Tinycals: step by step tacticals

Claudio Sacerdoti Coen <sacerdot@cs.unibo.it>
Enrico Tassi <tassi@cs.unibo.it>
Stefano Zacchiroli <zacchiro@cs.unibo.it>

University of Bologna

21/08/2006

Objective

User-friendly structured procedural scripts

Cfr. Structured (i.e. syntax oriented) script editing

- Takahashi, Hagiya. “Proving as editing HOL tactics”
- Syme’s TkHOL

Objective

User-friendly structured procedural scripts

Cfr. Structured (i.e. syntax oriented) script editing

- Takahashi, Hagiya. “Proving as editing HOL tactics”
- Syme’s TkHOL

Outline

CtCoq/Proof General Interaction Mode

- An editable script window
- A sequents window (for the current state)
- Commands executed atomically and one at a time
- Already executed commands are locked

Declarative Proof Languages (Mizar/Isar/...)

- Commands are meaningful to the user
- The sequents window is not that useful
- The “script” is fully structured by delimited blocks
  - e.g.: `show P ... done`
  - e.g.: `per cases on n case O ... case S ... done`
- The structure reflects (is?) the proof tree

Procedural Proof Languages (LCF/Coq/Isabelle/...)

- Commands are meaningful to the system
- The sequents window is fundamental
- The script is not naturally structured
  - e.g. `induction n.` `reflexivity.` `intros.` `rewrite H.` `auto.` `assumption.`
- The structure does not reflect the proof tree

Metavariables and Side Effects (1/2)

- Formulae can contain metavariables
- A metavariable is a non-linear and typed placeholder
  - e.g. $\forall$`x,y.?n[x]` $\leq$ `S x` $\wedge$ `P ?n[x]`
  - associated sequent: `x:nat` $\vdash$ `?n : nat`
- Commands (e.g. tactics) can instantiate metavariables
  - e.g. `x:nat` $\vdash$ `?n : nat := S x`
- Instantiation acts on every sequent/branch (side effect)
- Sequents to prove are also metavariables (Curry-Howard)

Metavariables and Side Effects (2/2)

- Proof branches are not independent (and cannot become lemmas)
  - Cfr. Takahashi, Hagiya. "Proving as editing HOL tactics"
- Tactics acting on different sequents cannot be permutated
  - e.g. when the first sequent is `?n[x] = O` is closed by reflexivity and the second sequent `P ?n[x]` is automatically closed
- Sometimes sequents must be addressed in strange orders to drive automation
  - Cfr. Syme's TkHOL (structured editing of HOL scripts by juxtaposition of subscripts)

Metavariables and Side Effects: Why?

- Metavariables and side effects difficult to handle
- Are they useful/necessary? (interesting question)
- We do not care: we want to address the most difficult case

LCF Tacticals

- Higher order tactics: sequencing, branching, repetition, error recovery
- Used to form atomic tactics
- Make the script more robust and more synthetic (code factorization)
- Debugging is an issue
- Sequencing and branching primary way to machine understandable script structuring
- A fully structured script is an atomic tactic

LCF Structured Scripts: UI Issues

A fully structured script is an atomic tactic

- Difficult and time consuming to write
  1. Write a non-structured script;
     make it structured if possible (side effects)
  2. Add a tactic; execute the atomic script; undo;
     repeat until finished
- Difficult to replay
  - Impossible to statically de-structure it
    - e.g. `T1;T2` becomes `T1.  T2.  ...  T2.`
  - De-structure it bit by bit inserting execution points
- Difficult to debug
  - It fails atomically
  - Errors reported on hidden states

LCF Structured Scripts: No, Thanks (1/2)

- Coq/Isabelle/. . . scripts usually not structured with LCF tacticals
- Indentation/blank lines used to structure the script
- Users are happy. . .

LCF Structured Scripts: No, Thanks (2/2)

- until they change the order of hypotheses in a lemma!
- until they change the order of fields in a structure/record!
- until they change the order of constructors of an inductive definition!
- ...
- No help by the system
- We propose a solution that
  - Is fully backward compatible
  - Does not force the user to abandon their style
  - No additional burden to write structured scripts; some advantages
  - Try it once, you won't do without it!

Matita Tinycals: branching and sequencing

- Branching and sequencing can be expressed with more fine grained sequential atomic operations (tinycals)
  - `T1 ; T2` becomes `T1` `;` `T2`
    - After `;` two sequents are selected at once
    - Tactics are executed in sequence on every selected sequent
  - `T1 ; [ T2 | T3 ]` becomes `T1` `;` `[` `T2` `|` `T3` `]`
- Each tinycal can be parsed and executed immediately
- Requires an enriched proof status

More Tinycals: unstructured editing and accept

- Embedding of unstructured script fragments allowed
  - i.e. `.` is a tinycal
  - e.g. `T1` `;` `[` `T2` `.` `T3` `.` `T4` `|` `T5` `]`
- Branches closed by side effects aknowledged by the user to preserve the correspondence with the proof tree
  - i.e. `accept` is a tinycal
  - e.g. `T1` `;` `[` `T2` `|` `accept` `]`

More Tinycals: out of order execution

- Out of order execution of (multiple) branches
  - the tinycal `n1,...,ni:` selects inner branches by position
  - e.g. `T1` `;` `[` `2:` `T2` `|` `3:` `T3` `|` `T4` `]`
  - special case: `*:` selects all the remaining inner branches
  - user not obliged to close the branch before moving to the next ones
- Out of order execution of far away branches
  - `focus` $n_1, \ldots n_i$ ... `unfocus`
  - sometimes necessary for side effects
  - the user is obliged to close the selected branches

Are There More Tinycals?

- Tinycals execution is efficient: tactics are not executed twice
- This is a constraint on the semantics of the tacticals to be mimicked
  - e.g. sequencing can be implemented with tinycals since it is left associative: `T1;T2;T3` = `(T1;T2);T3`
  - e.g. the right associative variant cannot be implemented efficiently using tinycals (because of side effects)
- Repetition and error recovery (`try`, `first`, `OrElse`) cannot be split into tinycals
- We allow atomic LCF tacticals as special cases of tactics

A Note on `try/OrElse`

`try/OrElse` used

- Inside a repetition tactical (rare)
- After sequencing to apply a tactic only to some goals (frequent)
  - e.g. `elim n; (trivial || (simplify ; try auto))`
    some sequents (which ones?) are trivial; the other ones are simplified and solved automatically if possible (which ones?)
- The frequent case is handled by selection of multiple branches
- e.g. `elim n` `;` `[` `1,3:` `trivial` `|` `2:` `simplified` `|` `*:` `simplified` `;` `auto` `]`

Further Considerations

- Code is not duplicated!
    - LCF tacticals still necessary to implement tactics
    - They can be implemented on top of tinycals to avoid code duplication and semantics mismatch
    - Not trivial: tactics work on a poorer proof status
    - Requires a parametric implementation of tinycals on abstract proof statuses with embedding/projections
- We provide a small steps formal operational semantics
    - Look for it in the paper
- A procedural proof language can be implemented more easily on top of tinycals
    - Because the proof status has been enriched
    - Tinycals reduce the gap between procedural and declarative implementations

Conclusions

- LCF tacticals quite bad for proof structuring
- LCF tacticals quite bad with metavariables and side effects
- This is an user interface issue!
- We propose fine grained atomic tinycals that destructure LCF tacticals
- We put some care on the issue of side effects
- We provide a formal semantics and an efficient implementation without code duplication
- We show that the work cannot be extended any further