Planning the project. About 9-10 plan to participate. 5 teams, 1 theorem prover.

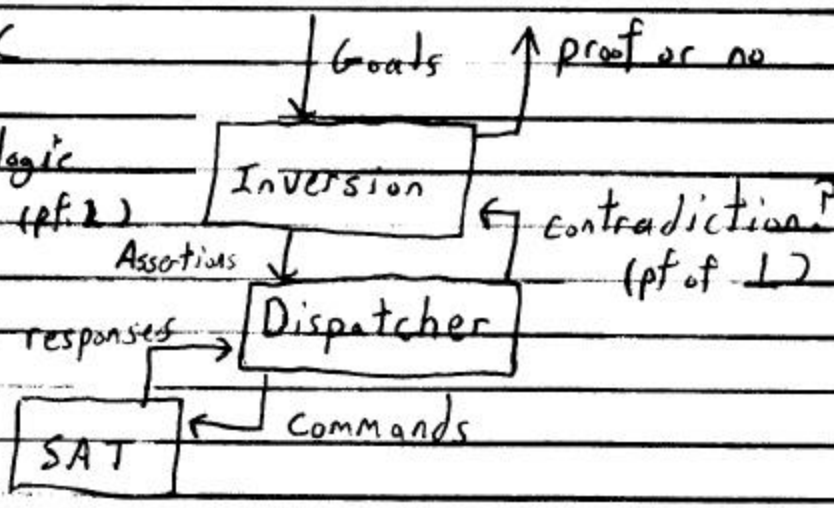
Review. Nelson - Oppen strategy.

Combine SAT procedures for theories that meet conditions, produce a SAT procedure for the combination.

Topic: Want to generate proofs.

(1) Helps to pinpoint cause of failure. (cf. model checking)
(2) what happens on success?
(3) Decision procedures can be far from the axioms of the theory.
(4) Don't want to trust the theorem prover.
(5) helps with PCC

Goals ⊆ 1st order logic (pf 2)



Dispatcher broadcasts commands, monitors responses.

If 1 detects a contradiction, the conjunction is UNS

Kernels of the proof come from SAT procedures Produce proofs for equality. Make contradictions carry a proof of false.

axiom

Axioms $\dfrac{}{T}$ truei truei : "proof of true"
: pf T

- Need dependent types. Types that contain expressions (in our case, predicates).

pf : pred → Type pred : Type

$D ::= \dfrac{\vdash P_1 \quad \vdash P_2}{\vdash P_1 \wedge P_2}$ andi andi : pf P_1 → pf P_2 → pf $(P_1 \wedge P_2)$

("D_1" = $\vdash P_1$, "D_2" = $\vdash P_2$)

Proof checks? andi (D_1, D_2) is a valid proof of $P_1 \wedge P_2$?
check recursively... andi (D_1, D_2) : pf $(P_1 \wedge P_2)$
D_1 : pf P_1 ?
D_2 : pf P_2 ?

$\dfrac{\vdash E_1 = E_2 \quad \vdash E_2 = E_3}{\vdash E_1 = E_3}$ treq treq : pf $(E_1 = E_2)$ → pf $(E_2 = E_3)$ → pf $(E_1 = E_3)$

$\dfrac{\overset{u}{\overline{\vdash P_1}} \quad \vdots \quad \vdash P_2}{\vdash P_1 \Rightarrow P_2}$ Assumption of P_1 can only be used locally.

e.g., you cannot prove:

$$\dfrac{\dfrac{\overline{\vdash P}}{\vdots}}{\dfrac{\vdash P \Rightarrow P \quad \vdash P}{\vdash (P \Rightarrow P) \wedge P}}$$

unsound!

So proofs are not trees. Need side conditions.

Give assumption a name, and say that it

is bound ... treat the proof like a function and the assumption like a local argument.

impi : $(pf\ P_1 \to pf\ P_2) \to pf\ (P_1 \Rightarrow P_2)$

Need more than dependent types. We need higher order representation. No more side conditions. Constructors now apply to functions, not just sub-trees.

We have "$\Rightarrow$" : pred → pred → pred
"T" : pred

How about $\forall x.\ a$? all "x" P_x Problems.
① α-substitution all "y" P_y
same predicate, different representation
② $\forall x\ \forall x\ P_x$ all "x" (all "x" P_x) No!
Can be confusing

Use notion of bound variables in the meta-language:

all : (exp → pred) → pred

Body of "all" is a function that takes an exp and produces a function:

all (λx : exp . P_x)

Can do substitution for free, using meta-language.

$$\frac{\vdash \forall x.\ P}{\vdash P[E/x]}\ \text{alle}$$

alle: (all F) → pf(F(E))

Proof checker implements substitution, but you can express it clearly.

Edinburg LF = dependent types, higher order representation designed for proofs, meta proofs

$$\frac{E_1 = E_2 \quad E_2 \neq E_3 \quad E_3 = E_4}{\varepsilon_1 \neq \varepsilon_4}$$ (not often used)

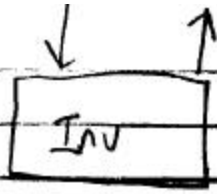
$$\frac{\vdash E_1 = E_2 \quad \vdash E_1 \neq E_2}{\vdash \bot}\ \text{eq neq}$$

$$\frac{\begin{array}{c}\vdash \neg P \\ \vdots \\ \vdash \bot\end{array}}{\vdash P}\ \text{contra}$$

needs higher order representation

contra: (pf(not P) → pf ⊥) → pf P

$$\frac{\vdash \bot}{\vdash P}\ \text{falsee}$$



inv: $G \to pf\ G$ throws Failure

assert: $H \to pf\ H \to unit$
throws Contradiction (pf ⊥)

SAT

} per-theory

snapshot: $unit \to unit$
undo: $unit \to unit$
satproc: $\bot \to pf\ \bot \to \bot\ set$ throws Contradiction (PP ⊥

$G ::= T \mid \bot \mid G_1 \wedge G_2 \mid H \Rightarrow G \mid \forall x.\ G \mid L$
$H ::= L \mid T \mid \bot \mid H_1 \wedge H_2 \mid \neg L$

Will only write negation for Literals.

inv(T) = true;

~~inv($G_1 \wedge G_2$)~~

inv(⊥) = throw Failure

inv($\forall x.\ G$) = alli(λx. inv(G)) ← replace x with fresh, new variable

~~all : exp~~ ← must refer to name of argument in type of result

alli : (Πx : exp pf(P_x)) → pf(all P)

inv($G_1 \wedge G_2$) = andi(inv(G_1), inv(G_2))

inv($H \Rightarrow G$) = try snapshot ()
try
assert(H, u); u is fresh
impi (λu:pfH. inv(G))
handle Contradiction(prf) → ~~false~~

↳ impi (λu:pfH. falsee(prf))

```
inv(L) = try snapshot
         try
           assert(¬L, u)        /* u fresh */
           throw Failure
         handle Contradiction(prf) →
           contra(λu: pf ¬L. prf)
         finally undo()

assert(T, D) = ()
assert(H1 ∧ H2, D) = assert(H1, andel D);
                     assert(H2, ander D)
assert(⊥, D) = throw Contradiction(D)

assert(L, D) = acc = {(L, D)}
               while acc ≠ ∅ do
                 extract (L', D') from acc
                 acc = acc ∪ satprod(L', D')
               end
               ()
```

Example

$$x = y \Rightarrow \left(y = z \Rightarrow x = z \;\wedge\; x \neq y \Rightarrow P \right)$$

```
λu1: pf x=y.
  andi ( impi (λu2: pf (y=z)
                contra(λu3: x≠z.
                  eqneq (treq(u1,u2), u3))),
         impi (λu4: pf (x≠y).
                falsee (eqneq (u1, u4))))
```

didn't need contradiction here, could have optimized.