Interactive Formal Verification

10: Structured Proofs

Tjark Weber
(Slides: Lawrence C Paulson)
Computer Laboratory
University of Cambridge
A Proof about “Divides”

\[ b \text{ dvd } a \iff (\exists k. a = b \times k) \]
A Proof about “Divides”

\[ b \text{ dvd } a \iff (\exists k. a = b \times k) \]

We unfold the definition and get...?
A Proof about “Divides”

\[ b \text{ dvd } a \iff (\exists k. a = b \times k) \]

We unfold the definition and get...

an assumption
A Proof about “Divides”

\[ b \text{ dvd } a \iff (\exists k. a = b \times k) \]

We unfold the definition and get...

an assumption

locally bound variables
A Proof about “Divides”

\[ b \text{ dvd } a \iff (\exists k. a = b \times k) \]

We unfold the definition and get...

an assumption

locally bound variables

A messy proof with two subgoals...
Complex Subgoals
Complex Subgoals

• Isabelle provides many tactics that refer to bound variables and assumptions.

• Assumptions are often found by matching.

• Bound variables can be referred to by name, but these names are fragile.
Complex Subgoals

• Isabelle provides many tactics that refer to bound variables and assumptions.

• Assumptions are often found by matching.

• Bound variables can be referred to by name, but these names are fragile.

• *Structured proofs* provide a robust means of referring to these elements by name.
Complex Subgoals

• Isabelle provides many tactics that refer to bound variables and assumptions.

• Assumptions are often found by matching.

• Bound variables can be referred to by name, but these names are fragile.

• *Structured proofs* provide a robust means of referring to these elements by name.

• Structured proofs are typically verbose but much more readable than linear apply-proofs.
A Structured Proof

But how do you write them?
The Elements of Isar
The Elements of Isar

- A *proof context* holds local variables and assumptions of a subgoal.
- In a context, the variables are free and the assumptions are simply theorems.
- Closing a context yields a theorem having the structure of a subgoal.
The Elements of Isar

- A *proof context* holds local variables and assumptions of a subgoal.
- In a context, the variables are free and the assumptions are simply theorems.
- Closing a context yields a theorem having the structure of a subgoal.
- The Isar language lets us state and prove intermediate results, express inductions, etc.
Getting Started

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
```

```
proof (state): step 1

goal (2 subgoals):
1. \( \forall ka. n + k = k \times ka \Rightarrow \exists ka. n = k \times ka \)
2. \( \forall ka. \exists kb. k \times ka + k = k \times kb \)
```
Getting Started

indicating the start of a structured proof

proof (state): step 1

goal (2 subgoals):
1. \( \forall k. n + k = k \cdot k \Rightarrow \exists k. n = k \cdot k \)
2. \( \forall k. \exists b. k \cdot k + k = k \cdot k b \)
The Proof Skeleton

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "\exists m'. n = k * m'"
    sorry
next
  fix m
  show "\exists m'. k * m + k = k * m'"
    sorry
qed

-u-:*** Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting)

Successful attempt to solve goal by exported rule:
  \( (n + k = k \cdot m) \implies \exists m'. n = k \cdot m' \)

Successful attempt to solve goal by exported rule:
  \( \exists m'. k \cdot m + k = k \cdot m' \)

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
The Proof Skeleton

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "\exists m'. n = k * m'"
    sorry
next
  fix m
  show "\exists m'. k * m + k = k * m'"
    sorry
qed

-u-::** Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )

Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) \Longrightarrow \exists m'. n = k * m'

Successful attempt to solve goal by exported rule:
\exists m'. k * ?m2 + k = k * m'
lemma (?k dvd ?n + ?k) = (?k dvd ?n)
-u-::%% *response* All L7 (Isar Messages Utoks Abbrev;)
```

a name for the bound variable
The Proof Skeleton

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "\exists m'. n = k * m'"
  sorry
next
  fix m
  show "\exists m'. k * m + k = k * m'"
  sorry
qed
The Proof Skeleton

- **Assumption:** "(k dvd (n + k)) = (k dvd (n::nat))"
- **Proof:** (auto simp add: dvd_def)
- **Fix:** m
- **Assume:** "n + k = k * m"
- **Show:** "\(\exists m \cdot n = k \cdot m'\)"
- **Sorry**
- **Next**
- **Fix:** m
- **Show:** "\(\exists m' \cdot k \cdot m + k = k \cdot m'\)"
- **Sorry**
- **Qed**

A name for the bound variable.
The Proof Skeleton

- Assumption
- Conclusion
- Dummy proofs
- A name for the bound variable
The Proof Skeleton

- assumption
- conclusion
- dummy proofs

- a name for the bound variable
- separates proofs of goals
The Proof Skeleton

- The Proof Skeleton separates proofs of goals
- Terminates the proof
- A name for the bound variable
- Dummy proofs
- Assumption
- Conclusion

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
fix m
assume "n + k = k * m"
show "\exists m'. n = k * m'"
sorry
next
fix m
show "\exists m'. k * m + k = k * m'"
sorry
qed
```
Fleshing Out that Skeleton

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    sorry
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed

Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) ==> ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
Fleshing Out that Skeleton

inserting a helpful fact
Fleshing Out that Skeleton

labels for facts

inserting a helpful fact
Fleshing Out that Skeleton

labels for facts

more labels

inserting a helpful fact

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
  sorry
  show "\(\exists m'. n = k \cdot m'\)" using 2
  by blast
next
  fix m
  show "\(\exists m'. k \cdot m + k = k \cdot m'\)"
  sorry
qed

Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) \implies \exists m'. n = k * m'

Successful attempt to solve goal by exported rule:
\(\exists m'. k \cdot m2 + k = k \cdot m'\)

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
Fleshing Out that Skeleton

- labels for facts
- more labels
- inserting a helpful fact
- a real proof!
Completing the Proof

`lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed

Sledgehammer: external prover "spass" for subgoal 1:
  "∃m'. k * m + k = k * m'
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
  "∃m'. k * m + k = k * m'
-u-:%%-  *response*  Top l1  (Isar Messages Utoks Abbrev;)
menu-bar Isabelle Commands Sledgehammer
Completing the Proof

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m" 
  have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed

Sledgehammer: external prover "spass" for subgoal 1:
∃m'. k * m + k = k * m'
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
∃m'. k * m + k = k * m'

Completing the Proof

\begin{verbatim}
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  show "\exists m'. n = k * m'" using 2
    by blast

  sorry
qed
\end{verbatim}

Sledgehammer: external prover "spass" for subgoal 1:
\(\exists m'. k \cdot m + k = k \cdot m'\)
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
  atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
\(\exists m'. k \cdot m + k = k \cdot m'\)
Streamlining the Proof

assume 1: "n + k = k * m"
have 2: "n = k * (m - 1)" using 1
by (metis diff_add_inverse diff
show "\exists m'. n = k * m'" using 2
Streamlining the Proof

assume 1: "n + k = k * m"
have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff)
show "\exists m'. n = k * m'" using 2
    by (metis diff_add_inverse diff)
thus "\exists m'. n = k * m'"
Streamlining the Proof

- hence means have — using the previous fact
Streamlining the Proof

- hence means have — using the previous fact
- thus means show — using the previous fact
Streamlining the Proof

- hence means have — using the previous fact
- thus means show — using the previous fact
- There are numerous other tricks of this sort!
Another Proof Skeleton

```plaintext
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "¬ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

Successful attempt to solve goal by exported rule:

```
|m| = 1
```

```
lemma abs_m_1:
  !?m * ?n! = 1 → !?m! = 1
```

*response* All L5 (Isar Messages Utokens Abbrev;)
specify m's type
Another Proof Skeleton

specify m's type
declare a premise separately
Another Proof Skeleton

```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

- specify m’s type
- declare a premise separately
- restricting the range of abs m

Successful attempt to solve goal by exported rule:

```
?m1 = 1
```

```
lemma abs_m_1:
  !m * ?n1 = 1 → !m1 = 1
```
Another Proof Skeleton

specify m’s type
declare a premise separately
restricting the range of abs m
makes the conclusion trivial
Another Proof Skeleton

specify m's type

declare a premise separately

null proof step

restricting the range of abs m

makes the conclusion trivial
Starting a Nested Proof

```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows  "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
    thus "abs m = 1" using 0
      by auto
qed
```

proof (state): step 6

``` goal (1 subgoal):
1. 2 ≤ |m| ⇒ False
```
Starting a Nested Proof

```
lemma abs_m_1: 
  fixes m :: int 
  assumes mn: "abs (m * n) = 1" 
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "¬ (2 ≤ abs m)"
    proof
      thus "abs m = 1" using 0
        by auto
qed
```

proof (state): step 6

goal (1 subgoal):
1. 2 ≤ abs m ⇒ False
A Nested Proof Skeleton

```plaintext
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
      assume "2 ≤ abs m"
      thus "False"
      sorry
    qed
  thus "abs m = 1" using 0
    by auto
qed
```

```
Successful attempt to solve goal by exported rule:
(2 ≤ |m|) ⟹ False
```

```
Auto-saving...done
```
A Nested Proof Skeleton

```isar
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof
  have 0: "m ≠ 0" using mn
  by auto
  have "¬ (2 ≤ abs m)"
  proof
    assume "2 ≤ abs m"
    thus "False"
    sorry
  qed
  thus "abs m = 1" using 0
  by auto
qed
```

Successful attempt to solve goal by exported rule:

\[ (2 ≤ |m|) \implies False \]

have ¬ 2 ≤ |m|
A Nested Proof Skeleton

```plaintext
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
   by auto
  have "¬ (2 ≤ abs m)"
   proof
    assume "2 ≤ abs m"
    thus "False"
      sorry
  qed
  thus "abs m = 1" using 0
   by auto
qed
```

Successful attempt to solve goal by exported rule:

\[ (2 \leq |m|) \rightarrow \text{False} \]

have \(2 \leq |m|\)
A Complete Proof

```plaintext
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows "abs m = 1"
proof -
  have 0: "m ≠ 0" "n ≠ 0" using mn
    by auto
  have "¬ (2 ≤ abs m)"
  proof
    assume "2 ≤ abs m"
    hence "2 * abs n ≤ abs m * abs n"
      by (simp add: mult_mono 0)
    hence "2 * abs n ≤ abs (m*n)"
      by (simp add: abs_mult)
    hence "2 * abs n ≤ 1"
      by (auto simp add: mn)
    thus "False" using 0
      by auto
  qed
thus "abs m = 1" using 0
  by auto
qed
```
A Complete Proof

A chain of steps leads to contradiction
Calculational Proofs

proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1".
  thus "False" using 0

proof (prove): step 11

goal (1 subgoal):
  1. |m| * |n| = |m * n|
Calculational Proofs

form a series of equalities and inequalities
The Next Step

proof

have "~ (2 ≤ abs m)"

proof

assume "2 ≤ abs m"

have "2 * abs n ≤ abs m * abs n"

by (simp add: mult_mono 0)

also have "... = abs (m*n)"

by (simp add: abs_mult)

also have "... = 1"

by (simp add: mn)

finally have "2 * abs n ≤ 1".

thus "False" using 0

proof (prove): step 14

goal (1 subgoal):

1. 'm * n' = 1
The Next Step

refers to the previous right-hand side

proof

have "~ (2 ≤ abs m)"
proof
assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
also have "... = abs (m*n)"
  by (simp add: abs_mult)
also have "... = 1"
  by (simp add: mn)
finally have "2 * abs n ≤ 1"
thus "False" using 0

proof (prove): step 14

goal (1 subgoal):
  1. \( m \cdot n \) = 1
The Internal Calculation

have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1".
  thus "False" using 0

calculation: 2 * inl ≤ 1
The Internal Calculation

Isabelle displays the internal calculation when it encounters also and finally.
The Internal Calculation

Isabelle displays the internal calculation when it encounters also and finally

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n" by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1".
  thus "False" using 0
```

calculation: 2 * int ≤ 1
Ending the Calculation
Ending the Calculation

We have deduced $2 \times \text{abs } n \leq 1$.
Ending the Calculation

We have deduced $2 \times \text{abs } n \leq 1$ indicates a trivial proof.
Structure of a Calculation
Structure of a Calculation

- The first line is have/hence
Structure of a Calculation

- The first line is have/hence
- Subsequent lines begin, also have “... = “
Structure of a Calculation

- The first line is have/hence
- Subsequent lines begin, also have “… = “
- Any transitive relation may be used. New ones may be declared.
Structure of a Calculation

- The first line is have/hence
- Subsequent lines begin, also have “... = “
- Any transitive relation may be used. New ones may be declared.
- The concluding line begins, finally have/show, repeats the calculation and terminates with (.)