Static Checking vs. Theorem Proving

Goal:
- finds bugs at compile-time,
- proves that there is no violation.

Static Checking:
- e.g. Jahob and ESC/Java
- fully automatic (after annotation)
- can only verify simple properties

Theorem Proving:
- Needs lot of manual interaction
- complete calculus, can verify any property.
The Jahob Proof Language

Goals

- Improve the strength of the provable properties.
- Still fully automatic (after annotation).
- Have intermediate proof steps in annotation.

Paper:

We already know one command

\[ \text{note } \ell : F \]

which abbreviates

\[ \text{assert } \ell : F; \text{assume } \ell : F \]

- \( \ell \) is a label (or name) for the formula \( F \)
- When \( F \) cannot be proven Jahob tells that the check for \( \ell \) failed.
- \( \ell \) can also be used to tell the Jahob which formulas are relevant:

\[ \text{assert } G \text{ from } \ell \]

Why is this rule correct?
Correctness of Proof Commands

An incorrect program must not be verified successfully.

If $P \rightarrow wp(S_1; \text{note } F; S_2, Q)$ then $P \rightarrow wp(S_1; S_2, Q)$

This is the case if we can proof that for all $H$

$$wp(\text{note } F, H) \rightarrow H$$

The note $F$ command is correct:

$$wp(\text{note } F, H) \iff wp(\text{assert } F; \text{assume } F, H)$$

$$\iff F \land (F \rightarrow H)$$

$$\iff F \land H$$

$$\rightarrow H$$
Suppose you want to argue that $F$ implies $G$ by a implication chain

$$F \rightarrow F_1 \rightarrow F_2 \rightarrow G.$$ 

In Jahob there is a special syntax:

assuming $F$ in

( note $F_1$
note $F_2$
note $G$)

This command adds the assumption

assume $F \rightarrow G$
General syntax of assuming

The general syntax is

```
assuming F in
  (∧
    note G)
```

This is an abbreviation for

```
( assume F
  ∧
  assert G
  assume false
  □
  assume F → G
  )
```

• : stands for arbitrary proof statements
Correctness of assuming statement

The implication rule is correct, provided the proof statements used in between are correct.

\[ wp((\text{assume } F; \ p; \ \text{assert } G; \ \text{assume false} \ \Box \ \text{assume } F \rightarrow G, H)) \]
\[ \equiv (F \rightarrow wp(p, G)) \land ((F \rightarrow G) \rightarrow H) \]
\[ \rightarrow [\text{assuming that proof statements } p \text{ are correct}] \]
\[ (F \rightarrow G) \land ((F \rightarrow G) \rightarrow H) \]
\[ \rightarrow H \]
Case Splits

One can split cases, e.g.

\[ \text{cases } x \geq 0, x < 0 \text{ for } \text{abs}(x) \geq 0 \]

\[ \text{cases } F_1, \ldots, F_n \text{ for } G \]

is an abbreviation for

- assert \( F_1 \lor \cdots \lor F_n \);
- assert \( F_1 \to G; \ldots \)
- assert \( F_n \to G \);
- assume \( G \)

- Proof that \( F_1, \ldots, F_n \) are all possible cases.
- Proof for each case \( G \) separately.
- Assume \( G \) holds.
To prove a universal quantified formula the syntax is

```
pickAny x
:

note F
```

This is an abbreviation for

```
( havoc x
:

assert F[x]
assume false

assume ∀x. F[x]
)
```
Removing Universal Quantifiers

The inverse operation removes universal quantifiers:

\[ \text{instantiate } \forall x. F[x] \text{ with } t \]

This is an abbreviation for

\[
\begin{align*}
\text{assert } \forall x. F[x] \\
\text{assume } F[t]
\end{align*}
\]
Proving Existential Quantifiers

To prove an existential quantified formula the syntax is

\[ \text{witness } t \text{ for } \exists x. F[x] \]

This is an abbreviation for

\[ \text{assert } F[t] \]
\[ \text{assume } \exists x. F[x] \]
Removing Existential Quantifiers

The syntax is

\[
\text{pickWitness } x \text{ for } F[x]
\]

\[
: \quad \text{where } x \text{ does not occur in } G
\]

\[
\text{note } G
\]

This is an abbreviation for

\[
( \quad \text{assert } \exists x. F[x] \\
\text{havoc } x \\
\text{assume } F[x] \\
:\quad \quad \text{assert } G \\
\text{assume false} \\
\quad \square \\
\text{assume } G \\
) \]