A metalanguage for animating inductive definitions

M. R. Lakin

University of Cambridge
Computer Laboratory

20 February 2008

Talk outline

1. Motivation—schematic rule-based definitions
2. Brief introduction to MLSOS
3. Translating inductive definitions into MLSOS
4. Conclusions
5. Related & Future work

Rule-based definitions

- A relation is just a set of mathematical objects.
- We usually define infinite relations using inference rules, as the least set closed under the rules.
- This involve *schematic patterns* which we *instantiate* somehow to produce the underlying mathematical objects. For example:

$$\frac{}{\text{even } 0} \qquad\qquad \frac{\text{even } n}{\text{even } n+2}$$

- As we can see, this is straightforward in the first-order case.
- As usual, things get more complicated when we introduce binders.

Definitions involving binders

- Notions of instantiation are no longer straightforward.
- Suppose we have a schematic term

$$\lambda x.\, \lambda y.\, Var\ x$$

  where x and y are schematic pattern variables.
- Given *concrete atoms* a and b, which of the following are valid instantiations of that pattern?
  1. $\lambda a.\, \lambda b.\, Var\ a$
  2. $\lambda a.\, \lambda a.\, Var\ a$

Animating rule-based definitions

- We are not concerned with proof, but with animating rule-based inductive definitions.
- This means you get an executable prototype (almost) for free when you define your inductive rules.
- The prototype does proof-search over the rules, in order to model your programming language.
- I will present the metalanguage for defining these prototypes later...
- ..first I will present a formal model of inductive rule-based definitions, which hopefully models informal practice reasonably well.

A language of schematic rules

- We define a language of schematic patterns, p:

$$p ::= x \mid () \mid (p_1, \ldots, p_n) \mid \mathsf{K}\ p \mid \langle\langle x \rangle\rangle p$$

- ...which are used to build up formulae, φ:

$$\varphi ::= R\ p \mid x\ =/=\ x' \mid \varphi_1 \wedge \ldots \wedge \varphi_n \mid \mathsf{true}.$$

- A nominal inductive definition, $\mathcal{N}$, is a (finite, well-formed) set of schematic rules, $\mathcal{R}$, of the form:

$$(\mathcal{R})\ \frac{\varphi}{R\ p}$$

Term model semantics

- We give a semantics to $\mathcal{N}$ in terms of ground instantiations, γ, of variables in patterns to produce α-equivalence classes $[g]_\alpha$ of ground nominal terms:

$$\frac{}{\gamma \odot x = \gamma(x)} \qquad \frac{\gamma(x) = \{a\} \qquad \gamma \odot p = [g]_\alpha}{\gamma \odot (\langle\!\langle x \rangle\!\rangle p) = [\langle\!\langle a \rangle\!\rangle g]_\alpha} \qquad \frac{}{\gamma \odot () = \{()\}}$$

$$\frac{\gamma \odot p = [g]_\alpha}{\gamma \odot (\mathsf{K}\ p) = [\mathsf{K}\ g]_\alpha} \qquad \frac{\forall i \in \{1, \ldots, n\}.\ \gamma \odot p_i = [g_i]_\alpha}{\gamma \odot (p_1,\ \ldots,\ p_n) = [(g_1,\ \ldots,\ g_n)]_\alpha}$$

- *Note that, at this point, distinct variables may be instantiated with the same atom—even if in abstraction position.*

Term model semantics

- For a definition which defines relations $R_1, \ldots, R_n$, we say that a ground term model, $\mathcal{H}$, is an n-tuple $(\mathcal{H}_1, \ldots, \mathcal{H}_n)$ of models—one per relation symbol.
- We define a satisfaction relation $\mathcal{H} \models_\gamma \varphi$ as follows:

$$\frac{}{\mathcal{H} \models_\gamma \mathsf{true}} \qquad \frac{\forall i \in \{1, \ldots, n\}.\ \mathcal{H} \models_\gamma \varphi_i}{\mathcal{H} \models_\gamma (\varphi_1 \wedge \ldots \wedge \varphi_n)}$$

$$\frac{\gamma \odot p = [g]_\alpha \qquad [g]_\alpha \in \mathcal{H}_i}{\mathcal{H} \models_\gamma (R_i\ p)}$$

$$\frac{\gamma(x) = \{a\} \qquad \gamma(x') = \{a'\} \qquad a \neq a'}{\mathcal{H} \models_\gamma (x\ =/=\ x')}$$

Closure under rules

- To get the set of $\mathcal{H}$ which satisfy a definition $\mathcal{N}$, we close under the schematic rules, as follows.

$$\frac{(\mathcal{H} \models_\gamma \varphi) \Rightarrow (\mathcal{H} \models_\gamma R\ p)}{\mathcal{H} \models_\gamma (\varphi \Rightarrow R\ p)}$$

$$\frac{\forall \mathcal{R} \in \mathcal{N}.\ \forall \gamma.\ P(\mathcal{R}, \gamma) \Rightarrow \mathcal{H} \models_\gamma \mathcal{R}}{\mathcal{H} \models \mathcal{N}}$$

- The predicate $P(\mathcal{R}, \gamma)$ restricts the instantiations that can be required to a particular set of schematic rules.

Restrictions on instantiation

- There are various choices for predicate P, e.g.
  1. $P(\mathcal{R}, \gamma) \triangleq \mathsf{true}$
     (any instantiation at all is permitted)
  2. $P(\mathcal{R}, \gamma) \triangleq \forall x, y \in \mathsf{av}(\mathcal{R}).\ x \neq y \Rightarrow \gamma(x) \neq \gamma(y)$
     (γ must be injective on names in abstraction position)
  3. $P(\mathcal{R}, \gamma) \triangleq \forall x, y \in \mathsf{vars}(\mathcal{R}).$
     $\mathit{sort}(x) = \mathit{sort}(y) = \alpha \wedge x \neq y \Rightarrow \gamma(x) \neq \gamma(y)$
     (γ must be injective on *all* names of atom sort)
- The choice here is largely personal.
- However, if (1) were chosen, then proof-search using nominal matching would probably not be complete (cf $\lambda a.\, \lambda b.\, a$ vs $\lambda a.\, \lambda a.\, a$).

An example rule

- An example: the β-rule (using syntactic sugar for substitution).

$$(\beta)\ \frac{t_1'[t_2'/x] \equiv t_3}{\mathsf{beta}\ ((\mathsf{App}\ (\mathsf{Lam}\ \langle\!\langle x \rangle\!\rangle t_1), t_2), t_3)}$$

- Note that the names and λ-terms are both drawn represented using the same syntactic class of schematic variables.
- We shall see later how they are actually implemented in the metalanguage.

Talk outline

1. Motivation—schematic rule-based definitions
2. Brief introduction to MLSOS
3. Translating inductive definitions into MLSOS
4. Conclusions
5. Related & Future work

What is MLSOS?

- A minimal calculus for animating rule-based inductive definitions involving binders.
- A little functional/logic programming language which extends the functionality of FreshML.
- MLSOS offers operations useful for proof-search computation over inductive definitions, e.g.:
  1. support for binders using nominal techniques,
  2. pattern-matching using nominal unification
  3. (with a few extra rules for name inequality),
  4. generation of fresh atoms and metavariables, and
  5. branching constructs for proof-search.

MLSOS grammar

$$
\begin{array}{rcll}
\text{Nominal Arities, } \sigma & ::= & \alpha & \text{atom sort,} \\
 & | & \delta & \text{data sort,} \\
 & | & 1 & \text{unit type,} \\
 & | & \sigma_1 * \cdots * \sigma_n & n\text{-tuple,} \\
 & | & \langle\langle\alpha\rangle\rangle\sigma & \text{abstraction type.} \\
\text{Types, } \tau & ::= & \sigma & \text{nominal arity,} \\
 & | & \mathsf{ans} & \text{answer type,} \\
 & | & \tau \to \tau' & \text{function type.}
\end{array}
$$

MLSOS grammar

Constraints, c	$::=$	$v \;=:=\; v'$	equality constraint,
	$\mid$	$a \mathbin{\#} v$	freshness constraint,
	$\mid$	$v \;=/=\; v'$	*name* inequality constraint.
Values, v	$::=$	x	value identifier,
	$\mid$	$\pi\, X$	suspension,
	$\mid$	$()$	unit,
	$\mid$	$(v_1,\ \ldots,\ v_n)$	n-tuple,
	$\mid$	$\text{fun}\, f(x : \tau) : \tau' = e$	recursive function,
	$\mid$	yes	success,
	$\mid$	$\mathrm{K}\ v$	data construction,
	$\mid$	a	atom,
	$\mid$	$\langle\!\langle a \rangle\!\rangle v$	atom abstraction.

MLSOS grammar

$$
\begin{array}{rcll}
\text{Expressions, } e & ::= & v & \text{value,} \\
& \mid & \mathsf{let}\ x = e\ \mathsf{in}\ e' & \text{let-binding,} \\
& \mid & v\ v' & \text{function application,} \\
& \mid & \mathsf{fresh}\ a : \alpha\ \mathsf{in}\ e & \text{fresh atom,} \\
& \mid & \mathsf{some}\ x : \sigma\ \mathsf{in}\ e & \text{new unification variable,} \\
& \mid & c & \text{constraint,} \\
& \mid & e_1\ \mathsf{or}\ \cdots\ \mathsf{or}\ e_n & n\text{-ary branch.} \\
\text{Frame Stacks, } S & ::= & Id & \text{empty frame stack,} \\
& \mid & S \circ (x.\, e) & \text{non-empty frame stack.}
\end{array}
$$

- **NB: branches introduce non-determinism.**

Operational semantics

- MLSOS evaluation contexts are of the form

$$\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(e)).$$

- We define a binary transition relation $\longrightarrow_M$ between configurations.
- As we will see, this relation is non-deterministic...
- This is necessary to do proof-search.

Operational semantics

A few selected operational rules:

- $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(c)) \longrightarrow_M \text{И}\overline{a}\ \exists\overline{X}\ ((\overline{c} \cup \{c\}); S(\mathsf{yes}))$
  if $\models \overline{c} \cup \{c\}$
- $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(\mathsf{fresh}\ a : \alpha\ \mathsf{in}\ e)) \longrightarrow_M \text{И}\overline{a}, a : \alpha\ \exists\overline{X}\ (\overline{c}'; S(e))$
  if $a \notin dom(\overline{a})$ and $\overline{c}' \triangleq \{a \# X \mid X \in dom(\overline{X})\} \cup \overline{c}$
- $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(\mathsf{some}\ x : \sigma\ \mathsf{in}\ e)) \longrightarrow_M \text{И}\overline{a}\ \exists\overline{X}, X : \sigma\ (\overline{c}; S(e[\iota\, X/x]))$
  if $X \notin dom(\overline{X})$
- $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(e_1\ \mathsf{or}\ \cdots\ \mathsf{or}\ e_n)) \longrightarrow_M \text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(e_i))$
  where $i \in \{1, \ldots, n\}$

Operational semantics

- We define two notions of observation on configurations:

  1. $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(e)) \downarrow$
     if some branch of execution leads to a terminal configuration (i.e. $\text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; Id(v))$, where $\overline{c}'$ is a satisfiable set of constraints).

  2. $\text{И}\overline{a}\ \exists\overline{X}\ (\overline{c}; S(e))$ *fails*
     if all branches of execution leads to a stuck configuration (i.e. $\text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(c'))$, where $\overline{c}' \cup \{c'\}$ is an unsatisfiable set of constraints).

- These mirror the $\longrightarrow_M$ rules from earlier.

Operational equivalence

- We write (closed) operational equivalence as $\cong$.
- Two closed MLSOS expressions e and e' are operationally equivalent if their termination and failure behaviour is the same in any context $\text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(-))$, i.e.

$$\begin{array}{rcl} \text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(e)) \downarrow & \iff & \text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(e')) \downarrow \\ \text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(e))\ \mathit{fails} & \iff & \text{И}\overline{a}'\ \exists\overline{X}'\ (\overline{c}'; S'(e'))\ \mathit{fails} \end{array}$$

  both hold.
- We extend this to a relation $\cong^{\circ}$ on open expressions by closing, ground substitutions for free value identifiers.

CIU and data correctness results

- CIU theorem: $\cong^{\circ}$ has various nice properties, including being an equivalence relation and a congruence.
- All fairly straightforward except for compatibility, which means that operational equivalence respects the term-formers of the language.
- Data correctness: α-equivalent ground terms cannot be distinguished operationally. The main theorem:

$$\overline{\mathsf{a}} \vdash g \cong g' : \sigma \iff \overline{\mathsf{a}} \vdash g \approx_{\alpha} g' : \sigma.$$

- The proofs both involve drawn-out operational reasoning.
- These are the proofs I am trying to automate using nominal Isabelle.

Talk outline

Modelling nominal inductive definitions

- We have now established some fundamental correctness results about the metalanguage.
- Next job: say something about the expressiveness of the metalanguage.
- Strategy: define a formal translation of *nominal inductive definitions* into MLSOS, and prove and prove that the implementations are:
  - Adequate: if MLSOS reports that a given term is in the relation then it actually is in the relation, and
  - Complete: the system will find all members of the relation (probably doesn't hold in general—see later).

Translating rules into MLSOS

- Consider the following β-rule:

$$(\beta)\ \frac{t_1'[t_2'/x] \equiv t_3}{\text{beta } ((\text{App } (\text{Lam } \langle\!\langle x \rangle\!\rangle t_1), t_2), t_3)}$$

- Things to be done to translate this rule into MLSOS:

  1. Generate fresh atoms and unification variables to stand for the pattern variables in the rule.
  2. Decide which atoms need to be fresh for which unification variables.
  3. Match against the pattern from the conclusion.
  4. If successful, recursively process the formulae from the premise.

- In more detail...

Translating rules into MLSOS

$$(\beta)\ \frac{t_1'[t_2'/x] \equiv t_3}{\mathsf{beta}\ ((\mathsf{App}\ (\mathsf{Lam}\ \langle\langle x \rangle\rangle t_1), t_2), t_3)}$$

1. Generate fresh atoms and unification variables to stand for the pattern variables in the rule.
    - Any variable that appears in abstraction position (coloured red above) is implemented using a fresh atom.
    - Any other variable is implemented using a unification variable (*including those of atom sort*).

Translating rules into MLSOS

$$(\beta)\ \frac{t_1'[t_2'/x] \equiv t_3}{\mathsf{beta}\ ((\mathsf{App}\ (\mathsf{Lam}\ \langle\!\langle x \rangle\!\rangle t_1), t_2), t_3)}$$

2. Decide which atoms need to be fresh for which unification variables.
   - *In this case*, the name x must be constrained to be fresh for the variables coloured in red.
   - Names bound in the conclusion should be fresh for the conclusion.
   - Names *only* bound in the *premises* should be fresh for both the premises *and* the conclusion.
   - *These freshness constraints should not effect adequacy, but will affect completeness. Not clear yet whether these are strong enough / too strong...*

Translating rules into MLSOS

$$(\beta)\ \frac{t_1'[t_2'/x] \equiv t_3}{\mathsf{beta}\ ((\mathsf{App}\ (\mathsf{Lam}\ \langle\!\langle x \rangle\!\rangle t_1), t_2), t_3)}$$

3. Match against the pattern from the conclusion.
   - We create a nominal pattern consisting of fresh atoms and metavariables, and use an equality constraint to match against it.
4. If successful, recursively process the formulae from the premise.
   - This is just a recursive call to the function $f_{\mathcal{N}}$ which implements the inductive definition $\mathcal{N}$.

Adequacy and completeness

- An adequacy proof is mostly complete...
- However, completeness is more tricky.
- One can think of "bad" rules for which proof search using nominal unification would fail, e.g.:

$$\frac{}{R\ (x, t, \langle\!\langle x \rangle\!\rangle t)}$$

  which produces the graph of λ-abstraction.
- When encoded into MLSOS, this would fail because of the freshness constraints in the conclusion (which would seem reasonable...)
- A syntactic criterion on schematic rules is needed to rule out such definitions.

Talk outline

1. Motivation—schematic rule-based definitions
2. Brief introduction to MLSOS
3. Translating inductive definitions into MLSOS
4. Conclusions
5. Related & Future work

Conclusions

- Proof-search over inductive definitions with binders has raised issues similar to those in theorem-proving, e.g. "VC-compatibility"
- The status of these rules, which are equivariant yet still somehow "bad", needs further investigation
- However, we hope that our system will permit a reasonable number of interesting programs to be written!

Talk outline

1. Motivation—schematic rule-based definitions
2. Brief introduction to MLSOS
3. Translating inductive definitions into MLSOS
4. Conclusions
5. Related & Future work

Related work

- FreshML: Shinwell, Pitts, Gabbay
- αProlog: Cheney, Urban
- Abella, λProlog, Bedwyr etc: Miller, Baelde et al
- Nominal Isabelle: everyone here!
- PLTRedex: Findler et al
- Curry: Hanus et al
- Twelf etc: Pfenning et al

Future work

- Finish soundness and completeness proofs
- Implement the system and program some real examples
- Investigate optimisations in the compilation of rules into MLSOS code
- Compare our system with others...
- ... und endlich muß ich eine Dissertation schreiben!