Adding tactic support to Nelson-Oppen

Recall Isabelle state

\[ [G_1, \ldots, G_n] \Rightarrow G \]

- \( G \) is the ultimate proving goal
- \( G_1, \ldots, G_n \) are the current goals
- Initially \([G] \Rightarrow G\)

At any time, Isabelle has proved the theorem \( G_1 \land \ldots \land G_n \Rightarrow G \)

Back to Nelson-Oppen

The state consists of a set of literals. Some are hypotheses and others are negated goals.

What NO tries to prove is

\[ F_1 \land \ldots \land F_n \Rightarrow \bot \]

NO works by discovering equalities,

\[ F_1 \land \ldots \land F_n \Rightarrow E \]

and adding them to the state

\[ F_1 \land \ldots \land F_n \land E \Rightarrow \bot \]

This is just like the "cut" rule in logic.

\[
\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \quad \text{or} \quad \frac{\vdash A}{\vdash B} \]

\begin{proof}
\end{proof}
We want to allow tactics to produce subgoals.

Say that a tactic has discovered a subgoal $G$ such that

$$F_1 \land \cdots \land F_n \land G \Rightarrow \bot$$

What remains to be done is

$$F_1 \land \cdots \land F_n \Rightarrow G$$

Consider the case when $F_1 \land \cdots \land F_n \Rightarrow E$ (an equality is entailed)

This is equivalent to saying that $\neg E$ is a subgoal, i.e.

$$F_1 \land \cdots \land F_n \land \neg E \Rightarrow \bot$$

Thus the prover tries to prove next

$$F_1 \land \cdots \land F_n \Rightarrow \neg E \text{ or equivalently}$$

$$F_1 \land \cdots \land F_n \land E \Rightarrow \bot$$

What happens is that the equality $E$ is announced to everybody just like the normal operation of NO
Consider the case of a non-convex sat. proc.
\[ F_1 \land \ldots \land F_n \Rightarrow \bot \]
and \[ F_1 \land \ldots \land F_n \Rightarrow E_1 \lor E_2 \]
This is equivalent with announcing the subgoal \[ \neg E_1 \land \neg E_2 \]
meaning that
\[ F_1 \land \ldots \land F_n \equiv \neg E_1 \land \neg E_2 \Rightarrow \bot \]
reducing the state to
\[ F_1 \land \ldots \land F_n \Rightarrow \neg E_1 \land \neg E_2 \]
or to the two subgoals.
\[ F_1 \land \ldots \land F_n \land E_1 \Rightarrow \bot \]
and
\[ F_1 \land \ldots \land F_n \land E_2 \Rightarrow \bot \]

- Essentially we end up doing case analysis !!

- This means that the subgoal mechanism generalizes both the equality sharing and disjunction of equalities sharing mechanism

\[ \Rightarrow \text{we add the ability for sat. proc. to announce subgoals, not only equalities.} \]
Modified NO prover

Each sat. proc. can call the function

\[ \text{addsubgoal} : \text{pred} \times (\text{pf}(P) \Rightarrow \text{pf}(\bot)) \rightarrow \text{unit} \]

If we expect a contradiction and we do not get one, we try the subgoals.

\[
\begin{align*}
\text{inv} (F) &= \text{try snapshot} () ; \\
&\quad \text{try assert} (~L, u) ; \\
&\quad \text{try AllSubgoals} (c) ; \\
&\quad \text{raise failure} \\
&\quad \text{handle Contra} (\text{pf}) \\
&\quad \text{contra}(u, \text{pf}) \\
&\quad \text{finally undo} (c)
\end{align*}
\]
try AllSubgoals() =
    while - Subgoal. isEmpty() do
        (sg, pftrans) = Subgoal. getNext();
        try
            raise Contra(pftrans(inv(sg)))
        handle Failure
        continue

Subgoal management

- at any given moment there are several possible subgoals
- which one we choose is unimportant
- each subgoal is a choice
- it is important that we discard subgoals on undo
- a subgoal that fails might succeed after more assumptions are made
  => after a snapshot we might want to retry a subgoal

keep subgoals as a stack of sets of subgoals.

`type subgoals = Stack of (Set of pred × (proof → proof))`
addSubgoal \( (sg, phons) = \)
\[ \text{Stack.replaceTop (subgoals,}
\text{Set.add (Stack.top (subgoals),}
\text{(sg, phons) )))} \]

undo () = Stack.pop (subgoals)

snapshot () = Stack.push (subgoals, Stack.top (subgoals))

getNext () = (sg, phons) = Set.pickOne (Stack.top (subgoals))
\[ \text{Stack.replaceTop (subgoals,}
\text{Set.remove (Stack.top (subgoals),}
\text{(sg, phons) )}) \]
\[ \text{return (sg, phons)} \]

- There is still a choice in how the next subgoal is selected.

- There is a completeness bug in this implementation. Find it!
Using tactics to decide satisfiability

Consider the theory for typing:

\[
\frac{e_1 : \text{int} \quad e_2 : \text{int}}{e_1 + e_2 : \text{int}}
\]

\[
\frac{e : \text{pts}(e) \quad m : \text{mem}}{\text{sel}(m, e) : \mathbb{Z}}
\]

\[
\frac{m : \text{mem} \quad a : \text{pts}(e) \quad e : \mathbb{Z}}{\text{upd}(m, a, e) : \text{mem}}
\]

- We will later consider the question of whether this theory is adequate for typing.
- For now, let's just consider it syntactically.

All these rules are appropriate for backward chaining.

- We have a sat. proc. that collects all assertions \( e : \mathbb{Z} \) or \( \neg(e : \mathbb{Z}) \)

- When it sees \( \neg(e_1 + e_2 : \text{int}) \) produces the subgoal \( \neg(e_1 : \text{int} \land e_2 : \text{int}) \)

- and so on.
Is it possible to generate such a tactic automatically from the inference rules?

Answer: sometimes.
  • when the rules are appropriate for backward chaining.

In general one might want to attach a trigger function to each rule
  • the rule is used only when the conclusion matches a goal and the trigger function returns true.

What if you have a rule

\[
H_1 \ldots H_n \quad \text{where } \text{Var}(H_i) / \text{Var}(C) \neq \emptyset
\]

Then it is useful to have some matching functions that find instantiations of the missing vars.

E.g. \[\text{mem } a : \text{ptr}(e) \quad e : e\]

\[\text{upd } (m, a, e) : \text{mem}\]

need to have a function that finds \( e \) such that \( a : \text{ptr}(e) \) (or \( e : e \))