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 $G_1 \wedge \ldots \wedge G_n \Rightarrow G$

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

$$F_1 \wedge \ldots \wedge F_n \stackrel{?}{\Longrightarrow} \bot$$

NO works by discovering equalities.

$$F_1 \wedge \ldots \wedge F_n \Rightarrow E$$

and adding them to the state

$$F_1 \wedge \ldots \wedge F_n \wedge E \stackrel{?}{\Longrightarrow} \bot$$

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

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \qquad \text{or} \qquad \frac{\vdash A \quad \begin{array}{c}\overline{\vdash A}\\ \vdots \\ \vdash B\end{array}}{\vdash B}$$

We want to allow tactics to produce subgoals.

Say that a tactic has discovered a subgoal G such that

$$F_1 \wedge \ldots \wedge F_n \wedge G \Rightarrow \bot$$

What remains to be done is

$$F_1 \wedge \ldots \wedge F_n \stackrel{?}{\Rightarrow} G$$

Consider the case when $F_1 \wedge \ldots \wedge F_n \Rightarrow E$

(an equality is entailed)

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

i.e. $F_1 \wedge \ldots \wedge F_n \wedge \neg E \Rightarrow \bot$

Thus the prover tries to prove next

$F_1 \wedge \ldots \wedge F_n \stackrel{?}{\Rightarrow} \neg E$ or equivalently

$F_1 \wedge \ldots \wedge F_n \wedge E \stackrel{?}{\Rightarrow} \bot$

- 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.

$$F_1 \wedge \ldots \wedge F_n \stackrel{?}{\Rightarrow} \bot$$

and $F_1 \wedge \ldots \wedge F_n \Rightarrow E_1 \vee E_2$

This is equivalent with announcing the subgoal $\neg E_1 \wedge \neg E_2$

meaning that

$$F_1 \wedge \ldots \wedge F_n \wedge \neg E_1 \wedge \neg E_2 \Rightarrow \bot$$

reducing the state to

$$F_1 \wedge \ldots \wedge F_n \stackrel{?}{\Rightarrow} \neg E_1 \wedge \neg E_2$$

or to the two subgoals.

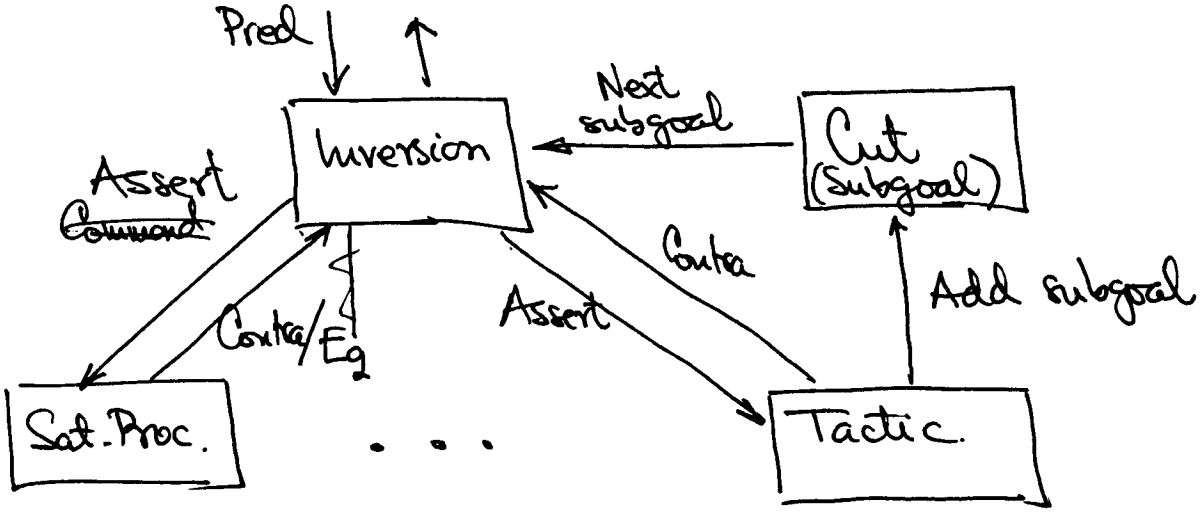
$$F_1 \wedge \ldots \wedge F_n \wedge E_1 \stackrel{?}{\Rightarrow} \bot$$

and

$$F_1 \wedge \ldots \wedge F_n \wedge E_2 \stackrel{?}{\Rightarrow} \bot$$

- essentially we end up doing case analysis !!
- This means that the subgoal mechanism generalizes both the equality sharing and disjunction of equalities sharing mechanisms
  $\Rightarrow$ we add the ability for sat. proc. to announce subgoals, not only equalities.

Modified NO prover



Each sat. proc. can call the function

$$\text{addSubgoal} : \text{pred}_{P:} \times (\text{pf}(P) \to \text{pf}(1)) \to \text{unit}$$

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

```
:
inv(F) = try  snapshot();
              try  assert(¬L, u);
                   tryAllSubgoals();
                   raise Failure
              handle Contra(prf)
                   contra(u, prf)
       finally undo()
```

```
tryAllSubgoals() =
    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 <u>choice</u>
- 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, ptrons) =
    Stack.replaceTop (subgoals,
                      Set.add (Stack.top (subgoals),
                               (sg, ptrons)))

undo () = Stack.pop (subgoals)

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

getNext () = (sg, ptrons) = Set.pickOne (Stack.top (subgoals))
             Stack.replaceTop (subgoals,
                               Set.remove (Stack.top (subgoals),
                                           (sg, ptrons)))
             return (sg, ptrons)
```

- 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

$$\frac{e_1 : int \quad e_2 : int}{e_1 + e_2 : int} \qquad \frac{e : ptr(\tau) \quad m : mem}{sel(m, e) : \tau}$$

$$\frac{m : mem \quad a : ptr(\tau) \quad e : \tau}{upd(m, a, e) : mem}$$

- 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
- have a sat. proc. that collects all assertions $e : \tau$ or $\neg(e : \tau)$
- when it sees $\neg(e_1 + e_2 : int)$ produces the subgoal $\neg(e_1 : int \wedge e_2 : int)$
- 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 ~~not~~ when the conclusion matches a goal and the trigger function returns true.

What if you have a rule

$$\frac{H_1 \ldots H_n}{C} \quad \text{where} \quad Var(H_i) \,/\, Var(C) \neq \emptyset$$

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

E.g. $$\frac{m : mem \quad a : ptr(\tau) \quad e : \tau}{upd(m, a, e) : mem}$$

• need to have a function that finds τ such that $a : ptr(\tau)$ (or $e : \tau$)