Dependencies in Formal Mathematics: Applications and extraction for **Coq** and **Mizar**

Jesse Alama[1], Lionel Mamane, Josef Urban[2]

[1]Center for Artificial Intelligence, New University of Lisbon, Portugal

[2]Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands

MKM 2012, Bremen, 2012-07-09

Outline

- Senses of dependency
- Extracting/eliciting fine-grained dependency info
  - For **Mizar** and the **Mizar** Mathematical Library
  - For **Coq** and the Constructive **Coq** Repository at Nijmegen (CoRN)
- Exploiting dependency info
  - Editing support
  - Ensuring library coherence in the face of changes to its contents
  - Training automated theorem provers
  - Speeding up dependency extraction

Dependencies: many senses

Possible definitions: A *depends on* B iff:

- *All* proofs of A *need* (use, pass through, require) B
- *This* proof/definition of A *uses* B
- *This* proof/definition of A *tries to use* B
- Evaluating this proof script (of A) will error out in the absence of B
- If B's statement/type is changed, this proof/definition of A may become invalid (while keeping the same semantics)
- If B's proof/body is changed, this proof/definition of A may become invalid (while keeping the same semantics)
- If B's type is changed, this definition of A changes semantics
- If B's body is changed, this definition of A changes semantics

What depends on what?

Definition (Needs)

A definition/theorem T *depends* on some definition/lemma/theorem T', (or equivalently, that T' is a *dependency* of T) if T "needs" T' to exist or hold.

- Well-formedness/justification for/provability of T fails in the absence of T'.

Dependencies: harder than you think (**Coq**)

- But for **Coq**, dependencies are easy: *walk the λ-term!*
- **No:** For items A and B, in the absence of B, A's proof script has another behaviour:

```
try solve apply B;
solve apply C.
```

- What can B be?
  - Definition / lemma
  - user-defined tactic: not visible in λ-term
  - parametrisation of a tactic, e.g. search depth, morphism declaration, ...
    Need knowledge about each and every tactic!
  - parser / lexer parametrisation: notation; hook into lexer/parser?

Dependencies: harder than you think (**Mizar**)

- Explicit dependencies are easy to gather
- If a proof of statement `A` looks like

  ```
  A by Def1, Lemma2, Theorem5
  ```

  then this step obviously depends on `Def1`, `Lemma2`, and `Theorem5` for some senses, but not necessarily for others
- Much dependency information is *implicit*: cannot be computed from the text alone
- Typing of terms in **Mizar**'s type hierarchy (functions are relations; a field is a group; etc.) are not explicit
- A number of equations may be implicitly available when doing reasoning, e.g. $\forall X(X - \emptyset = X)$

Mizar

- **Mizar** is a proof assistant/interactive theorem prover built on classical first-order logic and set theory.
- Declarative, natural deduction-tyle proof language.
- One directly writes the proof (no tactics).

Simple **Mizar** proof

```
theorem
for X being set holds X is empty implies X = {}
proof
  let X be set;
  assume not ex x st x in X;
  then x in {} iff x in X by Def1; :: def. of {}
  hence thesis by TARSKI:1; :: extensionality
end;
```

Dependency extraction for **Mizar**

- Every **Mizar** article (unified collection of definitions, theorems, proofs, etc.) has its own environment.
- The environment is a (very) conservative overestimate of all items on which the article depends.
- Split up every article of the **Mizar** Mathematical Library into its constituent items, then minimize (brute-force) its environment.
- Minimization in the sense: what does *this proof* depend upon? Unique minimal set of needed items.
- Potential incompleteness: dependencies are extracted up to the level of **Mizar** automation.

Coq: coverage of dependencies

Dependencies on logical constructs.
Dependency logical or non-logical.

Coq: theoretical structure

- Curry-Howard-de Bruijn isomorphism: **pCIC**
- Statement-as-type
- theorem = definition = name $\rightarrow$ (type, body)
- axiom = parameter = name $\rightarrow$ type

Coq: tactic command structure

- parsing
- Ltac (domain-specific programming language) evaluation expression tree; nodes: OCaml tactics
- OCaml tactics evaluation expression tree; nodes: atomic tactics (`refine`, `intro`, `...`)

Result of each step kept in *proof tree*: can inspect there

- arguments passed to tactics
- references pulled by user-defined tactics
- references pulled by OCaml tactics

Coq: proof script structure

Command types:

- Register new logical construct from scratch.
  `Definition Name : type := body.`
- Start a new theorem.
  `Theorem Name : type.`
- Register finished in-progress proof
  `Qed.`
- Make progress in proof
  tactics

Hook into common functions these commands eventually call.

Coq: hook into commands

- Register new logical construct from scratch.
  Walk over type & body, collect references.
- Start a new theorem.
  Walk over type, collect references.
- Register finished in-progress proof
  Walk over body, collect references.
- Make progress in proof
  Walk over top node of proof tree, all three levels.

Interactive front-end: **tmEgg**

- Allow interleaving work on different theorems
- When asked “load this theorem”, load only necessary lemmas
- When changing loaded theorem, invalidate those that use it
  (environment well-formed)

Library recompilation, I: **Coq**

Change in a definition or lemma: is the library still consistent?

- Want to do the least work possible: recheck only reverse dependencies.
- State-of-the-art: per-file dependencies
  Recompile whole file when any item inside depends on any *(un)changed* item in changed file.
- Our contribution: fine-grained per-item dependencies
  Result: recompile *whole file* whenever *any item* inside depends on *changed* item
- With system support: recompile *only* those items that depend in changed item

MathWiki Project ar RU Nijmegen

Coq/CoRN `http://mws.cs.ru.nl/cwiki`

Mizar `http://mws.cs.ru.nl/mwiki`

Library recompilation: gains

	CoRN/item	CoRN/file	MML-100/item	MML-100/file
Items	9 462	9 462	9 553	9 553
Deps	175 407	2 214 396	704 513	21 082 287
TDeps	3 614 445	24 385 358	7 258 546	34 974 804
P(%)	8	54.5	15.9	76.7
ARL	382	2 577.2	759.8	3 661.1
MRL	12.5	1 183	155.5	2 377.5

Deps Number of dependency edges

TDeps Number of transitive dependency edges

P Probability that given two randomly chosen items, one depends (directly or indirectly) on the other, or vice versa.

ARL Average number of items recompiled if one item is changed.

MRL Median number of items recompiled if one item is changed.

Learning dependencies

- One can train a machine learner with dependency info.
- Automated theorem proving (ATP) in mathematics. Prove theorems from earlier knowledge.
- Premise selection problem: of many available (and logically admissible) premises, which should we use?
- Given a **Mizar** item, use the fine-grained dependency info to train a machine learner.
- Result: improvement of ATP performance compared to unassisted premise selection by about 40–50%.

Dependencies aiding dependencies

- Dependency extraction for **Mizar** is very expensive: brute-force search taking **weeks** for the entire **Mizar** library.
- Knowing the exact dependencies for all items logically preceding *I* can help significantly in extracting the dependencies of *I*.
- Machine learner can provide a good guess. Use exact information to train.
- Speedup: roughly 2x.

Conclusion and future work

- Multiple senses of 'dependency' in formal mathematics are available.
- Extracting dependencies can be quite complex. But big rewards can be reaped.
- We have done an analysis and extraction of dependencies for **Coq** (specifically, the CoRN library) and for **Mizar**.
- Future work:
  - Speed up dependency extraction (ideal: more-or-less live dependency extraction. Web interface.)
  - Handle non-logical objects: tactics, notations, ...
  - Trial by fire in mature applications
    - interactive UI front-end ($\approx$ **tmEgg**)
    - library-wide fast rechecking / recompilation ($\approx$ MathWiki)
  - Support in systems to make better use of deps