Lego and related work*

Zhaohui Luo
Department of Computer Science
University of Durham

August 26, 1999

In these lectures, I shall give an introduction to the Lego proof development system, and briefly describe some of the work related to Lego. This is not meant to be a complete summary in any sense, but only serves the purpose of illustration. The references, not complete either, may serve as pointers to the work mentioned and to some other related work.

1 What is Lego

- The Lego proof system
  - Lego is a proof checker and a proof development system [19, 24].
  - Lego implements expressive type theories ECC [12] and UTT [15, 7].
  - Lego has been used in formalisation of mathematics, program specification/verification, and other applications.
- The development of the system
  - Lego was developed at Edinburgh in projects led by Rod Burstall, and implemented by Randy Pollack and others. See Lego's home page `http://www.dcs.ed.ac.uk/home/lego/`.
  - Lego has been used in a lot of related research (including many PhD and MSc work), in projects at Edinburgh, Durham, Göteborg, Aarhus, Manchester, Nijmegen, Erlangen, Munich, Darmstadt, and Ulm, and in undergraduate and MSc teaching.

2 A brief introduction to Lego

- The structure of UTT, the type theory implemented in Lego:
  - Separation of data types and logical propositions.
  - Inductive types and predicative universes (cf, Martin-Löf's type theory [23]).
  - Higher-order logic (cf, the calculus of constructions [6])
- Syntax – the expression language (cf, Lego reference card):
  - Π-types, λ-abstraction and applications: `{x:A}B, A->B, [x:A]b, (f a)`.
  - Inductive types: macro `Inductive` with options such as `Theorems`, `Relation`, `Inversion`, `Double`, etc. For example (also see examples like the less-than relation in exercises):

```
Inductive  [List : Type] Theorems
Parameters [A : Type]
Constructors [nil  : List]
             [cons : A->List->List];
```

*Lecture notes for Types Summer School'99: Theory and Practice of Formal Proofs, Giens, France, 1999.

- Predicative universes (with 'typical ambiguity'): `Type(i), Type`.
- Logical universe (impredicative, giving HOL): `Prop`.
- Local definitions: `[x=a]b`.
- Argument synthesis: `{x|A}B, [x|A]b`.

- Syntax – the command language of Lego (cf, Lego reference card):
  - Commands for context manipulation: eg, `Ctxt, Decls`.
  - Global definition: `[x=a]`.
  - Commands for proof development: eg, `Goal, intros, Refine, Induction, Qrepl, Save, Undo`.
  - The module mechanism: `Module module-name Import module-name-list;` and the associated commands (eg, `Make; Load`.)

- Some other features of the system:
  - Meta-variables.
  - Experimental features such as user-entered computation rules (eg, this allows experiments on non-standard type theory in Lego).
  - Inversion and pattern-matching (recent work by Conor McBride [21]).
  - Simple tacticals.

3 Work closely related to Lego

- Development of expressive type theory:
  - Work on ECC and UTT (see eg, [15])
  - Typed operational semantics [7, 8]
  - Coercive subtyping [16, 29] (cf, implementation of coercions in Lego [2] and in Coq [27])
  - Extensionality and related issues [9]

- Application examples:
  - Formalisation of Pure Type Systems [3] and verification of proof checker [24, 22, 25]
  - Verification of functional programs [4] imperative programs [11, 28], and concurrent programs [30].
  - Model checking in Lego [31]
  - SN proof of the system F [1]
  - Formalisation of synthetic domain theory in Lego [26]

- Development of representation schemes/methods:
  - Program specification/verification and data refinement [14]
  - Methods in formalisation of names and binding operators [24]
  - Internal theory mechanism and development of abstract mathematics [13, 2]
  - Representation mechanisms provided by coercive subtyping (eg, overloading, reasoning with inductive subtypes)
  - Many others: eg, formalisation of algebraic/imperative concurrent languages, the modal u-calculus, and notions such as bisimulation [30].

4 Some recent or on-going development at Durham

- Plastic [5]: a proof system being developed at Durham (see the web page of Computer-Assisted Reasoning Group: `http://www.dur.ac.uk/CARG/`)
- Coercive subtyping: eg, dependent coercions [20].
- Mathematical vernacular and natural language semantics [18, 17]
- Functional programming with dependent types [10]
- Manifest record types (on-going work by Pollack etc at Durham.)

5 The Lego environment and some simple exercises

The Lego system (the current version 1.3.1) is better used in the ProofGeneral environment (see `http://zermelo.dcs.ed.ac.uk/ proofgen/index.phtml?page=doc`.) Also, please consult the reference card of Lego. (Some other Lego documents, like the manual and its incremental changes are provided in the machine room of the summer school.)

For beginners, the following simple exercises may be helpful for you to start:

- Simple examples in logic: Import `lib_logic`, and choose your favourite (intuitionistic) tautologies as goals to prove.
- Import `lib_nat` and get familiar with natural number inductions.
- Do the exercises in the directory `/usr/local/Lego-exercises` (read the file `README` first.)

References

[1] Th. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, Edinburgh University, 1993.

[2] A. Bailey. *The Machine-checked Literate Formalisation of Algebra in Type Theory.* PhD thesis, University of Manchester, 1998. Forthcoming.

[3] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, 1992.

[4] R. Burstall and J. McKinna. Deliverables: a categorical approach to program development in type theory. LFCS report ECS-LFCS-92-242, Dept of Computer Science, University of Edinburgh, 1992.

[5] P.C. Callaghan. Plastic: an implementation of typed LF with coercions. Talk given in the Annual Conf of TYPES'99, June 1999.

[6] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3), 1988.

[7] H. Goguen. *A Typed Operational Semantics for Type Theory.* PhD thesis, University of Edinburgh, 1994.

[8] H. Goguen. The metatheory of UTT. *Types for proofs and programs, LNCS 996*, 1995.

[9] M. Hofmann. *Extensional concepts in intensional type theory.* PhD thesis, Department of Computer Science, University of Edinburgh, 1995.

[10] R. Kiessling, Z. Luo, and J. M. McKinna. Some issues in functional programmiung with dependent types. Talk in the Annual Conference of TYPES'99, June 1999.

[11] Th. Kleymann. *Hoare Logic and VDM: Machine-checked soundness and completeness proofs.* PhD thesis, University of Ediburgh, 1998.

[12] Z. Luo. *An Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, 1990.

[13] Z. Luo. A higher-order calculus and theory abstraction. *Information and Computation*, 90(1), 1991.

[14] Z. Luo. Program specification and data refinement in type theory. *Mathematical Structures in Computer Science*, 3(3), 1993.

[15] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[16] Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.

[17] Z. Luo and P. Callaghan. Coercive subtyping and lexical semantics (extended abstract). *LACL'98*, 1998.

[18] Z. Luo and P. Callaghan. Mathematical vernacular and conceptual well-formedness in mathematical language. *Proceedings of the 2nd Inter. Conf. on Logical Aspects of Computational Linguistics, LNCS/LNAI 1582*, 1998.

[19] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, 1992.

[20] Z. Luo and S. Soloviev. Dependent coercions. *The 8th Inter. Conf. on Category Theory and Computer Science*, 1999.

[21] C. McBride. PhD thesis, University of Edinburgh, 1999. Forthcoming.

[22] J. H. McKinna and R. Pollack. Some lambda calculus and type theory formalized. *Journal of Automated Reasoning*, 1999. Forthcoming.

[23] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[24] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions.* PhD thesis, Edinburgh University, 1994.

[25] R. Pollack. A verified proof checker. *Proceedings of the 2nd Inter. Conf. on Typed Lambda Calculi and Applications, Edinburgh*, April 1995.

[26] B. Reus. *Program Verification in Synthetic Domain Theory.* PhD thesis, LMU Munich, 1995.

[27] A. Saibi. Typing algorithm in type theory with inheritance. *Proc of POPL'97*, 1997.

[28] Thomas Schreiber. Auxiliary variables and recursive procedures. *TAPSOFT'97: Theory and Practice of Software Development, LNCS 1214*, 1997.

[29] S. Soloviev and Z. Luo. Coercion completion and conservertivity in coercive subtyping. Talk given at TYPES'98, 1998.

[30] S. Yu. *Verification of Concurrent Programs Based on Type Theory.* PhD thesis, University of Durham, 1999. Forthcoming.

[31] S. Yu and Z. Luo. Implementing a model checker for Lego. *Proc. of the 4th Inter Symp. of Formal Methods Europe, FME'97: Industrial Applications and Strengthened Foundations of Formal Methods, Graz, Austria. LNCS 1313*, 1997.