

# Beyond Script Management

---



David Aspinall  
School of Informatics  
University of Edinburgh

Christoph Lüth  
DFKI Bremen



# **Script Management: The Story So Far**

# Script Management: Recap

```
text {* Proper proof text -- \textit{advanced version}. *}

theorem "A & B --> B & A"
proof
  assume "A & B"
  then obtain B and A ..
  then show "B & A" ..
qed

text {* Unstructured proof script. *}

theorem "A & B --> B & A"
  apply (rule impI)
  apply (erule conjE)
  apply (rule conjI)
  apply assumption
  apply assumption
done

--:-- Example.thy 53% L38 CVS:8.3 (Isar script Scripting)-----
[]
proof (prove): step 3

goal (theorem, 2 subgoals):
1. [! A; B !] ==> B
2. [! A; B !] ==> A

--:-- *goals* All L1 (proofstate)-----
```

# Script Management: Recap

The screenshot displays the Proof General Eclipse IDE interface. The main editor shows the file `PER.thy` with the following content:

```
x ≅ y ==> f x ≅ g y
unfolding eqv_fun_def by blast

text {
  eqv_fun_def
  f ≅ g ≡ ∀x∈domain. ∀y∈domain. x ≅ y → f x ≅ g y
  The class of partial equivalence relations is
  closed under function spaces (in \emph{both}
  argument positions).
}

instance "fun" :: (partial_equiv, partial_equiv)
  partial_equiv

proof
  fix f g h :: "'a::partial_equiv ⇒
    'b::partial_equiv"
  assume fg: "f ≅ g"
  show "g ≅ f"
  proof
    assume gh: "g ≅ h"
    show "f ≅ h"
  proof
    fix x y :: 'a
    assume x: "x ∈ domain" and y: "y ∈ domain"
    and "x ≅ y"
```

The `Prover Output` window shows the current proof state:

```
proof (state): step 21
goal (1 subgoal):
1. Ax y. [[x ∈ domain; y ∈ domain; x ≅ y]] ⇒ f x ≅ h y
```

The `Proof Object` window on the right shows the selected theorem:

```
Type: theorem
Theory: all theories
Matching: *equiv*

Equiv_Relations.UN_equiv_class
Equiv_Relations.UN_equiv_class2
Equiv_Relations.UN_equiv_class_injec
Equiv_Relations.UN_equiv_class_type
Equiv_Relations.UN_equiv_class_type
Equiv_Relations.comp_equiv1
Equiv_Relations.eq_equiv_class
Equiv_Relations.eq_equiv_class_iff
Equiv_Relations.eq_equiv_class_iff2
Equiv_Relations.equiv.intro
Equiv_Relations.equiv.refl
Equiv_Relations.equiv.sym
Equiv_Relations.equiv.trans
theorem
Equiv_Relations.equiv.intro
[[refl A r; sym r; trans r]] ⇒
equiv A r
```

The `Outline` window on the left shows the project structure:

- Partial equivalence relations
- theory PER imports Main begin
- Partial equivalence
- Equivalence on function spaces
- Total equivalence
- Quotient types
  - typedef 'a quot = "{x. a ≅
  - quotI (lemma quotI [intro]:
  - quotE (lemma quotE [elim]:

# Script Management: History

- Beginnings: unclear
- Bertot et al: CtCoq
- Aspinall et al: Proof General
  - First with a toolbar
- Proven technology

# Script Management: History

- Beginnings: unclear
- Bertot et al: CtCoq
- Aspinall et al: Proof General
  - First with a toolbar
- Proven technology
- Easy to understand & use, close to prover
- . . . but is it really best we can do?

# Script Management: Disadvantages

- Highly **linear**
  - Forced to work **one error at a time**
  - Interface **directs** user's attention
- Confusing **undo**
  - Theorem history **discarded** upon completion
  - **Nested** proofs?
  - **End** of theories?
- Too **restrictive**
  - Background colouring is **invasive**
  - Locking is **frustrating**
  - **Explorative** proof development restricted

# **Beyond Script Management: Four Ideas**



# Eager Script Management, Lazy Locking

- Do **not** colour, instead **highlight** next command
- Process next command **behind the scenes**
- See **good/bad** effect by marker highlight
- Examine **detail** by **hover** or **forward**

# Eager Script Management, Lazy Locking

The screenshot shows the Proof General Eclipse IDE interface. The main editor displays a theorem prover script for PER.thy. The script includes assumptions, a goal, and a proof structure with subgoals and lemmas. A tooltip is visible over a subgoal, showing its state and goal.

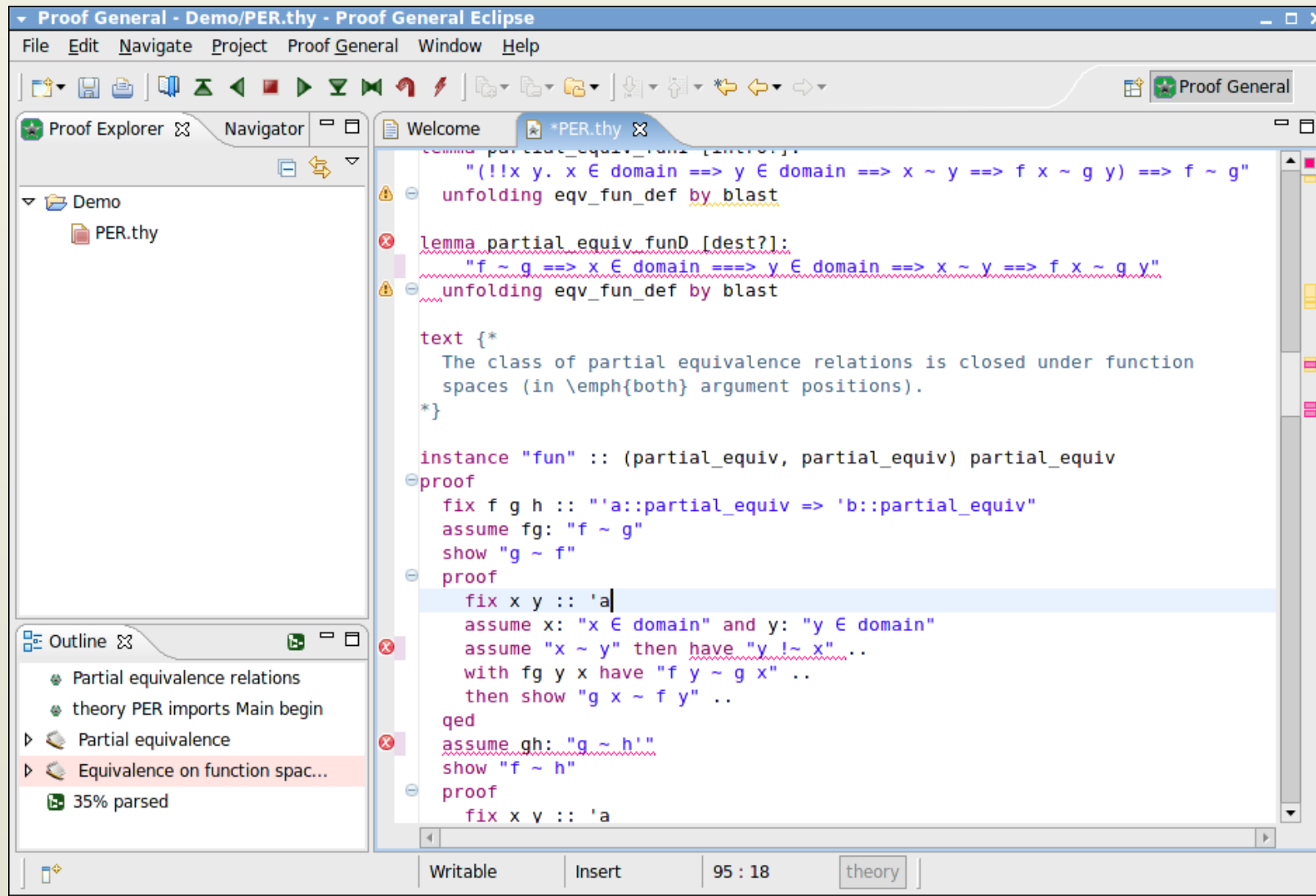
```
assume fg: "f ~ g"
show "g ~ f"
proof
  fix x y :: 'a
  assume x: "x ∈ domain" and y: "y ∈ domain"
  assume "x ~ y" then have "y ~ x" ..
  proof (state): step 8
  this:
  x ~ y
  goal (1 subgoal):
  1.  $\forall x y. [x \in \text{domain}; y \in \text{domain}; x \sim y] \implies g x \sim f y$ 
  show "..."
  proof
    fix x y :: 'a
    assume x: "x ∈ domain" and y: "y ∈ domain" and "x ~ y"
    with fg have "f x ~ g y" ..
    also from y have "y ~ y" ..
    with gh y y have "g y ~ h y" ..
    finally show "f x ~ h y" .
  qed
qed
subsection {* Total equivalence *}
```

proof (state): step 8  
this:  
x ~ y  
goal (1 subgoal):  
1.  $\forall x y. [x \in \text{domain}; y \in \text{domain}; x \sim y] \implies g x \sim f y$

# Whole-Document Editing

- Do not stop at **first** error
- Use `<postponegoal>` and **proceed**
  - Prover should record **proof obligation**
- How to deal with **dependencies**?
  - Maybe not at all: **reprove** when **necessary**

# Whole-Document Editing



The screenshot displays the Proof General Eclipse IDE interface. The main editor window shows a theorem prover document with the following content:

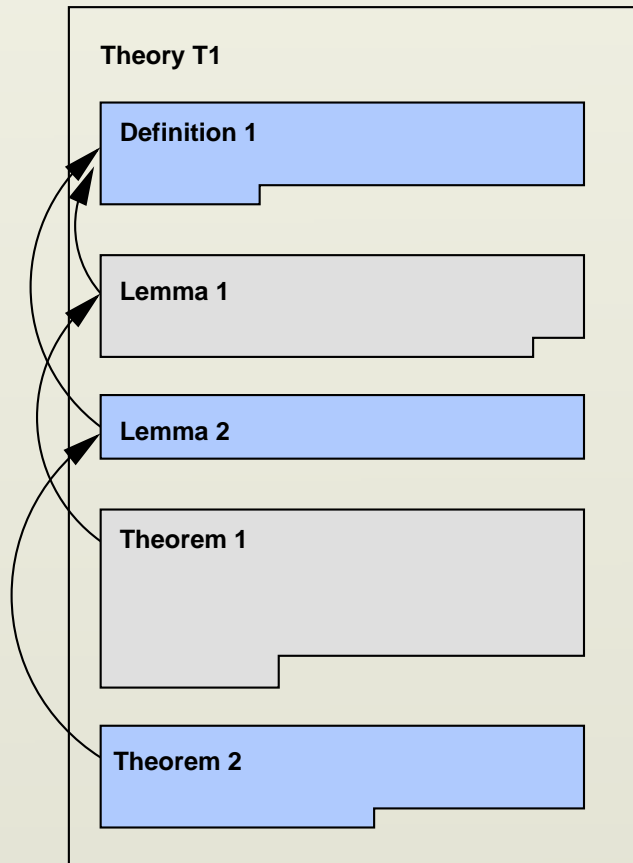
```
lemma partial_equiv_funD [general?]:  
  "(!!x y. x ∈ domain ==> y ∈ domain ==> x ~ y ==> f x ~ g y) ==> f ~ g"  
  unfolding eqv_fun_def by blast  
  
lemma partial_equiv_funD [dest?]:  
  "f ~ g ==> x ∈ domain ==> y ∈ domain ==> x ~ y ==> f x ~ g y"  
  unfolding eqv_fun_def by blast  
  
text {*  
  The class of partial equivalence relations is closed under function  
  spaces (in \emph{both} argument positions).  
*}  
  
instance "fun" :: (partial_equiv, partial_equiv) partial_equiv  
proof  
  fix f g h :: "'a::partial_equiv => 'b::partial_equiv"  
  assume fg: "f ~ g"  
  show "g ~ f"  
  proof  
    fix x y :: 'a  
    assume x: "x ∈ domain" and y: "y ∈ domain"  
    assume "x ~ y" then have "y !~ x" ..  
    with fg y x have "f y ~ g x" ..  
    then show "g x ~ f y" ..  
  qed  
  assume gh: "g ~ h"  
  show "f ~ h"  
  proof  
    fix x y :: 'a
```

The interface includes a menu bar (File, Edit, Navigate, Project, Proof General, Window, Help), a toolbar with various icons, and a sidebar with a Navigator and Outline view. The Outline view shows a tree structure of the document's content, including "Partial equivalence relations", "theory PER imports Main begin", "Partial equivalence", "Equivalence on function spac...", and "35% parsed". The status bar at the bottom indicates "Writable", "Insert", "95 : 18", and "theory".

# Non-linear dependencies

- #1 complaint about script management: **it's too linear**
  - Must **step** through proof from **top** to **bottom**
  - To work on **later stages**, must **process early**
  - To **change early**, must **undo later**
- **Plan:**
  - Use **approximate** dependency reporting from theorem prover
  - Augment with user-level **annotations**
  - Script management uses **independent states** for **regions** (already in PGIP display protocol)

# Non-linear dependencies



Explicit dependencies:

- Requires **accurate** prover assistance
- Better **change** management

## Alternative paths in proofs

- Allow easy **exploration** (and recording) of **alternatives** in proof scripts
- Add `<alternatives>` element with **named** subtrees.
- Add **marker** indicating **processed** state of sub-tree
- Indicate **visually** in the interface
  - notion of **active** choice
  - does **some** successful proof path exist?
  - are **all** proof paths correct (ideal)?
- **Export** operation to **flatten** active choices

# Conclusions



# Conclusions

- Four **incremental** and **achievable** extensions to usual script management
- Work with **existing** provers **out-of-the-box**

# Conclusions

- Four **incremental** and **achievable** extensions to usual script management
- Work with **existing** provers **out-of-the-box**
- **Alternative** interface metaphors exist:
  - **graphical** presentation: **graph**, **tree**, **box**; desktop direct manipulation
  - whole document/constant validation
    - ▷ most **obvious** & possibly most **productive** (cf WYSIWYG)
    - ▷ disadvantage: **short** UI opn → **long-running** prover opn
    - ▷ **tight** prover integration required