Stainless Formal Verification

on the interdependency between mathematical foundations, semantics, and proof score programming

Răzvan Diaconescu

FSSV 2010, Kanazawa

Outline

Contents

1 Introduction **1**

2 Mathematical foundations: many sorted algebra **3**
- 2.1 Signatures, algebras, sentences, satisfaction . . . 3
- 2.2 Equational proof theory . . . 5
- 2.3 Induction . . . 6

3 Example: verification of termination of generic bubble sorting **8**
- 3.1 Formal specification . . . 8
  - 3.1.1 Natural numbers . . . 8
  - 3.1.2 Bubble sorting . . . 8
- 3.2 Proof of termination of topological (generic) bubble sorting . . . 9
  - 3.2.1 Specification of auxiliary functions . . . 10
- 3.3 Proof management . . . 10
- 3.4 Proof score programming . . . 11
- 3.5 Mind semantics! . . . 13

1 Introduction

Goal of lecture

Gain some understanding on what is a correct specification and verification process through clarification of

- mathematical foundations
- semantics
- specification
- proof score programming

and of the relationships between them.

Most of 'formal verification' practice, while assumed to be rigorous by definition, in reality still confused wrt some of above aspects, hence difficult to be trusted.

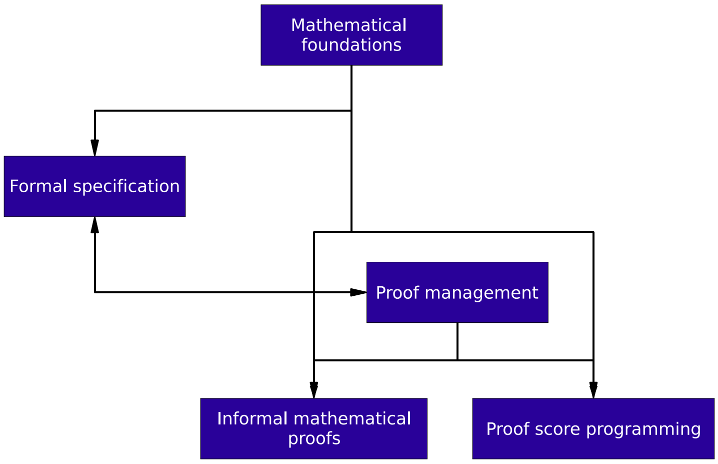
'Stainless' does not mean 'perfectly formal'

But rather

- being clear about what are the formal and the informal proof parts,
- justification of informal parts by solid clear mathematical arguments,
- proof scores for the formal proof parts strictly based upon mathematical foundations, and
- clean(!) mathematical foundations.

Specification and verification schema

Each level is defined and makes sense wrt the upper levels.



Mathematical foundations

- There is a formal logical system, including both model theory (for semantics) and proof theory. Very desirable that these constitute an *institution*.
- The model theory is sufficiently developed in order to support necessary specification properties.
- The eventual operational level of the proof theory (e.g. rewriting) is rigorously supported by mathematics.

Formal specification

- There is a formal specification language such that the language constructs correspond exactly to mathematical entities in the underlying logic.
- A specification consists of
  - a set of *axioms* in the underlying logic (this includes the specification of a corresponding signature), and
  - eventually, structuring constructs.
- Each such specification defines the *class of models* satisfying its axioms.
  - In the structured case, this is also determined by the structuring constructs (requires a bit of mathematical sophistication).

The whole point of formal specification:
axiomatic definition of certain classes of models.

Role of semantics and foundations

Tendency to forget this sense of formal specification and verification by neglecting semantics.

No semantics = No meaning!

This implies incorrect non-sense specification and verification.

Proof management

- Introduce auxiliary entities (e.g. functions, predicates) necessary for the proof; this means extension of the original specification.
- Separate the formal from the informal parts of the proof.
  - Informal parts get mathematical proofs.
  - Formal parts are proved by proof score running. Much higher 'horizontal' complexity than the informal parts.

Proof score programming

- Specification of the proof structure, including lemmas, conditions, proof tasks to be executed by the system, etc.
- Should be rigorously, directly and *transparently* based upon mathematical results lying foundations to corresponding proof methodologies.
  - In particular, this means to avoid abuse or even any use of extra-logical features of the language (such as ==, etc.)

2 Mathematical foundations: many sorted algebra

Many sorted algebra (MSA)

- It is the most classical logical system of algebraic specification.
  - Deeply rooted in conventional mathematics.
  - Originating from early mathematical foundations of semantics of programming languages and of abstract data types.
- Has very convenient model theoretic and computational properties, supporting high integration between the specification and the verification aspects.
- However, there are myriads of other logical systems used for formal specification, some of them just refinements of MSA, others quite different (at least through a gross perspective).

2.1 Signatures, algebras, sentences, satisfaction

Signatures

S-sorted signature (S,F)

- S – set of sort symbols,
- $F = \{F_{w\to s} \mid w \in S^*,\ s \in S\}$ – indexed family of operation symbols.

Algebras

(S,F)*-algebra* A consists of

- a set A_s for each $s \in S$, and
- a function $A_{\sigma:w\to s}:\ A_w \to A_s$ for each $\sigma \in F_{w\to s}$ (where $A_w = A_{s_1} \times \cdots \times A_{s_n}$, for $w = s_1 \dots s_n$).

(S,F)*-homomorphism* $h:\ A \to B$ consists of

- $h_s:\ A_s \to B_s$ for each $s \in S$,

such that the following *homomorphism condition*

$$h_s(A_\sigma(a)) = B_\sigma(h_w(a)).$$

for each $a \in A_w$.

Sentences

The set of (S,F)-*sentences* is the least set such that:

- Each (S,F)-equation $t = t'$ is an (S,F)-sentence.
- If ρ_1 and ρ_2 are (S,F)-sentences then
  - $\rho_1 \wedge \rho_2$ (*conjunction*),
  - $\rho_1 \vee \rho_2$ (*disjunction*),
  - $\rho_1 \Rightarrow \rho_2$ (*implication*) and
  - $\neg\rho_1$ (*negation*)

  are also (S,F)-sentences.
- If X is a set of variables for (S,F), then $(\forall X)\rho$ and $(\exists X)\rho$ are (S,F)-sentences whenever ρ is an $(S,F\cup X)$-sentence.

Satisfaction

Defined recursively on the structure of the sentences.

- $A \models t = t'$ if and only if $A_t = A_{t'}$ (where $A_{\sigma(t_1,\ldots,t_n)} = A_\sigma(A_{t_1},\ldots,A_{t_n})$),
- $A \models \rho_1 \wedge \rho_2$ if and only if $A \models \rho_1$ and $A \models \rho_2$,
- $A \models \rho_1 \vee \rho_2$ if and only if $A \models \rho_1$ or $A \models \rho_2$,
- $A \models \rho_1 \Rightarrow \rho_2$ if and only if $A \not\models \rho_1$ or $A \models \rho_2$,
- $A \models \neg\rho_1$ if and only if $A \not\models \rho_1$,
- $A \models_{(S,F)} (\forall X)\rho$ if and only if $A' \models_{(S,F\cup X)} \rho$ for each $(S,F\cup X)$-expansion A' of A, and
- $A \models (\exists X)\rho$ if and only if $A \not\models (\forall X)\neg\rho$.

Initial semantics

Initial algebra A in class $\mathscr{C}$ of algebras: for each $B \in \mathscr{C}$ there exists an unique $h: A \to B$.

Fact 1. *Initial algebras are unique up to isomorphisms.*

Theorem 2. *Each set of* conditional equations, *i.e. sentences of the form* $(\forall X)(t_1 = t_1') \wedge \cdots \wedge (t_n = t_n') \Rightarrow (t = t')$, *has an initial algebra.*

Initial algebras are the models of tight denotation specifications (`mod!` in CafeOBJ, `fmod` in Maude)

2.2 Equational proof theory

Entailment systems

Entailment relation (for a signature Σ) consists of a binary relation $\vdash_\Sigma$ between sets of Σ-sentences such that:

1. *union:* if $\Gamma \vdash_\Sigma \Gamma_1$ and $\Gamma \vdash_\Sigma \Gamma_2$ then $\Gamma \vdash_\Sigma \Gamma_1 \cup \Gamma_2$,
2. *monotonicity:* if $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \Gamma$, and
3. *transitivity:* if $\Gamma \vdash_\Sigma \Gamma_1$ and $\Gamma_1 \vdash_\Sigma \Gamma_2$ then $\Gamma \vdash_\Sigma \Gamma_2$.

An *entailment system* $\vdash$ consists of an entailment relation $\vdash_\Sigma$ for each signature Σ.

Example: semantic entailment

$\Gamma \models_{(S,F)} \Gamma'$ if and only if $A \models_{(S,F)} \Gamma$ implies $A \models_{(S,F)} \Gamma'$ for each (S,F)-algebra A.

Modus Ponens (meta-rule)

Meta-rules are properties of the entailment systems.

$$\Gamma \vdash_{(S,F)} (H \Rightarrow C) \text{ if and only if } \Gamma \cup H \vdash_{(S,F)} C.$$

The semantic entailment $\models$ has Modus-Ponens.

Universal Quantification (meta-rule)

$$\Gamma \vdash_{(S,F)} (\forall X)\rho \text{ if and only if } \Gamma \vdash_{(S,F\cup X)} \rho.$$

The semantic entailment $\models$ has Universal Quantification.

Equational proof rules

Reflexivity: $\dfrac{\emptyset}{t = t}$

Symmetry: $\dfrac{\{t = t'\}}{t' = t}$

Transitivity: $\dfrac{\{t = t',\ t' = t''\}}{t = t''}$

Congruence: $\dfrac{\{t_i = t'_i \mid 1 \le i \le n\}}{\sigma(t_1,\ldots,t_n) = \sigma(t'_1,\ldots,t'_n)}$

Substitutivity: $\dfrac{\{(\forall X)H \Rightarrow C\}}{\{\theta(H \Rightarrow C)\}} \quad \theta : X \to T_{(S,F)}.$

The equational entailment system

Definition 3 (*entailment system for conditional equations*, $\vdash^{\mathrm{e}}$)**.** The least entailment system, containing the 5 proof rules and satisfying the 2 meta-rules above.

Unlike $\models$, $\vdash^{\mathrm{e}}$ is finitely defined. This makes $\vdash^{\mathrm{e}}$ usable for formal proofs.

Soundness

An absolutely necessary property for any logical system, constitutes the basis for the correctness of formal verification.

Not very hard to establish.

Theorem 4 (*Soundness of equational deduction*)**.** $\Gamma \vdash^{\mathrm{e}}_{(S,F)} \rho$ *implies* $\Gamma \models_{(S,F)} \rho$.

Completeness

A very desirable property of logical systems, but not absolutely necessary.

In general, much harder to establish than soundness.

Theorem 5 (*Completeness of equational deduction*)**.** $\Gamma \models_{(S,F)} \rho$ *implies* $\Gamma \vdash^{\mathrm{e}}_{(S,F)} \rho$.

Soundness and completeness means that $\models$ (for conditional equations) and $\vdash^{\mathrm{e}}$ are the same, hence, very importantly, we have a finitary definition of $\models$ (for conditional equations).

Rewriting

The standard way to mechanize equational deduction, to have it as a computation process.

Abandons *Symmetry* and replaces *Substitutivity* and *Congruence* by the following rule:

$$\textit{Rewriting:} \quad \frac{\{(\forall X)H \Rightarrow (t = t')\} \cup \theta(H)}{c[\theta(t)] = c[\theta(t')]}$$

In general completeness is lost, however under conditions such as *confluence* and *termination*, completeness may be retained.

2.3 Induction

Inductive properties

These are the properties satisfied by the initial algebra of a given set Γ of conditional equations, i.e. $0_\Gamma \models \rho$.

For example, for the specification Γ as follows:

```
0 : -> Nat
s : Nat -> Nat
+ : Nat Nat -> Nat
(∀ X) X + 0 = X
(∀ X,Y) X + s(Y) = s(X + Y)
```

we have that $0_\Gamma \models (\forall \mathtt{X})\mathtt{0 + X} = \mathtt{X}$ but $\Gamma \not\models (\forall \mathtt{X})\mathtt{0 + X} = \mathtt{X}$. Hence, direct ordinary (equational) deduction not enough for proving inductive properties.

Constructors

A great device for reducing the complexity of proofs of inductive properties.

Sub-signature of constructors

Signature (S,F), Γ a set of conditional equations for (S,F). (S,F^c) is a *sub-signature of constructors for* Γ when

- $F^c_{w\to s} \subseteq F_{w\to s}$, and
- $0_{(S,F^c)} \to 0_\Gamma\restriction_{(S,F^c)}$ surjective.

Example: `0` and `s` form a sub-signature of constructors for the specification above.

Sufficient completeness

The following equivalent characterization for sub-signature of constructors has two advantages:

1. does not depend on existence of initial algebras for Γ, hence applicable within more general situations, and
2. it gives a method to actually prove the constructors property.

Proposition 6. *(S,F^c) is a sub-signature of constructors for Γ if and only if for each (S,F)-term t there exists an (S,F^c)-term t' such that $\Gamma \models_{(S,F)} t = t'$.*

Proving inductive properties

The following gives a sufficient condition for proving inductive properties by an *infinite* set of ordinary proofs.

Theorem 7.

1. *Γ set of conditional equations for a signature (S,F),*
2. *(S,F^c) sub-signature of constructors for Γ, and*

3. *('lemmas':) E set of any sentences such that $0_\Gamma \models E$.*

Then for any $(S,F\cup X)$-sentence ρ

$$0_\Gamma \models (\forall X)\rho \ \ \textit{if} \ \ \Gamma\cup E \models \theta(\rho) \ \ \textit{for all substitutions} \ \ \theta: \ X \to T_{(S,F^c)}.$$

Structural induction

Theorem 8 (*sufficient **finitary** proof method for inductive properties*)**.**

1. *(S,F^c) sub-signature of constructors for Γ **any** set of (S,F)-sentences,*
2. *X finite set of variables for (S,F),*
3. *ρ **any** $(S,F\cup X)$-sentence.*

If for any sort preserving mapping $Q: \ X \to F^c$

$$\Gamma\cup\{\psi(\rho) \mid \psi: \ X\to Z = \cup_{x\in X} Z_x \ \ \textit{with} \ \ \psi(x)\in Z_x\} \models_{(S,F\cup Z)} Q^\sharp(\rho)$$

where

- *Z_x are strings of variables for the arguments of Q_x such that $Z_{x1}\cap Z_{x2} = \emptyset$ for $x1 \neq x2 \in X$, and*
- *$Q^\sharp$ is the substitution $X \to T_{(S,F^c\cup Z)}$ defined by $Q^\sharp(x) = Q_x(Z_x)$,*

then $\Gamma \models_{(S,F)} \theta(\rho)$ for all substitutions $\theta: \ X \to T_{(S,F^c)}$.

The finitary character of structural induction

Finite number of proof goals always involving finite conditions because:

- for any Q, finite Z because finite X and finite arities of operations, hence finite $\{\psi \mid \psi: \ X \to Z\}$, and
- if finite F^c, then finite $\{Q \mid Q: \ X \to F^c\}$ since finite X .
  - moreover smaller F^c implies fewer proof goals.

Generality of foundations

Many of MSA mathematical concepts and results above can be interpreted in the same form in other logical systems (such as *preordered algebra* for specification with transitions).

For example, the structural induction method above has such general character.

This means foundations can be mathematically developed at the level of abstract *institutions*.

3 Example: verification of termination of generic bubble sorting

Example of 'stainless' formal verification

We illustrate

- formal specification based rigorously upon mathematical foundations,
- proof management,
- proof score building based rigorously upon equational proof theory and the structural induction theorem above,
- the inter-dependency between semantic-oriented specification and proof score programming.

3.1 Formal specification

3.1.1 Natural numbers

The natural numbers with the usual zero and succesor operations constitute the initial algebra of `PNAT=`. Moreover we have a specification of the equality of numbers

```
mod! PNAT= {
  [ Nat ]
  op 0 :  -> Nat
  op s_ : Nat -> Nat
  op _=_ : Nat Nat -> Bool {comm}
  vars M N : Nat
  eq ((s M) = 0) = false .
  eq (0 = 0) = true .
  eq [succ=] : (s M = s N) = (M = N) .
}
```

The following adds a specification of the ‘strictly less than’ relation on the naturals numbers.

```
mod! PNAT< {
  protecting(PNAT=)
  op _<_ : Nat Nat -> Bool
  vars M N : Nat
  eq [succ<] : (s M) < (s N) = M < N .
  eq 0 < (s M) = true .
  eq M < 0 = false .
}
```

3.1.2 Bubble sorting

The strings of natural numbers with concatenation and empty string constitute the initial algebra of `STRG-PNAT`.

```
mod! STRG-PNAT {
  [ Nat < Strg ]
  op nil :  -> Strg
  op _;_ : Strg Strg -> Strg {assoc id: nil}
}
```

Bubble sorting algorithm appears as an instance of rewriting modulo associativity (of concatenation). The initial model is the *preordered algebra* that has strings of naturals as elements and the transitions given by the sorting algorithm as the preorder relation.

```
mod! SORTING-STRG-PNAT {
  protecting(STRG-PNAT))
  ctrans E:Nat ; E’:Nat => E’ ; E  if (E’ < E) .
}
```

We may use this specification as an actual sorting (very high level) program by executing it by rewriting modulo associativity.

```
exec s s s 0 ; s 0 ; 0 ; s s 0 .
```

However, if we look more carefully into the specification of bubble sorting, we note that it essentially requires only a binary relation $<$, no commitement to any property of the naturals, not even to the naturals as elements of the strings. This means bubble sorting is very general, has a *generic* nature. Such kind of sorting over binary relations is sometimes known as *topological* sorting.

The following parameterized module specifies strings over *any* set (`Elt`) of elements.

```
mod! STRG (X :: TRIV) {
  [ Elt < Strg ]
  op nil :  -> Strg
  op _;_ : Strg Strg -> Strg {assoc id: nil}
}
```

The following specifies a generic binary relation < on the elements, to be used for the sorting. We also specify a loose negation of `<`, namely `not<`, mainly for operational reasons. Note that the sentence specifying `not<` is *not* a conditional equation (although CafeOBJ notation refers to it as conditional equation); however this is OK since this is loose semantics specification, which does not require existence of initial algebras.

```
mod* PSEUDO-ORDER {
  [ Elt ]
  op _<_ : Elt Elt -> Bool
  op _not<_ : Elt Elt -> Bool
  vars E1 E2 E3 : Elt
  cq (E1 not< E2) = true if E2 < E1 or not(E1 < E2) .
}
```

The following instantiate the above specified pseudo-orders to the standard ordering of the natural numbers. It uses default mapping mechanism, hence only the mapping of `not<` needs to be specified explicitly. The view specification requires a proof that the initial algebra of `PNAT<` satisfies the axiom of `PSEUDO-ORDER` through the translation given by the view. This means an inductive proof, that can be done by proof score programming based upon the Structural Induction Theorem; we skip this here. Note that in this case we have that proof score programming is involved at the stage of specification writing.

```
view PNAT<asPO from PSEUDO-ORDER to PNAT<
     {op (E:Elt not< E’:Elt) -> ((E:Nat = E’:Nat) or (E’ < E))} .
```

The following is the specification of generic bubble sorting algorithm.

```
mod! SORTING-STRG(Y :: PSEUDO-ORDER) {
  protecting(STRG(Y))
  ctrans E:Elt ; E’:Elt => E’ ; E  if (E’ < E) .
}
```

`SORTING-STRG-PNAT` can be obtained as an instance of `SORTING-STRG` by the above defined view `PNAT<asPO`.

```
select SORTING-STRG(PNAT<asPO) .
exec (s s s s 0 ; s s s 0 ; s s s s s 0 ; s s s 0 ; s 0) .
```

3.2 Proof of termination of topological (generic) bubble sorting

For this we go back to the specification level and define a function `disorder :  Strg -> Nat` such that

$$(\forall S,S')[S\texttt{=>}S' \text{ implies } \texttt{disorder}(S') < \texttt{disorder}(S)].$$

(note this is a Horn clause in the *POA*, the logical system of preordered algebra). Then the mathematical argument of well-foundness of the natural numbers leads to the informal mathematical proof of termination.

3.2.1 Specification of auxiliary functions

The definition of function ‘disorder’ requires specification of other auxiliary functions too:

```
mod! PNAT+ {
  protecting(PNAT=)
  op _+_ : Nat Nat -> Nat
  vars M N : Nat
  eq [succ+] : M + (s N) = s(M + N) .
  eq M + 0 = M .
}
```

- E»S computes how many elements of S are less than E.
- `disorder`(S) computes how many steps of the sorting algorithm are needed for the sorting of S.

```
mod! SORTING-DISORDER (Y :: PSEUDO-ORDER) {
  protecting(SORTING-STRG(Y) + PNAT+)
  op _>>_ : Elt Strg -> Nat
  op disorder : Strg -> Nat
  vars E E' : Elt
  vars S S' : Strg
  eq E >> nil = 0 .
  cq E >> E' =  s 0 if (E' < E) .
  cq E >> E' =  0   if (E' not< E) .
  eq E >> (S ; S') = (E >> S) + (E >> S') .
  eq disorder(E) = 0 .
  eq disorder(E ; S) = disorder(S) + (E >> S) .
}

select SORTING-DISORDER(PNAT<asPO) .
red disorder(s s s 0 ; s 0 ; s s 0) .
```

The following specifies an equivalence relation on strings defined by

$$S \mathtt{<>} S' \text{ if and only if } (\forall E) E \text{»} S = E \text{»} S'.$$

Although this is not required by the specification of `disorder`, it is used in the formal proofs below. As this is beyond CafeOBJ logic, at this level we under-specify it, however we will use its complete definition in the proof scores.

```
mod* SORTING<> (Y :: PSEUDO-ORDER) {
  protecting(SORTING-DISORDER(Y))
  op _<>_ : Strg Strg -> Bool
}
```

3.3 Proof management

The proof management of this problem means the following:

1. By an mathematical argument we have reduced the task of proving termination to proving a Horn clause sentence as inductive property in *POA*.
2. Extension of the original specification with new functions.
3. Mathematical proof of the fact that

   $$(\forall S, S', E1, E2) \mathtt{disorder}(S;E1;E2;S') < \mathtt{disorder}(S;E2;E1;S') \text{ if } E1 < E2$$

   implies

   $$(\forall S, S')\ [S \mathtt{=>} S' \text{ implies } \mathtt{disorder}(S') < \mathtt{disorder}(S)].$$

   (This mathematical argument is related to the theory of rewriting modulo axioms.)
4. Formal proof (by proof score programming and running) of

   $$(\forall S, S', E1, E2)\ [\mathtt{disorder}(S;E1;E2;S') < \mathtt{disorder}(S;E2;E1;S') \text{ if } E1 < E2]$$

   This requires several lemmas and tranformation of the proof tasks by meta-rules such as *Universal Quantification* or *Modus-Ponens*.
5. Mathematical proofs for sub-signatures of constructors.

3.4 Proof score programming

Formal proof of

$$(\forall S,S',E1,E2)\mathbf{disorder}(S;E1;E2;S') < \mathbf{disorder}(S;E2;E1;S') \ \ \mathbf{if} \ \ E1 < E2$$

```
open SORTING<> + PNAT< .
  vars E E' : Elt
  vars S S1 S2 : Strg
  vars M N P : Nat
```

We use a couple of lemmas:

```
  cq [Lemma-1] : disorder(S ; S1) < disorder(S ; S2) = true
     if S1 <> S2 and disorder(S1) < disorder(S2) .
  eq [Lemma-2] : (E ; E' ; S) <> (E' ; E ; S) = true .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
  ops e1 e2 :  -> Elt .
  ops s s' :  -> Strg .
```

By virtue of *Modus Ponens* we introduce the condition of the theorem.

```
  eq e1 < e2 = true .
```

We use another couple of lemmas for naturals.

```
-- [Lemma-3] :
  op _+_ : Nat Nat -> Nat {assoc comm}
  eq [Lemma-4] : M < s M = true .
```

The theorem is now proved by equational deduction (rewriting).

```
red disorder(s ; e1 ; e2 ; s') < disorder(s ; e2 ; e1 ; s') .
close
```

Formal proof of `Lemma-1`.

$$(\forall S,S1,S2)\mathbf{disorder}(S;S1) < \mathbf{disorder}(S;S2) \ \ \mathbf{if} \ \ S1 \ \mathbf{<>} \ S2 \ \ \mathbf{and} \ \ \mathbf{disorder}(S1) < \mathbf{disorder}(S2)$$

By direct application of the Structural Induction Theorem where

- $X = \{S\}$, and
- ρ is $(\forall S1,S2)\mathtt{disorder}(S;S1) < \mathtt{disorder}(S;S2)$ if $S1$ <> $S2$ and $\mathtt{disorder}(S1) < \mathtt{disorder}(S2)$.

The sub-signature of constructors consists of `nil`, all `e:Elt`, and `_;_`, this being established by mathematical proof.
— The case $Q_S =$ `nil`:

```
open SORTING<> + PNAT< .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
   ops s1 s2 :  -> Strg .
```

By virtue of Modus Ponens we introduce the condition of the theorem.

```
  eq disorder(s1) < disorder(s2) = true .
  eq E >> s1 = E >> s2 .
```

The proof of the conclusion.

```
red disorder(nil ; s1) < disorder(nil ; s2) .
close
```

— The case $Q_S =$ `e:Elt` :

```
open SORTING<> + PNAT< .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
  ops s1 s2 :  -> Strg .
  op e :  -> Elt .
  var E : Elt
  vars M N P : Nat
```

By virtue of *Modus Ponens* we introduce the condition of the theorem.

```
  eq disorder(s1) < disorder(s2) = true .
  eq E >> s1 = E >> s2 .
```

We use again `[Lemma-3]` :

```
  op _+_ : Nat Nat -> Nat {assoc comm}
```

And use another lemma for the naturals:

```
  cq [Lemma-5] : M + N < P + N = true if M < P .
```

The proof of the conclusion.

```
red disorder(e ; s1) < disorder(e ; s2) .
close
```

— The case $Q_S =$ `_;_` :
We have that $Z = \{x, y\}$.

```
open SORTING<> + PNAT< .
  ops x y :  -> Strg .
  vars S S1 S2 : Strg
```

We introduce the hypotheses for the condition of the Structural Induction Theorem :

```
  cq disorder(x ; S1) < disorder(x ; S2) = true
        if disorder(S1) < disorder(S2) and S1 <> S2 .
  cq disorder(y ; S1) < disorder(y ; S2) = true
        if disorder(S1) < disorder(S2) and S1 <> S2 .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
  ops s1 s2 : -> Strg .
```

By virtue of *Modus Ponens* we introduce the condition of the theorem.

```
  eq s1 <> s2 = true .
  eq disorder(s1) < disorder(s2) = true .
```

We use a new lemma:

```
  cq [Lemma-6] : S ; S1 <> S ; S2 = true if S1 <> S2 .
```

The proof of the conclusion.

```
red disorder(x ; y ; s1) < disorder(x ; y ; s2) .
close
```

Formal proof of `Lemma-2` :

$$(\forall E,E',S)(E;E';S) \texttt{<>} (E';E;S) = \texttt{true}$$

which means

$$(\forall E1,E,E',S)\ E1 \text{»} (E;E';S) = E1 \text{»} (E';E;S)$$

```
open SORTING<> + PNAT< .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
  ops e1 e e' :  -> Elt .
  op s :  -> Strg .
```

We use again `[Lemma-3]` :

```
  op _+_ : Nat Nat -> Nat {assoc comm}
```

And use a new lemma on naturals:

```
  eq [Lemma-7] : (M:Nat = M) = true .
```

The proof of the conclusion:

```
red e1 >> e ; e' ; s = e1 >> e' ; e ; s .
close
```

Formal proof of `Lemma-6` :

$$(\forall S,S1,S2)\ S;S1 \texttt{<>} S;S2 = \texttt{true if}\ \ S1 \texttt{<>} S2.$$

```
open SORTING<> + PNAT< .
```

By *Universal Quantification* we transform the theorem into a quantifier-free sentence.

```
  ops s s1 s2 :  -> Strg .
  op e :  -> Elt .
```

We introduce the condition of the theorem:

```
  var E : Elt
  eq E >> s1 = E >> s2 .
```

We use a new lemma on naturals:

```
  eq [Lemma-7] : (M:Nat = M) = true .
```

The proof of the conclusion:

```
red e >> s ; s1 = e >> s ; s2 .
close
```

3.5 Mind semantics!

The relation $a < b$ and $b < a$ on $\{a,b\}$ determines a model of `PSEUDO-ORDER`, however the bubble sorting algorithm for this instance is not terminating $a;b$=>$b;a$=>$a;b$=>... In spite of the fact that the proof is correctly built. Is there a gap somewhere?

We have both that $a < b$ and a `not<` b, which implies $0 = \texttt{s}\ \ 0$, hence the naturals are collapsed, and (by `PNAT=`) the Booleans also, which means the specification is inconsistent (it lacks models).

This situation is repaired by adding one more equation with only semantic role, not used in the proofs.

```
mod* PSEUDO-ORDER {
  [ Elt ]
  op _<_ : Elt Elt -> Bool
  op _not<_ : Elt Elt -> Bool
  vars E1 E2 E3 : Elt
  cq (E1 not< E2) = true if E2 < E1 or not(E1 < E2) .
  eq (E1 < E2) and (E1 not< E2) = false .
}
```

Conclusion: semantics of specifications comes first, correctness of proof scores depends on the semantic correctness of the specifications. For this it may be necessary to write axioms with absolutely no operational meaning, with only semantic meaning.