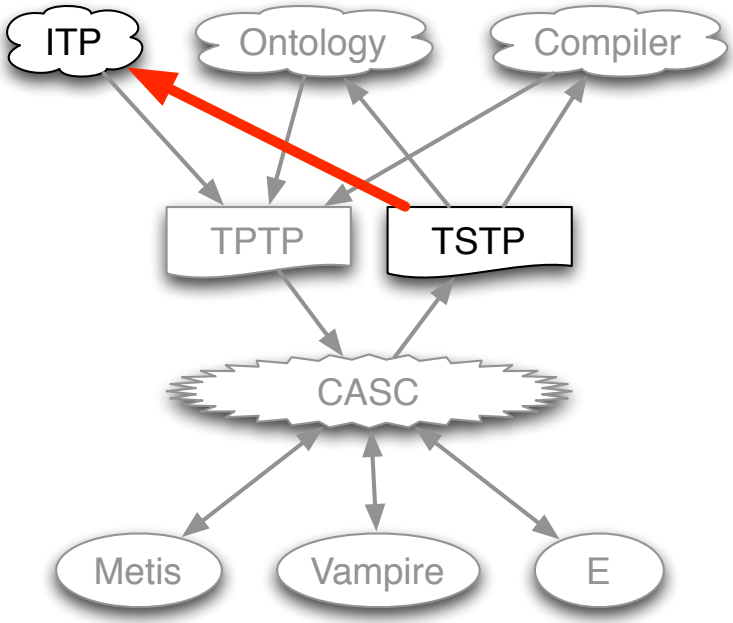
- Constructing a logical theory in an ITP is a labour-intensive activity.
- Solvers have the potential to automate much of this proof effort.
- The first part of this talk will look at success factors for solvers being used to support ITP.



galois

Domain-Specific Proof Metrics

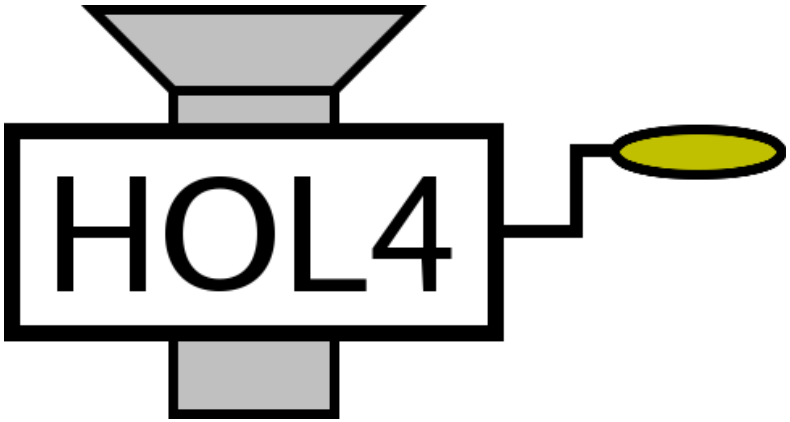
- ITPs have a need to store logical theories and their proofs for later use.
- The second part of this talk will look at desirable properties of proofs for consumption by ITPs.



galois

Anatomy of an Interactive Theorem Prover

Mechanically extracted proof



Theorems

- ITPs are really high assurance proof checkers.
- Tactics generate pieces of proof as a by-product of breaking down goals into subgoals.
- Integrating a solver into an ITP just means wrapping it up as a tactic.

galois

Theorem Provers in the LCF Design

- A theorem $\Gamma \vdash \phi$ states *"if all of the hypotheses* Γ *are true, then so is the conclusion* ϕ*"*.
- The novelty of Milner's Edinburgh LCF ITP was to make `theorem` an abstract ML type.
- Values of type `theorem` can only be created by a small logical kernel which implements the primitive inference rules of the logic.
- Soundness of the whole ML ITP thus reduces to soundness of the logical kernel.

galois

The HOL Logical Kernel

$$\frac{}{\vdash t = t}\ \text{refl } t \qquad \frac{}{\{\phi\} \vdash \phi}\ \text{assume } \phi \qquad \frac{\Gamma \vdash \phi = \psi \quad \Delta \vdash \phi}{\Gamma \cup \Delta \vdash \psi}\ \text{eqMp}$$

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\lambda v.\ t_1) = (\lambda v.\ t_2)}\ \text{absThm } v \qquad \frac{\Gamma \vdash f = g \quad \Delta \vdash x = y}{\Gamma \cup \Delta \vdash f\ x = g\ y}\ \text{appThm}$$

$$\frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{(\Gamma - \{\psi\}) \cup (\Delta - \{\phi\}) \vdash \phi = \psi}\ \text{deductAntisym} \qquad \frac{\Gamma \vdash \phi}{\Gamma[\sigma] \vdash \phi[\sigma]}\ \text{subst } \sigma$$

$$\frac{}{\vdash (\lambda v.\ t_1)\ t_2 = t_1[t_2/v]}\ \text{betaConv } ((\lambda v.\ t_1)\ t_2) \qquad \frac{}{\vdash c = t}\ \text{defineConst } c\ t$$

$$\frac{\vdash \phi\ t}{\vdash \mathit{abs}\ (\mathit{rep}\ a) = a \quad \vdash \phi\ r = (\mathit{abs}\ (\mathit{rep}\ r) = r)}\ \text{defineTypeOp } n\ \mathit{abs}\ \mathit{rep}\ \mathit{vs}$$

galois

Case Study: Using Metis to Automate Proof in HOL4

- Metis is an automatic theorem prover for first order logic.
  - Implements the ordered paramodulation calculus.
  - Reliably props up the table at CASC :-)
- It is a standalone tool, but has been integrated as a tactic in the HOL4 and Isabelle theorem provers.

Proof Script (HOL4 proof using the Metis tactic)

```
prove (
  ‘(∀P. (∀n. (∀m. m < n ⟹ P m) ⟹ P n) ⟹ ∀n. P n) ⟹
   ∀P. P 0 ∧ (∀n. P n ⟹ P (suc n)) ⟹ ∀n. P n‘,
  METIS_TAC [ ⊢ ∀n. n < suc n,
              ⊢ ∀m. m = 0 ∨ ∃n. m = suc n ]);
```

- By looking at this case study we hope to extract general principles for successfully integrating solvers into ITPs.

galois

Integrating Metis as a HOL4 Tactic

Here's how the Metis tactic proves the higher order logic goal g:

1. Call a CNF normalization tactic, generating a HOL4 proof

$$\{\neg\exists\,\vec{a}.\ (\forall\,\vec{v_1}.\ c_1) \wedge \cdots \wedge (\forall\,\vec{v_n}.\ c_n)\} \quad \vdash \quad g \qquad (1)$$

2. Map each HOL4 term c_i to a first-order logic clause C_i.
3. Run Metis to find a refutation ρ for the clause set $\{C_1, \ldots, C_n\}$.
4. Replay the refutation ρ as a HOL4 proof

$$\{(\forall\,\vec{v_1}.\ c_1), \ldots, (\forall\,\vec{v_n}.\ c_n)\} \quad \vdash \quad \bot \qquad (2)$$

5. Combine (1) and (2) to derive the HOL4 theorem $\vdash g$.

galois

Replay: Lifting First Order Refutations to HOL4 Proofs

- LCF Design: The only functions that can create theorems are the primitive inferences in the logical kernel.
- The Metis tactic must translate the refutation of first order clauses to a sequence of primitive inferences.
- Plea to Solver Developers: Please output explicit proofs!

Desirable Feature	**G**	**M**
A small number of possible inference steps	√	√
Each inference step is a small logical step	×	√
Easy to parse	×	√

galois

Replay: Implications for the Solver

- Soundness of the solver is actually not very important.
  - It's not in the trusted code base.
- The solver must store enough information to be able to output a proof at the end.
  - Return to the Stanford LCF days of storing explicit proofs.
- Interesting time/space trade-offs possible by replaying part of the proof as primitive inferences, and then keeping the HOL theorem instead of the solver proof.

galois

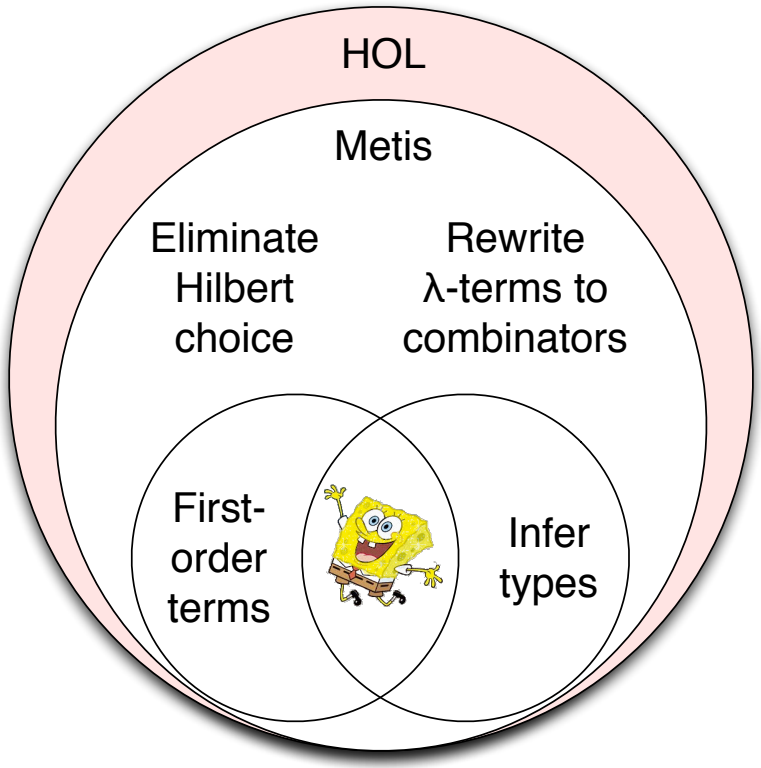
Run: Executing a Solver from an ITP

- The more tightly integrated, the easier it is.
- Metis gets a huge boost from being written in the same language as HOL4 and Isabelle (Standard ML).
  - No need for a separate build system.
  - No need for a foreign function interface.
  - No need to print goals and parse proofs.
- But not too tight!
  - Metis and HOL4 have separate term languages.
  - An intentional loose coupling to support different logical mappings.

galois

Map: Pushing HOL4 Goals into First Order Clauses

- There are many features of the HOL4 logic that make automatic proof difficult.
  - Hilbert choice ϵ.
  - Unification.
  - Boolean terms.
- Luckily, most HOL4 goals don't use these features.
- Simple strategies map valid goals to unsatisfiable first order clauses.
- More forgetful mapping $\approx$ happier prover.



galois

Mapping Push-Back

higher order $\surd$ first order $\times$

$\vdash \exists x.\ x$ (x is a boolean variable)

$\vdash \exists f.\ \forall x.\ f\ x = x$ (f is a function variable)

$\vdash \forall f, s, a, b.\ (\forall x.\ f\ x = a) \wedge b \in \text{image}\ f\ s \implies a = b$ (f has different arities)

typed $\surd$ untyped $\times$

$\vdash \text{length}\ ([] : \mathbb{N}^*) = 0 \wedge \text{length}\ ([] : \mathbb{R}^*) = 0 \implies \text{length}\ ([] : \mathbb{R}^*) = 0$ (indistinguishable terms)

$\vdash (\forall x.\ (x : \text{unit}) = c) \implies (a : \text{unit}) = b$ (bad proof via $\top = \bot$)

galois

Mapping Example

Map the HOL4 goal $n < n + 1$ to first order logic.

Mapping	**First Order Formula**
first order, untyped	$n < n + 1$
first order, typed	$(n : \mathbb{N}) < ((n : \mathbb{N}) + (1 : \mathbb{N}) : \mathbb{N})$
higher order, untyped	$\uparrow ((< \,.\, n) \,.\, ((+ \,.\, n) \,.\, 1))$
higher order, typed	$\uparrow (((< : \mathbb{N} \to \mathbb{N} \to \mathbb{B}) \,.\, (n : \mathbb{N}) : \mathbb{N} \to \mathbb{B}) \,.\, (((+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}) \,.\, (n : \mathbb{N}) : \mathbb{N} \to \mathbb{N}) \,.\, (1 : \mathbb{N}) : \mathbb{N}) : \mathbb{B})$

galois

Mapping Efficiency

- Effect of the mapping on the time taken by model elimination calculus to prove a HOL version of Łoś's 'nonobvious' problem:

Mapping	untyped	typed
first order	1.70s	2.49s
higher order	2.87s	7.89s

- These timing are typical, although 2% of the time higher order, typed does beat first order, untyped.

galois

Mapping Choice

- We choose between first order and higher order by analyzing the goal to see whether it has higher order features.
  - Presence of boolean/function/relation variables.
  - Different arity uses of the same function.
- We run in untyped mode, and if an error occurs during proof translation then restart search in typed mode.
  - Restart necessary in $< 1\%$ of the time.
- This automatic strategy results in a good coverage trade-off between broad and simple.
- Question for Solver Developers: What mental model does a typical user have of your solver's coverage?

galois

Performance

- What are the right performance metrics for ITP tactics?
- *"All tools are user interfaces"* – Clark Dodsworth
- Apply user-centered design to answer this question: start with the user interface and work backwards.
- Standard interaction mode: user drives the ITP by choosing tactics to break down goals.

galois

Response Times for Interactive Applications

This is a subject that has been well-studied by usability experts.[1]

- **0.1 seconds** gives the feeling of **instantaneous** response—that is, the outcome feels like it was caused by the user, not the computer. This level of responsiveness is essential to support the feeling of **direct manipulation**.
- **1 second** keeps the user's flow of thought **seamless**. Users can sense a delay, and thus know the computer is generating the outcome, but they still feel in control of the overall experience and that they're moving freely rather than waiting on the computer. This degree of responsiveness is needed for good navigation.
- **10 seconds** keeps the user's attention. From 1–10 seconds, users definitely feel at the mercy of the computer and wish it was faster, but they can handle it. After 10 seconds, they start thinking about other things, making it harder to get their brains back on track once the computer finally does respond.

[1] Jakob Nielsen's Alertbox, June 21, 2010

galois

Response Times for ITP Tactics

- **<0.1s: Direct manipulation**
  - *"I know this goal is true, and I know this tactic can prove it."*
  - Solver Completeness Matters: Solvers can lose big reputation points if they fail in these situations.
- **0.1–1s: Good navigation**
  - *"I know this goal is true; I wonder if this tactic can prove it."*
- **1–10s: Keeps the user's attention**
  - *"I don't know if this goal is true; let's see what this tactic does."*
- **>10s: Unacceptable for user interaction**
  - But solvers run in the background may be able to discover useful information before the user finishes the whole proof.

galois

Background Mode Proof Tools

- New category of proof tools developed for Isabelle.
- Sledgehammer runs powerful ATPs on Isabelle subgoals.
  - Metis helps translate the resulting proofs.
  - Competition Design Win: ATP effectiveness was found to be highly correlated with CASC performance.
- Nitpick is a powerful counter-example finder.
- Background processing mode completely changes the game.
  - √ Solvers have much longer to search.
  - √ Counter-example finders can 'help' the user fail early.
  - × The user provides no hints to the solver.
  - × *"Careful with that resource-usage, Eugene"*

galois

OpenTheory Proof Archive

- In theory, proofs are immortal.
- In practice, proofs that depend on theorem provers bit-rot at an alarming rate.
- Idea: Compile proofs to primitive inferences, and archive them as theory packages.
- The goal of the OpenTheory project is to transfer the benefits of package management to logical theories.[2]

[2] OpenTheory was started in 2004 with Rob Arthan.

galois

Theories and Proofs

- A theory $\Gamma \triangleright \Delta$ of higher order logic consists of:
  1. A set Γ of assumptions.
  2. A set Δ of theorems.
  3. A formal proof that the theorems in Δ logically derive from the assumptions in Γ.
- The assumptions and theorems are the interface of the theory.
- From a logical point of view, it only matters that a proof exists deriving the theorems from the assumptions.
- What does it mean for one proof to be better than another?

galois

Proof Quality Metrics

For a fixed theory $\Gamma \triangleright \Delta$, here are some proof quality metrics:

- The time/space cost of replaying the proof.
- The local definitions made by the proof.
- The size of the stored proof.

galois

Replay Time/Space Cost

- OpenTheory proofs are represented as programs for a stack-based virtual machine.
  - There are commands for building types and terms, and performing primitive inferences.
  - The stack avoids the need to store the whole proof in memory.
- A dictionary is used to support structure sharing.
  - The commands should preserve structure sharing as much as possible to avoid a space blow-up.
  - Interesting Challenge: Substitution.

galois

Local Definitions

- A proof may make local definitions of type operators and constants that do not appear in the theory interface.
- Many theorem provers will not allow two constants with the same name to be defined.
  - Preventing: Define $\vdash c = 0$; define $\vdash c = 1$; derive $\vdash 0 = 1$.
- In such systems, local definitions are symbol table pollution.
  - However, it is possible to avoid this problem by implementing a purely functional logical kernel.
  - The OpenTheory logical kernel demonstrates this.

galois

Storage Size

- Proofs are big!
- Example: The base theories loaded by the HOL Light ITP.
  - Proving them requires 769,138 primitive inferences.
  - Storing them as a gzipped OpenTheory proof file takes 18Mb.
- Most of the time, tools will process the interface of the theory $\Gamma \triangleright \Delta$ rather than the proof inside.
- But there's still storage and distribution requirements.

galois

Compressing Proofs

- It is possible to compress existing proofs.
  - The equivalent of hash-consing is effective on proofs that have been expanded to primitive inferences.
  - Solvers can be used to compress proofs (Larry Wos' work).
- However, there is greater potential for compression at proof construction time.
  - Solver Developers: Consider the effect on proof size when choosing strategies.
- Digression: Even more potential for compression at the level of the human guiding the overall proof.
  - Example: The proof of the 4 colour theorem.

galois

Example: Clausification

Naive clausification can generate really big proofs:

$$\text{CNF}\left(\begin{array}{l}(a_0 \wedge a_1 \wedge a_2 \wedge a_3) \vee (b_0 \wedge b_1 \wedge b_2 \wedge b_3) \vee \\ (c_0 \wedge c_1 \wedge c_2 \wedge c_3) \vee (d_0 \wedge d_1 \wedge d_2 \wedge d_3)\end{array}\right)$$

$$=$$

$$\begin{array}{c}(a_3 \vee b_3 \vee c_3 \vee d_0) \wedge (a_2 \vee b_3 \vee c_3 \vee d_0) \wedge \\ (a_1 \vee b_3 \vee c_3 \vee d_0) \wedge (a_0 \vee b_3 \vee c_3 \vee d_0) \wedge \\ \ldots 992 \text{ more atoms} \ldots \\ (a_0 \vee b_3 \vee c_3 \vee d_3) \wedge (a_1 \vee b_3 \vee c_3 \vee d_3) \wedge \\ (a_2 \vee b_3 \vee c_3 \vee d_3) \wedge (a_3 \vee b_3 \vee c_3 \vee d_3)\end{array}$$

galois

Example: Clausification

Definitional CNF guarantees the size of normalized terms will be linear in the size of original terms:

$$\text{DEF\_CNF}\left(\begin{array}{l}(a_0 \wedge a_1 \wedge a_2 \wedge a_3) \vee (b_0 \wedge b_1 \wedge b_2 \wedge b_3) \vee \\ (c_0 \wedge c_1 \wedge c_2 \wedge c_3) \vee (d_0 \wedge d_1 \wedge d_2 \wedge d_3)\end{array}\right)$$

$$=$$

$$\begin{array}{l}\exists v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}. \\ \quad (v_{11} \vee \neg d_0 \vee \neg v_{10}) \wedge (v_{10} \vee \neg v_{11}) \wedge (d_0 \vee \neg v_{11}) \wedge \\ \quad (v_{10} \vee \neg d_1 \vee \neg v_9) \wedge (v_9 \vee \neg v_{10}) \wedge (d_1 \vee \neg v_{10}) \wedge \\ \qquad \ldots 59 \text{ more atoms} \ldots \\ \quad (v_0 \vee \neg v_1) \wedge (a_1 \vee \neg v_1) \wedge (v_0 \vee \neg a_2 \vee \neg a_3) \wedge \\ \quad (a_3 \vee \neg v_0) \wedge (a_2 \vee \neg v_0) \wedge (v_2 \vee v_5 \vee v_8 \vee v_{11})\end{array}$$

galois

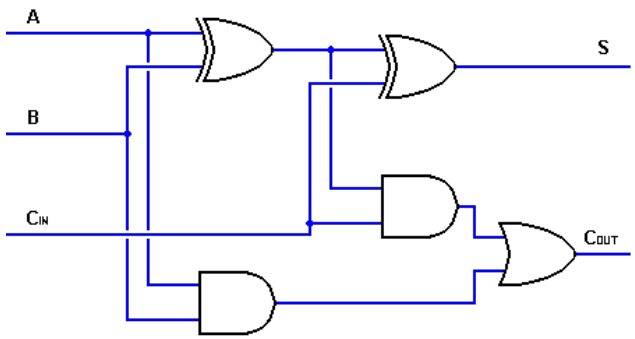
Example: Clausification

- The clausification for the Metis tactic in HOL4 uses definitional CNF.
  - Uses a greedy strategy to minimize the number of clauses.
- Example: In the current distribution of HOL4 there are 1,136 successful calls to the Metis tactic.
  - These result in 2,880,406 primitive inferences.
  - But 78% of these are spent on clausification.
- Open Question: Can we get the benefits of powerful proof calculi without paying a high normalization cost?

galois

A Module Language for Theories

- Can view a theory $\Gamma \triangleright \Delta$ as a functor from assumptions Γ to theorems Δ.
- A module language allows theories to be plugged together into compound theories.



- Crafting general theory functors has great potential.
  - Reduce storage requirements for proofs.
  - Avoid duplication of proof effort.
- Advert: Will talk about this tomorrow at the VERIFY workshop.

galois

Summary

- Application domains should be guiding the design of solvers.
  - Competition designers: continue to listen for new requirements.
  - Success Story: The LTB division at CASC.
- Domain-specific factors are often counter-intuitive.
  - Example: Soundness/completeness trade-off.
- For more details on the case studies see the project web pages:

`http://gilith.com/software/metis`
`http://gilith.com/research/opentheory`

galois