Proofs in Structured Specifications

Don Sannella

LFCS, School of Informatics, University of Edinburgh

MLPA, 15 July 2010

Algebraic specification

Starting point:

- take many-sorted algebras, or similar, as models of programs
- use axioms in a logical system involving equality for describing required properties

About:

- specifying programs
- proving correctness of programs with respect to specifications
- developing correct programs from specifications

Proof is obviously central, but the theory is primarily model-oriented

Algebraic specification

Modular structure is used to tame large programs.

- My favourite: ML module system (signatures, structures, functors)

Modular structure is needed to tame large specifications

- Build large structured specifications by combining smaller specifications

R. Burstall and J. Goguen. Putting theories together to make specifications. Proc. IJCAI 1977

Algebraic specification

Need to:

- prove correctness of modular programs with respect to structured specifications
- develop modular programs from structured specifications

(Problem: usually the structure of the specification will not match the structure of the program)

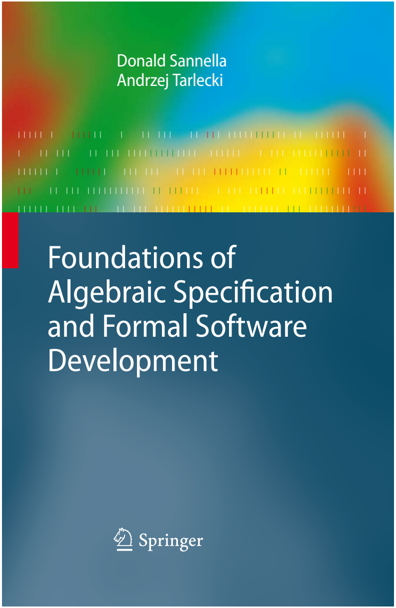
Based on: proof in structured specifications

- Obvious relation to the use of a proof assistant in the context of a library

My aim in this talk:

- breezy introduction to one approach to algebraic specification
- concentrating on proof

Advertisement



Early algebraic specification (1970s)

Specification $\langle \Sigma, E \rangle$

- Σ is a signature (set of sorts, set of operations $f : s_1 \times \cdots \times s_n \to s$)
- E is a set of equations
- Semantics: $\mathbf{Mod}(\langle \Sigma, E \rangle) = \{M \in \mathbf{Alg}(\Sigma) \mid M \models E\}$

$E \models e$ iff $\mathbf{Mod}(\langle \Sigma, E \rangle) \models e$

$E \vdash e$: equational calculus (reflexivity, transitivity, etc.)
Theorem: $E \vdash e$ is sound and complete w.r.t. $E \models e$

Rewriting techniques can be used to give a decision procedure for some specifications (Knuth-Bendix completion algorithm)

Early algebraic specification, continued

But there are too many models:

- including trivial ones
- including ones containing unreachable elements, so induction is not valid

Solution: take the initial model $T_\Sigma/{\equiv_E}$

- Proofs use equational logic and induction
- Completeness is lost: there is no complete proof system for initial models of equational specifications
- Term rewriting using inductionless induction (a.k.a. proof by consistency)

J. Goguen, J. Thatcher and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. IBM Technical Report 1976.

Early algebraic specifications: inadequacies (1)

Problems with large specifications:

- Management of name space, e.g. auxiliary operations
- It is difficult to deal with very long lists of axioms
- Simple unions of such sets can give surprising results, especially in combination with initiality
- ... unless strong restrictions are imposed ("sufficient completeness", "hierarchy consistency")
- ... which are problematic in connection with "loose" specifications

Solution: language of structured specifications

Early algebraic specifications: inadequacies (2)

Equations are inadequate:

- Need more powerful logics for convenient expressibility
- . . . but which one? There are many candidates
- On the other hand, we need (conditional) equations to guarantee existence of initial models
- But in structured specifications, initial models (i.e. free extensions) aren't guaranteed anyway
- What you really need is reachability (a.k.a. finite generation) which gives induction

Solution: specifications in an arbitrary logical system (a.k.a. *institution*), abandon restriction to initial models

Institutions

An institution consists of:

- a set of signatures, with signature morphisms $\sigma : \Sigma \to \Sigma'$
- for each Σ, a set $\mathbf{Sen}(\Sigma)$ of Σ-sentences
- and for each $\sigma : \Sigma \to \Sigma'$, a translation $\sigma : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$
- for each Σ, a set $\mathbf{Mod}(\Sigma)$ of Σ-models
- and for each $\sigma : \Sigma \to \Sigma'$, a translation $\cdot|_\sigma : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$
- for each Σ, a satisfaction relation $\models_\Sigma \subseteq \mathbf{Mod}(\Sigma) \times \mathbf{Sen}(\Sigma)$
- such that for any $\sigma : \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$, $M' \in \mathbf{Mod}(\Sigma')$

$$M'|_\sigma \models_\Sigma \varphi \qquad \text{iff} \qquad M' \models_{\Sigma'} \sigma(\varphi)$$

Institutions: equational logic

- Signatures: the usual ones, with the usual signature morphisms (renamings)
- Sentences over Σ: equations with sorts/operations from Σ
- Signature morphisms rename the sorts/operations
- Models over Σ: Σ-algebras $\mathbf{Alg}(\Sigma)$
- A signature morphism $\sigma : \Sigma \to \Sigma'$ induces reduct, $\cdot|_\sigma : \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$
- Satisfaction relation $\models_\Sigma$: satisfaction of Σ-equations by Σ-algebras

Institutions: first-order equational logic

- Signatures: the usual ones, with predicates, with the usual signature morphisms (renamings)
- Sentences over Σ: closed first-order formulae with equations and predicate applications as atomic formulae
- Signature morphisms rename the sorts/operations/predicates
- Models over Σ: Σ-structures
- A signature morphism $\sigma : \Sigma \to \Sigma'$ induces reduct, $\cdot|_\sigma : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$
- Satisfaction relation $\models_\Sigma$: satisfaction of Σ-formulae by Σ-structures

Institutions: others

Dozens of other examples:

- order-sorted logic
- higher-order logic
- logics of partial functions
- multiple-valued logics
- temporal logics
- logic for hybrid systems
- observational logic
- process description logics
- etc. etc. etc.

Usually equations are (one kind of) atomic formula

Assuming for this talk

- induction provided as a special kind of sentence
- ... or as part of the signature (designating constructors).

Structured specifications

Using a simple set of specification-building operations

Semantics:

- $\mathbf{Sig}(SP)$ is a signature
- $\mathbf{Mod}(SP)$ is a class of $\mathbf{Sig}(SP)$-models

Basic specifications $\langle \Sigma, \Phi \rangle$

- $\mathbf{Sig}(\langle \Sigma, \Phi \rangle) = \Sigma$
- $\mathbf{Mod}(\langle \Sigma, \Phi \rangle) = \{M \in \mathbf{Mod}(\Sigma) \mid M \models_\Sigma \Phi\}$

Structured specifications

Union $SP \cup SP'$, requiring $\mathbf{Sig}(SP) = \mathbf{Sig}(SP')$

- $\mathbf{Sig}(SP \cup SP') = \mathbf{Sig}(SP)$
- $\mathbf{Mod}(SP \cup SP') = \mathbf{Mod}(SP) \cap \mathbf{Mod}(SP')$

Translation $\sigma(SP)$, requiring $\sigma : \mathbf{Sig}(SP) \to \Sigma'$

- $\mathbf{Sig}(\sigma(SP)) = \Sigma'$
- $\mathbf{Mod}(\sigma(SP)) = \{M \in \mathbf{Mod}(\Sigma') \mid M|_\sigma \in \mathbf{Mod}(SP)\}$

Derive $SP|_\sigma$, requiring $\sigma : \Sigma \to \mathbf{Sig}(SP)$

- $\mathbf{Sig}(SP|_\sigma) = \Sigma$
- $\mathbf{Mod}(SP|_\sigma) = \{M|_\sigma \mid M \in \mathbf{Mod}(SP)\}$

Structured specifications

Other operations can be expressed in these terms:

$\textbf{enrich } SP \textbf{ by sorts } S \textbf{ opns } \Omega \textbf{ axioms } \Phi = \sigma(SP) \cup \langle \Sigma, \Phi \rangle$
where $\Sigma = \mathbf{Sig}(SP) \cup \langle S, \Omega \rangle$ and $\sigma : \mathbf{Sig}(SP) \hookrightarrow \Sigma$

$SP + SP' = \sigma(SP) \cup \sigma'(SP')$
where $\Sigma = \mathbf{Sig}(SP) \cup \mathbf{Sig}(SP')$, $\sigma : \mathbf{Sig}(SP) \hookrightarrow \Sigma$ and
$\sigma' : \mathbf{Sig}(SP') \hookrightarrow \Sigma$

$\textbf{export } \Sigma \textbf{ from } SP = SP|_\sigma$
where $\sigma : \Sigma \hookrightarrow \mathbf{Sig}(SP)$

Stepwise refinement, simple version

$SP \rightsquigarrow SP_1 \rightsquigarrow \cdots \rightsquigarrow SP_n$

$SP \rightsquigarrow SP'$ means

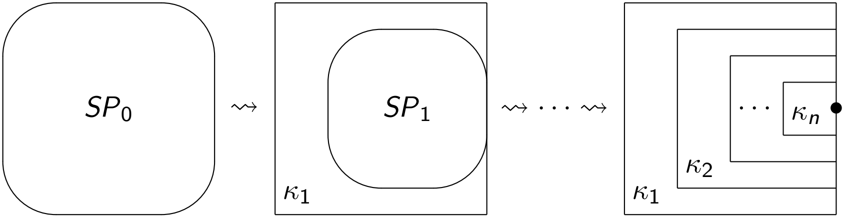
- $\mathbf{Sig}(SP) = \mathbf{Sig}(SP')$
- $\mathbf{Mod}(SP) \supseteq \mathbf{Mod}(SP')$

Stop when we have a model M_n of SP_n

- Then $M_n \in \mathbf{Mod}(SP)$

Stepwise refinement

Problem:



Stepwise refinement, improved

A better way:

$$\boxed{SP_0} \rightsquigarrow_{\kappa_1} \boxed{SP_1} \rightsquigarrow_{\kappa_2} \cdots \rightsquigarrow_{\kappa_n} \bullet \quad SP_n = \textsc{Empty}$$

Stepwise refinement, improved

$$SP \rightsquigarrow_{\kappa_1} SP_1 \rightsquigarrow_{\kappa_2} \cdots \rightsquigarrow_{\kappa_n} SP_n$$

$SP \rightsquigarrow_{\kappa} SP'$ means

- $\kappa : \mathbf{Mod}(SP') \to \mathbf{Mod}(SP)$
- κ is a "constructor" taking SP' models to SP models
- Given a model of what SP' requires, κ builds a model of what SP requires
- Think of κ as a program fragment (better: ML functor)

Stop when we have a model M_n of SP_n

- Then $\kappa_1(\kappa_2(\cdots \kappa_n(M_n) \cdots)) \in \mathbf{Mod}(SP)$

What about proofs?

$\Phi \models_\Sigma \varphi$ means

- $M \models_\Sigma \varphi$ for all $M \in \mathbf{Mod}(\Sigma)$ such that $M \models_\Sigma \Phi$

We need a consequence relation $\Phi \vdash_\Sigma \varphi$ that is sound w.r.t. $\Phi \models_\Sigma \varphi$

- Completeness is nice but hardly ever achievable, despite results to come
- For equational logic, $\vdash$ is given by the equational calculus
- For first-order logic, $\vdash$ is some standard proof system for first-order logic
- Etc.

Assume we have such a $\vdash$ for the institution at hand

Other levels of proof

We need proofs at more levels:

- $SP \vdash \varphi$: φ holds in all models of SP
- $SP \vdash SP'$: correctness of refinement steps $SP' \rightsquigarrow SP$
- $SP \vdash_\kappa SP'$: correctness of refinement steps $SP' \rightsquigarrow_\kappa SP$

What about proving that a program P correctly implements a specification SP?

- This is covered by $\vdash_P$, if we view the program as a constructor $P : \mathbf{Mod}(\textsc{BuiltIns}) \rightarrow \mathbf{Mod}(SP)$
- More generally, if P is an ML functor

$$\mathbf{functor}\ P(X : SP) : SP' = \ldots$$

then we need $SP \vdash_P SP'$

Proof in structured specifications

Most relevant to MLPA

$SP \models \varphi$ means

- $M \models_{\mathbf{Sig}(SP)} \varphi$ for all $M \in \mathbf{Mod}(SP)$

Need $SP \vdash \varphi$ that is sound w.r.t. $SP \models \varphi$

Builds on $\Phi \vdash \varphi$:

$$\frac{SP \vdash \varphi_1 \quad \cdots \quad SP \vdash \varphi_n \qquad \{\varphi_1, \ldots, \varphi_n\} \vdash \varphi}{SP \vdash \varphi}$$

Proof in structured specifications

Rules for the specification-building operations:

$$\frac{}{\langle \Sigma, \Phi \rangle \vdash \varphi} \quad \varphi \in \Phi$$

$$\frac{SP_1 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi} \qquad \frac{SP_2 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi}$$

$$\frac{SP \vdash \varphi}{\sigma(SP) \vdash \sigma(\varphi)} \qquad \frac{SP \vdash \sigma(\varphi)}{SP|_\sigma \vdash \varphi}$$

Then $SP \vdash \varphi$ is sound w.r.t. $SP \models \varphi$

Proof in structured specifications

Theorem: If

1. $\Phi \vdash \varphi$ is complete w.r.t. $\Phi \models \varphi$
2. **INS** is finitely exact (signature category is finitely cocomplete, model functor is finitely continuous)
3. **INS** has interpolation
4. **INS** can express truth, negation and conjunction

then $SP \vdash \varphi$ is complete w.r.t. $SP \models \varphi$

Interpolation:

- Suppose $\varphi_1 \models \varphi_2$
- ... then there is θ using only the common symbols of φ_1, φ_2
- ... such that $\varphi_1 \models \theta$ and $\theta \models \varphi_2$

Exercise: Express interpolation institutionally

If SP doesn't involve $\cdot|_\sigma$, then (2)–(4) aren't required

Proof in structured specifications

Problem: equational logic doesn't have interpolation
And $SP \vdash \varphi$ is not complete for equational logic

Alternative: normalise specifications

Theorem: If **INS** is finitely exact, then for any SP there is an equivalent specification $\mathsf{nf}(SP)$ of the form $\langle \Sigma, \Phi \rangle|_\sigma$

Then use

$$\frac{\Phi \vdash \sigma(\varphi)}{SP \vdash \varphi} \quad \mathsf{nf}(SP) = \langle \Sigma, \Phi \rangle|_\sigma$$

Proof in structured specifications

Theorem: If

1. $\Phi \vdash \varphi$ is complete w.r.t. $\Phi \models \varphi$
2. **INS** is finitely exact

then $SP \vdash \varphi$ via $\mathrm{nf}(SP)$ is complete w.r.t. $SP \models \varphi$

Proof in structured specifications

But we would like to take advantage of the structure of specifications in proof search.

D. Sannella and R. Burstall. Structured theories in LCF. Proc. CAAP 1983
W. Farmer, J. Guttman and F. Thayer. Little theories. Proc. CADE 1992

The earlier rules are compositional

- Proofs follow the structure of the specification
- Lemmas proved in sub-specifications are put together in a bigger specification to obtain the result

The nf rule is non-compositional

Proof in structured specifications

A middle way: transform specifications without completely flattening them:

$$\frac{SP' \vdash \varphi}{SP \vdash \varphi} \quad SP \equiv SP'$$

Or, using $SP \vdash SP'$ below:

$$\frac{SP \vdash SP' \qquad SP' \vdash \varphi}{SP \vdash \varphi}$$

This seems to be required quite frequently in examples to get easy proofs

Proof in structured specifications

We can give derived rules that encapsulate some of the transformations we need, for instance

$$\frac{SP \vdash \varphi \quad SP' \vdash \varphi' \quad \{\varphi, \sigma(\varphi')\} \vdash \sigma(\psi)}{SP|_{\sigma} \cup SP' \vdash \psi}$$

Uses $SP|_{\sigma} \cup SP' \equiv (SP \cup \sigma(SP'))|_{\sigma}$

A systematic approach along these lines:

T. Mossakowski, S. Autexier and D. Hutter. Development graphs: proof management for structured specifications. Journal of Logic and Algebraic Programming 2006

Entailment between specification

Need $SP \vdash SP'$ that is sound w.r.t. $\mathbf{Mod}(SP) \subseteq \mathbf{Mod}(SP')$

Why?

- Provides a system for transforming specifications for use in proving $SP \models \varphi$
- For proving correctness of simple refinements $SP \rightsquigarrow SP'$
- As a basis for $SP \rightsquigarrow_\kappa SP'$ later

Builds on $SP \vdash \varphi$:

$$\frac{SP \vdash \varphi_1 \quad \cdots \quad SP \vdash \varphi_n}{SP \vdash \langle \mathbf{Sig}(SP), \{\varphi_1, \ldots, \varphi_n\} \rangle}$$

Entailment between specification

Rules:

$$\frac{SP \vdash SP_1 \qquad SP \vdash SP_2}{SP \vdash SP_1 \cup SP_2}$$

$$\frac{SP'|_\sigma \vdash SP}{SP' \vdash \sigma(SP)}$$

$$\frac{\widehat{SP} \vdash SP'}{SP \vdash SP'|_\sigma} \quad \begin{array}{l} \sigma : SP \to \widehat{SP} \\ \text{admits model expansion} \end{array}$$

Theorem: If $SP \vdash \varphi$ is complete then $SP \vdash SP'$ is complete.

Admits model expansion

$$\frac{\widehat{SP} \vdash SP'}{SP \vdash SP'|_\sigma} \quad \begin{array}{l}\sigma : SP \to \widehat{SP} \\ \text{admits model expansion}\end{array}$$

$\sigma : SP \to \widehat{SP}$ admits model expansion:

- for any $M \in \mathbf{Mod}(SP)$
- … there exists $\widehat{M} \in \mathbf{Mod}(\widehat{SP})$)
- … such that $\widehat{SP}|_\sigma = M$

This is a sufficient condition for $\sigma : SP \to \widehat{SP}$ being conservative

Admits model expansion?

$$\textsc{Sort} = \textbf{export } \mathit{sort} \textbf{ from}$$
$$\textbf{enrich} \text{ specification of } \mathit{issorted} \textbf{ by}$$
$$\textbf{opns } \mathit{sort} : \mathit{list} \rightarrow \mathit{bool}$$
$$\textbf{axioms } \mathit{issorted}(\mathit{sort}(l))$$

Then we refine: $\textsc{Sort} \rightsquigarrow$ definition of sort

Need to show:

$$\text{definition of } \mathit{sort} \cup \text{definition of } \mathit{issorted} \vdash \mathit{issorted}(\mathit{sort}(l))$$

We require a *definition* of *issorted* to avoid inconsistency

Relates to $\exists$ for modelling hiding:

$$\text{definition of } \mathit{sort} \vdash \exists \mathit{issorted} : \mathit{list} \rightarrow \mathit{bool}.$$
$$\text{specification of } \mathit{issorted} \wedge \mathit{issorted}(\mathit{sort}(l)))$$

A proof requires a witness for *issorted*.

Entailment between specification via κ

Need $SP \vdash_\kappa SP'$ that is sound w.r.t. $SP' \rightsquigarrow_\kappa SP$, i.e. $\kappa : \mathbf{Mod}(SP) \to \mathbf{Mod}(SP')$

Regard κ as a specification-building operation

- $\mathbf{Sig}(\kappa(SP)) = \Sigma$, where $\kappa : \mathbf{Sig}(SP) \to \Sigma$
- $\mathbf{Mod}(\kappa(SP)) = \{\kappa(M) \mid M \in \mathbf{Mod}(SP)\}$

Reduces the problem to $SP \vdash SP'$:

$$\frac{\kappa(SP) \vdash SP'}{SP \vdash_\kappa SP'}$$

Main constructors of interest:

- $\cdot|_\sigma$ (rules already provided)
- Free extension
- "Anarchic" case, i.e. ML-style algebraic datatypes, is most important and easiest

Behavioural specifications

Motivating example: sets

- $S \cup T = T \cup S$
- $a \in S \cup T = a \in S \vee a \in T$

Are lists okay as a model of sets, with $\cup$ implemented as *append*?

- $append(a, b) \neq append(b, a)$

But the order of elements in a list isn't observable using set membership, so who cares?

An interpretation that reflects this is

- $\mathbf{Mod}_{\equiv}(\textsc{Set}) = \{M \mid M \equiv M' \in \mathbf{Mod}(\textsc{Set})\}$
- $M \equiv M'$ is behavioural equivalence: M and M' give the same value to any term of basic type (e.g. booleans)
- cf. indistinguishability, contextual equivalence

Behavioural specifications

We need another proof system: $SP \vdash_{\equiv} \varphi$

This is much more difficult, but there are methods available
One approach is via regarding equality in axioms as denoting indistinguishability $\sim$

This gives a new satisfaction relation $M \models_{\sim} \varphi$

- $M \models_{\sim} \varphi$ iff $M/{\sim} \models \varphi$
- $M \equiv M'$ iff $M/{\sim_M} \cong M/{\sim_{M'}}$

M. Bidoit and R. Hennicker. Constructor-based observational logic. Journal of Logic and Algebraic Programming 2006

Conclusion

- There is relevant work in algebraic specification going back to about 1980, with a well-investigated theory
- An interesting point is the independence of the structuring mechanisms from the logical system
  - not just the syntax of axioms
  - also the details of its type system
- I don't know if any of this, when seen from the proof assistant point of view, is new or surprising
- ... but a lot of it pre-dates most work on libraries in proof assistants (exception: Mizar)
- Relatively little work on implementation, compared with the proof assistant community
- ... but see work on development graphs

That advertisement again