Combining generic judgments with recursive definitions

Andrew Gacek[1] Dale Miller[2] Gopalan Nadathur[1]

[1]Department of Computer Science and Engineering
University of Minnesota

[2]INRIA Saclay - Île-de-France
& LIX/École polytechnique

LICS'08
June 24, 2008

Preview

Context

We want to specify and reason over syntactic objects with binding

Useful logical features in this context

- higher-order abstract syntax representation of objects
- ∇-quantifier for *generic judgments*
- recursive definitions for inductive specifications
- natural number induction

Contribution

We combine generic judgments with recursive definitions to obtain a mechanism for internalizing properties related to the treatment of binding

Running example: type assignment

$$\frac{x : a \in \Gamma}{\Gamma \vdash x : a}$$

$$\frac{\Gamma \vdash t_1 : a \to b \quad \Gamma \vdash t_2 : a}{\Gamma \vdash (t_1\ t_2) : b}$$

$$\frac{\Gamma, x : a \vdash t : b}{\Gamma \vdash (\lambda x{:}a.\ t) : a \to b}\ x \notin dom(\Gamma)$$

∇ quantifier: generic judgments

Miller & Tiu "Generic Judgments" [LICS03, ToCL05]
Tiu "LG^ω" [LFMTP06]

$\nabla x.F$ means F has a generic proof—one which depends on the freshness, but not the form of x

$$\forall x.F \supset \nabla x.F \qquad \nabla x.F \not\supset \forall x.F$$

$$\nabla x.\nabla y.F \equiv \nabla y.\nabla x.F$$

$$\nabla x.F \equiv F \qquad \text{if } x \text{ does not appear in } F$$

These structural rules allow a treatment of ∇ based on *nominal constants* which make quantification implicit

Logical rules for ∇

$$\frac{\Gamma, B[a/x] \vdash C}{\Gamma, \nabla x.B \vdash C}\ \nabla\mathcal{L} \qquad\qquad \frac{\Gamma \vdash B[a/x]}{\Gamma \vdash \nabla x.B}\ \nabla\mathcal{R}$$

a is a nominal constant not appearing in B

Nominal constants have (implicit) formula level binding

Nominal constants are equivariant (always permutable)

$$\frac{\pi.B = \pi'.B'}{\Gamma, B \vdash B'}\ id_\pi$$

Role of definitions

Definitions for atomic judgments encode specifications

$$member\ A\ (A :: L) \triangleq \top$$

$$member\ A\ (B :: L) \triangleq member\ A\ L$$

For an atomic judgment,

- right introduction corresponds to backchaining on the predicate definition
- left introduction corresponds to case-analysis over the predicate definition

Typing example with ∇

$$of\ \Gamma\ X\ A \triangleq member\ (X : A)\ \Gamma$$

$$of\ \Gamma\ (app\ T_1\ T_2)\ B \triangleq \exists A.\ of\ \Gamma\ T_1\ (arr\ A\ B) \wedge of\ \Gamma\ T_2\ A$$

$$of\ \Gamma\ (abs\ A\ T)\ (arr\ A\ B) \triangleq \nabla x.\ of\ ((x : A) :: \Gamma)\ (T\ x)\ B$$

Example property:

$$\forall L, a, b, t_1, t_2. \nabla x.$$
$$of\ ((x : a) :: L)\ (t_1\ x)\ b \wedge of\ L\ t_2\ a \supset of\ L\ (t_1\ t_2)\ b$$

Reasoning about type uniqueness

$$\forall t, a_1, a_2.(of\ nil\ t\ a_1 \wedge of\ nil\ t\ a_2) \supset a_1 = a_2$$

$$\forall \Gamma, t, a_1, a_2.(of\ \Gamma\ t\ a_1 \wedge of\ \Gamma\ t\ a_2) \supset a_1 = a_2$$

$$\forall \Gamma, t, a_1, a_2.(cntx\ \Gamma \wedge of\ \Gamma\ t\ a_1 \wedge of\ \Gamma\ t\ a_2) \supset a_1 = a_2$$

$cntx\ \Gamma$ should enforce

- $\Gamma = (x_1 : a_1) :: (x_2 : a_2) :: \ldots :: (x_n : a_n) :: nil$
- Each x_i is atomic
- Each x_i is unique

We want a mechanism for defining well-formed contexts so that these kinds of (generic) properties are satisfied

Extended form of definitions

Definitional clauses take the form

$$\forall \vec{x}.(\nabla \vec{z}.H) \triangleq B$$

where

- no nominal constants appear in H or B (equivariance)
- clauses are stratified (consistency)

Meaning of such a clause

An instance of H is true if the corresponding instance of B is true, provided

- $\vec{z}$ is instantiated with unique nominal constants $\vec{a}$
- $\vec{x}$ is instantiated with terms not containing $\vec{a}$

Definition examples

$$(\nabla x.\mathit{name}\; x) \triangleq \top$$

$$\forall E.\; (\nabla x.\mathit{fresh}\; x\; E) \triangleq \top$$

$$\mathit{cntx}\; \mathit{nil} \triangleq \top$$

$$\forall L, A.\; (\nabla x.\mathit{cntx}\; ((x : A) :: L)) \triangleq \mathit{cntx}\; L$$

Raising and the encoding of dependencies

Many proof rules require ∇-bound variables to have minimal scope

In the general sequent $\Sigma : \Gamma \vdash C$

- eigenvariables in Σ have sequent-level scope
- nominal constants have formula-level scope

Principles which allow ∇ to be moved inwards over quantifiers

$$\nabla x.\forall y.F\ x\ y \equiv \forall y'.\nabla x.F\ x\ (y'\ x)$$
$$\nabla x.\exists y.F\ x\ y \equiv \exists y'.\nabla x.F\ x\ (y'\ x)$$

This device is called *raising*

Right rule for definitions

The right introduction rule for atomic judgments corresponds to backchaining on a definitional clause

Clause: $\forall \vec{x}.(\nabla \vec{z}.H) \triangleq B$
Sequent: $\Sigma : \Gamma \vdash A$

1. Raise $\vec{x}$ over the nominal constants in A, and instantiate $\vec{z}$ with unique nominal constants: $\forall \vec{x}'.H' \triangleq B'$
2. Raise Σ over the nominal constants instantiating $\vec{z}$: $\Sigma' : \Gamma', A' \vdash C'$
3. Match $A' = (\pi.H')\theta$ where π is a permutation of the nominal constants in H'

$$\frac{\Sigma' : \Gamma' \vdash (\pi.B')\theta}{\Sigma : \Gamma \vdash A}\ \mathit{def}\mathcal{R}$$

Left rule for definitions

The left introduction rule for atomic judgments is the natural counterpart to *def*$\mathcal{R}$: it considers all possible ways an atomic judgment may have been derived

Clause: $\forall \vec{x}.(\nabla \vec{z}.H) \triangleq B$
Sequent: $\Sigma : \Gamma, A \vdash C$

1. Raise $\vec{x}$ over the nominal constants in A, and instantiate $\vec{z}$ with unique nominal constants: $\forall \vec{x}'.H' \triangleq B'$
2. Raise Σ over the nominal constants instantiating $\vec{z}$: $\Sigma' : \Gamma', A' \vdash C'$
3. Unify $A'\theta = (\pi.H')\theta$ where π is a permutation of the nominal constants in H'

$$\frac{\{\Sigma'\theta : \Gamma'\theta, (\pi.B')\theta \vdash C'\theta\}}{\Sigma : \Gamma, A \vdash C}\ \mathit{def}\mathcal{L}$$

Consistency of $\mathcal{G}$

Consistency is shown by establishing the eliminability of cut

$$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \Sigma' : \Gamma' \vdash (\pi.B')\theta\end{matrix}}{\Sigma : \Gamma \vdash A}\,\mathit{def}\mathcal{R} \qquad \dfrac{\left\{\begin{matrix}\Pi_2^{\rho,\pi',B''}\\ \Sigma''\rho : (\pi'.B'')\rho, \Delta''\rho \vdash C''\rho\end{matrix}\right\}}{\Sigma : A, \Delta \vdash C}\,\mathit{def}\mathcal{L}}{\Sigma : \Gamma, \Delta \vdash C}\; \mathit{cut}$$

$$\dfrac{\begin{matrix}\Pi_1\\ \Sigma' : \Gamma' \vdash (\pi.B')\theta'\end{matrix} \qquad \begin{matrix}\Pi_2^{\theta',\pi,B'}\\ \Sigma' : (\pi.B')\theta', \Delta' \vdash C'\end{matrix}}{\Sigma' : \Gamma', \Delta' \vdash C'}\; \mathit{cut}$$

Raising (in $\mathit{def}\mathcal{L}$ and $\mathit{def}\mathcal{R}$) preserves provability and proof height

Application: $\mathcal{G}$ as meta-logic

Goal: specify and reason over syntactic objects with binding

- Decide on a suitable specification logic
- Use the specification logic to encode an object language
- Encode that specification logic in $\mathcal{G}$
- Reason in $\mathcal{G}$ via this specification about the object language

This is approach is implemented in Abella (Gacek 2008) and has been used to give proofs of

- determinacy and type preservation of various evaluation strategies
- cut admissibility for a sequent calculus
- Church-Rosser property for λ-calculus
- Tait-style weak normalizability proof

Related Work

Locally nameless representation

A first-order representation with de Bruijn indices for bound variables and names for free variables [Aydemir *et. al.* PoPL08]

Nominal logic approach

A formalization of bound and free variable names in an existing theorem prover (Isabelle/HOL) [Urban and Tasson CADE04]

Twelf

An expressive specification logic (LF) with a relatively weak meta-logic ($\mathcal{M}_2^+$) [Schürmann and Pfenning CADE98]

Conclusions

Focus has been on a particular approach to specifying and reasoning over syntactic objects with binding

- λ-terms and generic judgments for encoding binding
- recursive definitions for encoding specifications
- support for inductive arguments

Contribution

Combining definitions with generic judgments enables expressive and declarative reasoning over implicit properties of those specifications, such as the structure of contexts

Future work

- induction and coinduction on definitions
- continued work Abella and its applications
  - experimenting with different specification logics
  - automating proof search
  - applications to practical software systems