Planning the project. About 9-10 plan to participate. 5 teams, 1 team is proven.

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 \downarrow \text{proof or no}

Dispatcher broadcasts

goals \rightarrow \text{Inversion}

contradiction?

pf of \text{I}

Dispatcher

\text{SAT}

If I detects a contradiction, the conjecture is unsat.

Kernels of the proof come from SAT procedures produce proofs for equality. Make contradictions carry a proof of false.
Axioms: \[ \text{true} : \text{true} : \text{"proof of true"} \]

\[ \text{pf } \text{T} \]

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

\[ \text{pf } : \text{pred } \rightarrow \text{Type} \quad \text{pred } : \text{Type} \]

\[ \text{D}_1, \text{D}_2 : \text{pf p}_1, \text{pf p}_2 \rightarrow \text{pf (p}_1 \land \text{p}_2) \]

\[ \text{proof check?} \quad \text{and:} \quad \text{D}_1, \text{D}_2 : \text{pf (p}_1 \land \text{p}_2) \]

\[ \text{check recursively:} \quad \text{D}_1 : \text{pf p}_1 ? \quad \text{D}_2 : \text{pf p}_2 ? \]

\[ \text{treq : pf (e}_1 = e_2) \rightarrow \text{pf (e}_2 = e_3) \]

\[ \text{assumption of p}_1 \text{ can only be used locally.} \]

\[ \text{unsound!} \]

\[ \text{e.g.: you cannot prove: } \text{pf (p } \Rightarrow \text{p) } \text{ } \text{p} \]

- So proofs are not trees. Need side conditions.

\[ \text{Give assumption a name, and say that it} \]
is bound... treat the proof like a function and the assumption like a local argument.

\[ \text{imp:} \ (pf\ p_1 \rightarrow pf\ p_2) \rightarrow pf\ (p_1 \Rightarrow p_2) \]

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

We have 

"⇒" : pred → pred → pred

"⇒" : pred

How about \( \forall x\ a \) all "\( x \)" \( P_x \) Problems.

1. \( \alpha \) - substitution all "\( y \)" \( P_y \)

some 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:

\( \text{all:} \ (\text{exp} \rightarrow \text{pred}) \rightarrow \text{pred} \)

Body of "\( \text{all} \)" is a function that takes an exp and produces a function:

\( \text{all} \ (\exists x : \text{exp} , P_x) \)

Can do substitution for free, using meta-language,
\[ \forall x. P \quad \text{alle} \quad \text{alle} \]
\[ \vdash P[\xi/x] \quad \text{alle}: (\forall F)(\exists F(F(E))) \]

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

Edinburgh LF = dependent types, higher order representations designed for proofs, meta-proofs.

\[ E_1 = E_2; E_2 \neq E_3; E_3 \neq E_4 \quad \text{(not often used)} \]
\[ E_1 \neq E_4 \]

\[ \vdash E_1 = E_2 \quad \vdash E_1 \neq E_2 \quad \text{eq neq} \]
\[ \vdash \top \]

\[ \vdash \top \quad \text{contra representation} \]

\[ \vdash (\text{pf}(\neg \text{pf P}) \Rightarrow \text{pf } \bot) \Rightarrow \text{pf } \text{P} \]

\[ \vdash \bot \quad \text{false} \]
\[ \vdash \bot \quad \text{false} \]
inv: \( G \rightarrow pf\ G \) throws Failure

assert: \( H \rightarrow pf\ H \rightarrow unit \)

\[ \text{throws} \quad \text{Contradiction (pf L)} \]

\[ \text{SAT} \]

snapshot: \( \text{unit} \rightarrow \text{unit} \)

undo: \( \text{unit} \rightarrow \text{unit} \)

sat proc: \( I \rightarrow pf\ L \rightarrow I \text{ set} \quad \text{throws} \quad \text{Contradiction (pf L)} \)

\[ G ::= \, T \mid I \mid G \cdot G \mid H \rightarrow G \mid \forall x.\ G \mid L \]

\[ H ::= \, L \mid T \mid I \mid H \cdot H \mid 1 \mid L \]

(Will only write negation for literals,)

\[ \text{inv (T)} = \text{true}; \]

\[ \text{inv (I)} = \text{the_\_Failure}; \]

\[ \text{inv ( pf ( \forall x. \ G ) ) = all} \cdot (2 \times \text{ inv (G)}); \]

\[ \text{fresh, new variable} \]

\[ \text{aff} \cdot \text{ must refer to name of argument} \]

\[ \forall x \text{ in } \text{ type of } \text{result} \]

\\[ \text{all} \cdot ( \forall x : \text{exp} \quad pf\ (P_x) \rightarrow pf\ (\text{all}\ П) \]

\[ \text{inv (} G \cdot G\text{) = and (inv (} G\text{), inv (} G\text{))} \]

\[ \text{inv (} H \rightarrow G\text{) = try snapshot (} G\text{)} \]

\[ \text{try} \]

\[ \text{assert (} H, \text{ u); \ u \text{ is fresh} \]

\[ \text{impl (} false \cdot \text{inv (G))} \]

\[ \text{handle} \quad \text{Contradiction (pf)} \]

\[ \text{break} \]

\[ \text{impl (} 2 \cdot \text{impl (false (pf))} \} \]
\[
\text{inv}(L) = \text{try} \\
\text{assert}(L, u) \text{ / a fresh } \text{ /} \\
\text{throw Failure} \\
\text{handle Contradiction}(\text{prf}) \Rightarrow \\
\text{contra}(\text{nu} : \text{pf}(L, \text{prf})) \\
\text{finally undo(} \text{)}
\]

\[
\text{assert}(T, D) = () \\
\text{assert}(H_1 \land H_2, D) = \text{assert}(H_1, \text{ and } D) ; \\
\text{assert}(L, D) = \text{throw Contradiction}(D) \\
\text{assert}(L, D) = \text{acc} = \$ [L, D] \}
\]

\[
\text{while acc } \neq \emptyset \text{ do} \\
\text{extract }(L', D') \text{ from acc} \\
\text{acc} = \text{acc } \cup \text{ satproc}(L', D') \\
\text{end}
\]

**Example**

\[
x = y \Rightarrow (y = z \Rightarrow x = z \land x \neq z \Rightarrow P)
\]

\[
\lambda u_1 : \text{ pf } x = y. \\
\text{ and } \{ \text{ imp } (2u_2 : \text{ pf } (y = z)), \\
\text{ contra } (2u_3 : x \neq z), \\
\text{ eq } \text{ neq } (\text{ try } (u_1, u_2), u_3) \}
\]

\[
\text{didn't need } \text{ (contra here, but have opted) } \}
\]

\[
\text{false } \text{ (eq } \text{ neq } (u_1, u_2) )
\]