1 Objectives of this Lecture

To understand the basic ideas of discharging proof obligations.
To understand the use of inference rules, for forward or backward inference.
To gain a basic understand the use of the BToolkit BToolProver for discharging proof obligations.

A proof obligation has the form:

\[ \text{hypotheses} \Rightarrow \text{goal} \]

The hypotheses are coded as:

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Objectives of this Lecture</td>
<td>1</td>
</tr>
<tr>
<td>2 Prover Rules</td>
<td>8</td>
</tr>
<tr>
<td>2.1 INHYP</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Deduction (DED)</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Hypotheses (HYP)</td>
<td>8</td>
</tr>
<tr>
<td>2.4 LEMMA</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Proof of disjunction</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Proof by case analysis</td>
<td>9</td>
</tr>
<tr>
<td>2.7 Proof of false goals</td>
<td>10</td>
</tr>
<tr>
<td>3 But the proof rules may be wrong!</td>
<td>10</td>
</tr>
</tbody>
</table>
There are four types of proof obligations:

**Initialisation** This is the proof that the initialisation substitution establishes the invariant, hence the hypotheses consist only of \(\text{cst} \) and \(\text{ctx}\).

**Operation** This is the proof that the operation restores the invariant, and the hypotheses consist of \(\text{cst}\), \(\text{ctx}\), \(\text{inv}\), \(\text{asn}\) and \(\text{pre}\).

**Context** This is the proof that there exist sets and constants that satisfy the properties constraints. The hypotheses consist only of \(\text{cst}\).

**Assertion** The proof that the assertions are a consequence of the invariant, properties and constraints. The hypotheses consist of \(\text{cst}\), \(\text{ctx}\) and \(\text{inv}\).

Instead of using the notation \(\text{hypotheses} \Rightarrow \text{goal}\) we will use \(\text{hypotheses} \vdash \text{goal}\), which denotes a *lemma* not a predicate.

The strategy for discharging (proving) a proof obligation is to apply *inference rules* to the proof obligation until it is reduced to either of

\[\text{hypotheses} \vdash \text{true}\]

or

\[\text{false} \vdash \text{goal}\]

An inference theory has the same form as a proof obligation, but we will use *antecedent* in place of \(\text{hypotheses}\) and *consequent* in place of \(\text{goal}\). Thus the form of an inference rule is:

\[
\begin{array}{c}
\text{antecedent} \\
\Rightarrow \\
\text{consequent}
\end{array}
\]

One significant difference between an inference rule and a proof obligation is that the antecedent and consequent generally contain *jokers*, where the hypotheses and goal contain expressions consisting of variable names and other symbols.

The reason is that a single rule will apply to an infinite number of lemmas, but only if the antecedent *matches* the hypotheses, or the consequent *matches* the goal. The matching will instantiate the jokers to actual expressions.

Jokers are identifiers consisting of a single letter.

Instantiation is a form of substitution. Consider an expression \(E\) containing jokers \(j_1, \ldots, j_n\). Then

\[
[j_1, \ldots, j_n := e_1, \ldots, e_n]E
\]

represents the instantiation of \(E\) in which the jokers \(j_1, \ldots, j_n\) are replaced by the expressions \(e_1, \ldots, e_n\).

Consider the inference rule (expressed in ASCII)
\[ a <: c \land b <: c \]
\[ \Rightarrow \]
\[ a \lor b <: c \]

and suppose the consequent is matched against the goal of hypotheses \( \vdash \) accounts \( \lor \{\text{acc}\} <: \text{ACCOUNT} \)

then the antecedent will be instantiated to

\[ \text{accounts} <: \text{ACCOUNT} \land \{\text{acc}\} <: \text{ACCOUNT} \]

Forward inference matches the antecedents of an inference rule against the hypotheses of the lemma and merges the subsequently instantiated consequent with the hypotheses.

Forward inference attempts to expand the hypotheses in an attempt to generate a hypothesis that will match the goal or a conjunct of the goal.

Example: \( f : X \rightarrow Y \Rightarrow f : X \rightarrow Y \)

Backward inference matches the goal of a lemma against the consequent of an inference rule and replaces the goal by the subsequently instantiated antecedents of the inference rule.

Backward inference attempts to transform the current goal into one or more, hopefully, easier to prove goals.

Note: \( H \vdash G_1 \land G_2 \) can be replaced by \( H \vdash G_1 \) and \( H \vdash G_2 \).

Inference rules can be used as either forward or backward, but most are intended to be used in a particular direction.

Example: \( f : X \rightarrow Y \land x : \text{dom}(f) \Rightarrow f(x) : Y \)

A tactic is a sequence of inference rules, possibly with repetition, to be applied to the proof of a particular lemma.

We will not be dealing with tactics here, other than to observe and explain the tactics that appear in the proof theory files and menus.

In general tactics will involve the application of both forward and backward inference rules, although most use will be made of backward inference rules.

Forward inference rules are “blind”: they expand the hypotheses without reference to the goal, whereas backward inference rules are driven directly from the goal.

For a simple example we will take the Bank machine. All of the proof obligations, except Context, are discharged by the AutoProver, but we will look at doing the proofs interactively.

To follow this example you should follow using the BToolkit.

Apply the toolkit to the Bank development.

Move to the Provers environment.

If any of the proof obligations of the Bank machine have been discharged, select the \texttt{rpl} (Reset Proof Level) button and reset to level 0. There should be 7 proof obligations and 7 remaining to be proved.

Select the \texttt{prv} (Provers) button and choose the \texttt{BToolProver}.

On the Theories menu, select \textit{Initialisation} line, which shows a total of 2 proof obligations and 2 proof obligations remaining to be proved.

On the Theory Menu that is displayed you might want to change \textit{Display all} to \textit{Display unproved only}. Select \textit{Initialisation.1} by left-mouse clicking anywhere within the proof obligation for \textit{Initialisation.1}.  

3
Now click again anywhere in the body of the proof obligation, or select Prove Rule in the Rule Menu. The proof obligation is revealed inside a window as:

\[
\text{cst}(\text{Bank}) \; \& \; \text{ctx}(\text{Bank}) \\
\implies \\
\{\} : \text{ACCOUNT}
\]

This window is interactive; try clicking in it.

The DED (deduction) button will be highlighted in the Proof Menu. Select it. The deduction rule changes

\[H \vdash P \implies Q\]

to

\[H \wedge P \vdash Q\]

If DED is offering you should usually select it.

You will observe that the current goal window now contains

\[H \vdash \{\} <: \text{ACCOUNT}\]

Try clicking on the \(H\). You will get a window showing all the hypotheses. Notice that when you list the hypotheses, all the hypotheses hidden behind the abbreviations \(\text{cst}, \text{ctx}, \text{inv}\), etc are made visible.

Also, notice that any applicable forward rules are applied when DED is selected.

In the current case, notice that forward rule FwdInNat1X.19 has been selected. You can find that rule by selecting Browse Library looking for the 19th rule in FwdInNat1X. This rule reads

\[
\begin{align*}
\text{n} & : \text{NAT1} \\
\implies \\
\text{n} & : \text{NAT}
\end{align*}
\]

In the Proof Menu, the Show all applicable rules will be highlighted.

Select it to see what is offering.

There is one rule

\[
\text{Inclusion X.55} \\
\{\} <: \text{a}
\]

This says that the empty set is a subset of anything.

Notice that the validity of this depends on all expressions being well-typed.

Select the rule by clicking on it.

The proof completes.

Inspect the proof tree.

Select Quit to Proof Menu.

Select Initialisation.2

\[
\text{cst}(\text{Bank}) \; \& \; \text{ctx}(\text{Bank}) \\
\implies \\
\{\} : \{\} \implies \text{ACCOUNT}
\]
and take the proof through its steps

Select Quit to Proof Menu.

No more proof obligations left on this menu.

Select Quit to Main Menu

Select NewAccount and then select NewAccount.1.

Take the proof forward. Note two instances of DED.

Keep going to be offered InclusionX.38 and InclusionX.2. Take the former, and complete the proof.

Proceed to

\[
H \vdash balance \leq \{ acc \mapsto 0 \} \in accounts \cup \{ acc \} \rightarrow \mathbb{N}
\]

and be offered:

\[
\text{InTotalFunctionX.11}
\]

\[
\begin{align*}
\text{binhyp}(f &\in u \rightarrow t) \land \\
u \cup \{ a \} &= s \land \\
b &\in t \\
\Rightarrow \\
f &\leq \{ a \mapsto b \} : s \rightarrow t
\end{align*}
\]

To prove that \( f \leq g : s \rightarrow t \), it is sufficient to prove that \( f \) and \( g \) are total functions, that \( \text{dom}(f) \cup \text{dom}(g) = s \), and that \( \text{ran}(f) \subseteq t \) and \( \text{ran}(g) \subseteq t \).

Thus, we can prove that \( f \leq \{ a \mapsto b \} \in s \rightarrow t \) if we can prove:

1. \( f \) is a total function, say \( f \in u \rightarrow t \);
2. \( u \cup \{ a \} = s \);
3. \( b \in t \)

If the rule in the box in the previous frame is matched against the goal

\[
balance \leq \{ acc \mapsto 0 \} \in accounts \cup \{ acc \} \rightarrow \mathbb{N}
\]

then \( f, b, a, s \) and \( t \) will be instantiated, but \( u \) will not be instantiated.

In InTotalFunctionX.11, the antecedent \( f \in u \rightarrow t \) is contained in the special guard binhyp.

Rather than simply becoming a sub-goal, a guard is evaluated before the rule is applied. binhyp has the first letter \( b \), which identifies this as a guard, followed by inhyp.

binhyp(expression) will attempt to match expression in the current hypotheses and the rule will not be applied unless the match succeeds. As a side effect of the match any uninstantiated jokers will be instantiated.

Thus, in the current example, \( u \) will be instantiated to accounts.

Having selected InTotalFunctionX.11 there are three goals. One is discharged because it is in the hypotheses (INHYP), and the other is an equality and is discharged by equality (EQL). The remaining goal is

\[
0 \in \mathbb{N}
\]

for which we are offered rule InNat.24:
\[ \text{bnum}(n) \implies n \in \mathbb{N} \]

\textit{bnum} is a guard: \textit{bnum}(n) is true if \( n \) is a natural number.

Proceed to

\[ H \vdash \text{balance} \leq \{\text{account} \mapsto \text{balance}(\text{account}) + \text{amount}\} \in \text{accounts} \rightarrow \mathbb{N} \]

and take \textit{InTotalFunctionX}.12 as before.

We get three goals two of which are discharged as they are in the hypotheses (INHYP, and we have one goal

\[ \text{balance}(\text{account}) + \text{amount} \in \mathbb{N} \]

There is one rule on offer, \textit{InNatX}.23,

\[ n \in \mathbb{N} \land p \in \mathbb{N} \implies n + p \in \mathbb{N} \]

Take it.

We now have two goals of which the first is

\[ \text{balance}(\text{account}) \in \mathbb{N} \]

For which we will take the rule \textit{InSetX}.9

\[
\begin{align*}
\text{binhyp}(f \in s \mapsto t) \land \\
n \in \text{dom}(f) \implies f(a) \in t
\end{align*}
\]

Notice the use of \textit{binhyp} again.

When this rule is chosen the proof completes.

Follow the proof through to

\[ \text{balance}(\text{account}) - \text{amount} \in \mathbb{N} \]

for which we are offered \textit{InNat}.19

\textit{InNat}.19

\[ n \in \mathbb{N} \]

\[ p \in \mathbb{N} \]

\[ p \leq n \]

\[ \implies n - p \in \mathbb{N} \]

This gives three sub-goals:

1. \( \text{balance}(\text{account}) \in \mathbb{N} \)
2. \( \text{amount} \in \mathbb{N} \)
3. \( \text{amount} \leq \text{balance}(\text{account}) \)

The first is proved as in frame 26, the second and third are in the hypotheses; the third from the precondition.

The only remaining undischarged proof obligation is a context proof that would not be discharged by the AutoProver.

Take this proof through to

\[ H \vdash \exists \text{ACCOUNT}.(\text{card}(\text{ACCOUNT}) = \text{maxaccount} \land \text{card}(\text{ACCOUNT}) \in \mathbb{N}_1) \]

This obligation comes from our properties constraints and is requiring proof that such a set does exist.

At this stage the BToolProver is not offering any applicable proof rules.
Clicking on H to display the hypotheses shows that $maxaccount \in \mathbb{N}_1$. It is clear that such a set exists.

**Why no proof rules?**

If we browse the BTool theory library we encounter

$$\exists 0 X\text{.}17$$

The consequent is equivalent to our goal, but the conjuncts are swapped. If we can swap them around we should be offered this rule. We can do this with a *rewrite* rule.

Select Edit BTool Library and select BToolUsersTheory, which is for our own backward inference rules. Select Add New Rule and add:

$$(P \land Q) \Rightarrow (Q \land P)$$

Go back to the prover. The Show all applicable rules should be highlighted. If it isn’t you have entered the rule incorrectly.

Select the rule, and then select Exist0X.17 when it is offered.

The BTool prover accepts a number of guards, two of which we’ve met above. Guards are intended for use in the antecedents of backward inference rules, and in all cases the guard will be used to determine the applicability of a rule. A short summary of guards is given below.

- **binhyp** *binhyp(P)* succeeds only if *P* matches a current hypotheses.
- **bnum** *bnum(n)* succeeds only if *n* is a natural number.
- **bident** *bident(a)* succeeds only if *a* is a valid variable identifier.
- **btest** *btest(x rel y)* succeeds only if *x* and *y* are valid numbers and *x* *rel* *y* is true, where *rel* is one of $<,\leq,=,\geq,\geq,\geq$.
- **bstring** *bstring(s)* succeeds only if *s* is a valid quoted string.
- **bsearch** see BToolkit documentation.
- **breade** see BToolkit documentation.

Rewrite rules have the form

$antecedent \Rightarrow expression-1 == expression-2$

This has the understanding that if the antecedent holds then any sub-expression of the current goal matching *expression-1* may be rewritten as *expression-2*.

There is limited backtracking through rewrites so in general the antecedents should be guards.

There may be more than one applicable proof rule.

A proof rule *antecedent* $\Rightarrow$ *consequent* means, “if you want to prove *consequent* then it is sufficient to prove *antecedent*”.

Thus it may not be necessary to prove *antecedent*.

If you choose an inappropriate rule then this may lead to a goal that cannot be proved. In that case, it is necessary to back up to the most recently chosen rule and make another choice, and so on recursively.

The AutoProver tries all applicable rules until either the goal is discharged or all applicable rules are exhausted and the goal has not been discharged.
2 Prover Rules

A box of buttons on the top right of the screen represent rules for guiding the prover. INHYP

2.1 INHYP

INHYP

If INHYP is highlighted then the current hypotheses contain the current goal. Thus selecting this button will trivially discharge the goal.

This button represents the rule: \[ H, P \vdash P \].

2.2 Deduction (DED)

The DED button will be highlighted when the current goal contains implication.

The DED button represents the proof strategy:

\[
\frac{H, P \vdash Q}{H \vdash P \Rightarrow Q}
\]

which can be interpreted as

If the hypotheses are \( H \) and the current goal is \( P \Rightarrow Q \), then add \( P \) to the hypotheses and make \( Q \) the goal.

2.3 Hypotheses (HYP)

Some hypotheses are offered via the HYP button.

Selecting the HYP button, when it is highlighted, will offer hypotheses interpreted at proof rules, as follows.

Equality

A hypothesis of the form \( X = Y \)

can be interpreted as the rewrite proof rule

\[ X \equiv Y \]

Note: this means that equality, which in logic is an equivalence, when interpreted as a proof rule is directional and care must be taken in choosing between \( X = Y \) and \( Y = X \)!

In particular, you should avoid equalities with the same variables inconsistently on the left- and right-hand sides of the equalities.

Universal Quantification

The universal quantification

\[ \forall(z).(P \Rightarrow Q) \]
can be interpreted by the prover as the proof rule

\[
\begin{array}{c}
\Rightarrow \\
\end{array}
\begin{array}{c}
P \\
Q \\
\end{array}
\]

with \( z \) as jokers.

If \( Q \) is of the form \( X = Y \), then the consequent, \( Q \), is interpreted as a rewrite rule with antecedent \( P \).

2.4 LEMMA

The LEMMA button is always highlighted.

Selecting this button causes the current goal and hypotheses to be formulated as a lemma and the goal discharged as far as the current proof is concerned.

This can be used for at least two purposes:

1. to pass a simple lemma to the AutoProver.
2. to defer a difficult proof until later, but discharging the current goal now.

2.5 Proof of disjunction

Proof of a goal of the form \( P \lor Q \) may be achieved by one of the rules:

\[
\begin{array}{c}
\Rightarrow \\
P \\
P \lor Q \\
\end{array}
\begin{array}{c}
Q \\
\Rightarrow \\
P \lor Q \\
\end{array}
\]

but these rules will only succeed when the disjunction is exclusive.

When it is possible for either \( P \) or \( Q \) to be true then the following rule will be found to be more appropriate

\[
\begin{array}{c}
\neg(P) \Rightarrow Q \\
\Rightarrow \\
P \lor Q \\
\end{array}
\]

and the following rewrite rule may be required

\[
\neg\neg P \equiv P
\]

2.6 Proof by case analysis

There are many goals, for example those involving overridden functions where case analysis is required.

Consider the goal \((f \Rightarrow \{a \mapsto b\})(x) \leq m\), using jokers instead of actually variable names.

Clearly, there are two cases: \( x = a \) and \( x \neq a \), so the following rule is useful
\[(x = a \Rightarrow b \leq m) \land
(\neg(x = a) \Rightarrow f(x) \leq m) \Rightarrow
(f \Rightarrow \{a \mapsto b\})(x) \leq m\]

There are an infinity of variations on the above.

2.7 Proof of false goals

Proof of false goals
If you are presented with a goal such as \(1 < 0\) there are two possibilities:

1. the construct from which the proof obligation was derived is inconsistent, or
2. the hypotheses for the current goal are inconsistent

In the second case, which is not uncommon, you need to examine the hypotheses carefully to discover the inconsistency. It may be necessary to back through the last application of DED, allowing you to apply proof rules to reduce the antecedent of the implication to false.

Once the antecedent has been reduced to false, selecting DED will result in INHYP being highlighted.

2.8 Proof Rule Organisation

Proofs are organised in levels. Each level represents a pass through the AutoProver or the BToolProver. This is done in recognition of the fact that discharge of proof obligations will be a multipass process. Generally, proof rules will be associated with each level, and when proving at level \(n\) all proof rules from levels 1 to \(n-1\) will be visible. Proof rules at lower levels than the current level must not be modified, as previous proofs may depend on those proof rules.

Proof levels behave as a stack in that the only way you can return from level \(n\) to level \(m\) \((m < n)\) is to remove all levels \(m \ldots n - 1\).

2.9 Proof Rule Repositories

AutoProver and BToolProver proof rules these rules are stored in the PMD directory. Proof rules are stored under the name of the construct and the proof level. Additionally, a distinction is made between AutoProver proof rules and BToolProver proof rules.

Global proof rules It is possible (see the Provers options in the Options menu) to have a single global proof method file (SRC is suggested as the location) in which are stored proof rules that will be used —in addition to the rules in the PMD directory— across all proof obligations for any construct in the B development. This is useful if there are common structures across different constructs requiring the same proof rules.

In both the above cases proof rules are stored sequentially with semicolon separators. For the AutoProver rules are chosen from the bottom (last) rule towards the top. This establishes a priority between rules.
2.10 Interactive Proof Strategy

There are two interactive provers:

**InterProver** an interactive version of the AutoProver, which gives a trace of its attempts to discharge a proof obligation. The trace is a little hard to understand, but it can be useful in revealing exactly where the AutoProver is having trouble with a proof. New proof rules can be defined and tested within the InterProver. Saving the InterProver proofs is done by running the AutoProver.

**BToolProver** a completely separate prover from the AutoProver/InterProver. When BToolProver proofs are saved a script is written that will be used on a remake to drive the BToolProver. Unfortunately the scripts are dependent to some extent on the construct from which the proof obligations were derived. This can mean that modifications to the construct can invalidate the script.

It is probably best to treat both interactive provers as investigative tools for determining the required proof rules, and then it is most convenient to record those rules as AutoProver rules in order to achieve as much proof as possible via the AutoProver.

### 3 But the proof rules may be wrong!

Yes, that is correct, and it is important that, as far as possible, proof rules should be self-evidently correct.

What is the consequence?

If the proof rules are incorrect then the discharge of any proof obligations using those rules may be invalid.

Does this invalidate the whole B Method? How do we validate the outcomes? Do we resort to testing?

As stated at the beginning of this course, proof is not to be regarded as a “stamp of approval”. The proofs, including all dependent proof rules, can be marked up into documents by the BToolkit. These can be, and should be, read critically. All dependent proof rules should be very carefully checked.

Whether you test or not is a question that is quite independent of proof.

*Extra testing cannot make up for bad proof rules.*