Software specification in CASL - The Common Algebraic Specification Language

Till Mossakowski, Lutz Schröder

October 2006

SFB/TR 8 SPATIAL COGNITION

Semantics of CASL basic specifications

- Who knows what first-order logic is?

- Who knows what first-order logic is?
- Who knows what a first-order structure (model) is?

The CASL logic (institution)

- Signatures: a signature provides the vocabulary

The CASL logic (institution)

- Signatures: a signature provides the vocabulary
- Signature morphisms: for extending and renaming signatures

The CASL logic (institution)

- Signatures: a signature provides the vocabulary
- Signature morphisms: for extending and renaming signatures
- Models: interpret the vocabulary of a signature with mathematical objects (sets, functions, relations)

The CASL logic (institution)

- Signatures: a signature provides the vocabulary
- Signature morphisms: for extending and renaming signatures
- Models: interpret the vocabulary of a signature with mathematical objects (sets, functions, relations)
- Sentences (formulae): for axiomatizing models denote true or false in a given model

The CASL logic (institution)

- Signatures: a signature provides the vocabulary
- Signature morphisms: for extending and renaming signatures
- Models: interpret the vocabulary of a signature with mathematical objects (sets, functions, relations)
- Sentences (formulae): for axiomatizing models denote true or false in a given model
- Terms: parts of sentences, denote data values

The CASL logic (institution)

- Signatures: a signature provides the vocabulary
- Signature morphisms: for extending and renaming signatures
- Models: interpret the vocabulary of a signature with mathematical objects (sets, functions, relations)
- Sentences (formulae): for axiomatizing models denote true or false in a given model
- Terms: parts of sentences, denote data values
- Satisfaction of sentences in models

CASL many-sorted signatures

- a set S of sorts,

CASL many-sorted signatures

- a set S of sorts,
- an $S^* \times S$-indexed set $(TF_{w,s})_{w,s \in S^* \times S}$ of total operation symbols,

CASL many-sorted signatures

- a set S of sorts,
- an $S^* \times S$-indexed set $(TF_{w,s})_{w,s \in S^* \times S}$ of total operation symbols,
- an $S^* \times S$-indexed set $(PF_{w,s})_{w,s \in S^* \times S}$ of partial operation symbols, such that $TF_{w,s} \cap PF_{w,s} = \emptyset$,

CASL many-sorted signatures

- a set S of sorts,
- an $S^* \times S$-indexed set $(TF_{w,s})_{w,s \in S^* \times S}$ of total operation symbols,
- an $S^* \times S$-indexed set $(PF_{w,s})_{w,s \in S^* \times S}$ of partial operation symbols, such that $TF_{w,s} \cap PF_{w,s} = \emptyset$,
- an S^*-indexed set $(P_w)_{w \in S^*}$ of predicate symbols

Signature morphisms map these components in a compatible way

Example signatures

- $\Sigma^{Nat} = (\{Nat\}, \{0 : Nat, succ\colon Nat \longrightarrow Nat\},$
  $\{pre\colon Nat \longrightarrow ? Nat\}, \emptyset)$

- $(\{Elem\}, \emptyset, \emptyset, \{\_\_ < \_\_ : Elem * Elem\})$

- $(\{Elem, List\},$
  $\{Nil : Elem, Cons\colon Elem * List \longrightarrow List\}, \emptyset, \emptyset)$

CASL many-sorted models

For a many-sorted signature $\Sigma = (S, TF, PF, P)$ a many-sorted model $M \in \mathbf{Mod}(\Sigma)$ consists of

- a non-empty carrier set s^M for each sort $s \in S$ (let w^M denote the Cartesian product $s_1^M \times \cdots \times s_n^M$ when $w = s_1 \ldots s_n$),

CASL many-sorted models

For a many-sorted signature $\Sigma = (S, TF, PF, P)$ a many-sorted model $M \in \mathbf{Mod}(\Sigma)$ consists of

- a non-empty carrier set s^M for each sort $s \in S$ (let w^M denote the Cartesian product $s_1^M \times \cdots \times s_n^M$ when $w = s_1 \ldots s_n$),
- a partial function f^M from w^M to s^M for each function symbol $f \in TF_{w,s}$ or $f \in PF_{w,s}$, the function being required to be total in the former case,

CASL many-sorted models

For a many-sorted signature $\Sigma = (S, \mathit{TF}, \mathit{PF}, P)$ a many-sorted model $M \in \mathbf{Mod}(\Sigma)$ consists of

- a non-empty carrier set s^M for each sort $s \in S$ (let w^M denote the Cartesian product $s_1^M \times \cdots \times s_n^M$ when $w = s_1 \ldots s_n$),
- a partial function f^M from w^M to s^M for each function symbol $f \in \mathit{TF}_{w,s}$ or $f \in \mathit{PF}_{w,s}$, the function being required to be total in the former case,
- a predicate $p^M \subseteq w^M$ for each predicate symbol $p \in P_w$.

Example Σ^{Nat}-models

- $Nat^M = \mathbb{N}$, 0^M=0, $suc^M(x) = x + 1$,
  $pre^M(x) = \begin{cases} x - 1, x > 0 \\ \text{undefined, otherwise} \end{cases}$

Example Σ^{Nat}-models

- $Nat^M = I\!N$, 0^M=0, $suc^M(x) = x + 1$,
  $$pre^M(x) = \begin{cases} x-1, x>0 \\ \text{undefined, otherwise} \end{cases}$$

- $Nat^N = I\!N \cup \{\infty\}$, 0^N=0,
  $$suc^N(x) = \begin{cases} \infty, & \text{if } x = \infty \\ x+1, & \text{otherwise} \end{cases},$$
  $$pre^N(x) = \begin{cases} x-1, & \text{if } 0 < x \neq \infty \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Example Σ^{Nat}-models

- $Nat^M = I\!N$, 0^M=0, $suc^M(x) = x + 1$,
  $$pre^M(x) = \begin{cases} x - 1, x > 0 \\ \text{undefined, otherwise} \end{cases}$$

- $Nat^N = I\!N \cup \{\infty\}$, 0^N=0,
  $$suc^N(x) = \begin{cases} \infty, & \text{if } x = \infty \\ x + 1, & \text{otherwise} \end{cases},$$
  $$pre^N(x) = \begin{cases} x - 1, & \text{if } 0 < x \neq \infty \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- $Nat^T = \{*\}$, $0^T = *$, $suc^T(*) = *$, $pre^T(*) = *$

Example Σ^{Nat}-models

- $Nat^M = I\!N$, 0^M=0, $suc^M(x) = x + 1$,
  $pre^M(x) = \begin{cases} x - 1, x > 0 \\ \text{undefined, otherwise} \end{cases}$

- $Nat^N = I\!N \cup \{\infty\}$, 0^N=0,
  $suc^N(x) = \begin{cases} \infty, & \text{if } x = \infty \\ x + 1, & \text{otherwise} \end{cases}$,
  $pre^N(x) = \begin{cases} x - 1, & \text{if } 0 < x \neq \infty \\ \text{undefined,} & \text{otherwise} \end{cases}$

- $Nat^T = \{*\}$, $0^T = *$, $suc^T(*) = *$, $pre^T(*) = *$

- $Nat^K = \mathbb{N}$, $0^N = K$, $suc^K(x) = x$,
  $$pre^K(x) = \begin{cases} y, & \text{if TM } x \text{ outputs } y \text{ on input } x \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- $Nat^K = \mathbb{N}$, $0^N = K$, $suc^K(x) = x$,
  $pre^K(x) = \begin{cases} y, & \text{if TM } x \text{ outputs } y \text{ on input } x \\ \text{undefined}, & \text{otherwise} \end{cases}$

- $Nat^F = \mathbb{N} \to \mathbb{N}$, $0^F(x) = 0$, $suc^F(f)(x) = f(x) + 1$,
  $pre^F(f)$ undefined for each f

CASL many-sorted terms

Given a signature Σ and a variable system $(X_s)_{s \in S}$, the set of terms is defined inductively as follows:

- variables $x \in X_s$ are terms of sort s

CASL many-sorted terms

Given a signature Σ and a variable system $(X_s)_{s \in S}$, the set of terms is defined inductively as follows:

- variables $x \in X_s$ are terms of sort s
- applications $f_{w,s}(t_1, \ldots, t_n)$ is a term of sort s, if $f \in TF_{w,s} \cup PF_{w,s}$ and t_i is a term of sort s_i, $w = s_1 \ldots s_n$.

Semantics of terms

Given a Σ-model and a variable valuation $\nu\colon X \longrightarrow M$, the semantics $\nu^{\#}$ of terms is defined as follows:

- variables $\nu^{\#}(x) = \nu(x)$

Semantics of terms

Given a Σ-model and a variable valuation $\nu \colon X \longrightarrow M$, the semantics $\nu^{\#}$ of terms is defined as follows:

- variables $\nu^{\#}(x) = \nu(x)$
- applications $\nu^{\#}(f_{w,s}(t_1, \ldots, t_n)) = f^M_{w,s}(\nu^{\#}(t_1), \ldots, \nu^{\#}(t_n))$ if all components are defined (undefined otherwise)

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$
- existential equations $t_1 \stackrel{e}{=} t_2$

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$
- existential equations $t_1 \stackrel{\mathsf{e}}{=} t_2$
- predications $p_w(t_1, \ldots, t_n)$

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$
- existential equations $t_1 \stackrel{e}{=} t_2$
- predications $p_w(t_1, \ldots, t_n)$
- definedness assertions $def(t)$

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$
- existential equations $t_1 \stackrel{e}{=} t_2$
- predications $p_w(t_1, \ldots, t_n)$
- definedness assertions $def(t)$
- conjunctions, disjunctions, implications, equivalences of formulae

CASL formulae

The set of (Σ, X)-formulae is defined inductively as follows:

- strong equations $t_1 = t_2$
- existential equations $t_1 \stackrel{e}{=} t_2$
- predications $p_w(t_1, \ldots, t_n)$
- definedness assertions $def(t)$
- conjunctions, disjunctions, implications, equivalences of formulae
- universal, existential, unique-existential quantifications

Satisfaction of atomic formulae

A formula φ is satisfied in a model M w.r.t. a valuation $\nu\colon X \longrightarrow M$ (short notation: $M, \nu \models \varphi$), if

- $M, \nu \models t_1 = t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ or both sides are undefined,

Satisfaction of atomic formulae

A formula φ is satisfied in a model M w.r.t. a valuation $\nu\colon X \longrightarrow M$ (short notation: $M, \nu \models \varphi$), if

- $M, \nu \models t_1 = t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ or both sides are undefined,
- $M, \nu \models t_1 \stackrel{e}{=} t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ and both sides defined,

Satisfaction of atomic formulae

A formula φ is satisfied in a model M w.r.t. a valuation $\nu\colon X \longrightarrow M$ (short notation: $M, \nu \models \varphi$), if

- $M, \nu \models t_1 = t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ or both sides are undefined,
- $M, \nu \models t_1 \stackrel{e}{=} t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ and both sides defined,
- $M, \nu \models p_w(t_1, \ldots, t_n)$
  if $(\nu^{\#}(t_1), \ldots, \nu^{\#}(t_n))$ is defined and $\in p_w^M$,

Satisfaction of atomic formulae

A formula φ is satisfied in a model M w.r.t. a valuation $\nu\colon X \longrightarrow M$ (short notation: $M, \nu \models \varphi$), if

- $M, \nu \models t_1 = t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ or both sides are undefined,
- $M, \nu \models t_1 \stackrel{e}{=} t_2$ if $\nu^{\#}(t_1) = \nu^{\#}(t_2)$ and both sides defined,
- $M, \nu \models p_w(t_1, \ldots, t_n)$
  if $(\nu^{\#}(t_1), \ldots, \nu^{\#}(t_n))$ is defined and $\in p_w^M$,
- $M, \nu \models def(t)$ if $\nu^{\#}(t)$ is defined

Satisfaction of compound formulae

A standard in first-order logic, i.e.

- a conjuction is satisfied iff all the conjuncts are satisfied

Satisfaction of compound formulae

A standard in first-order logic, i.e.

- a conjuction is satisfied iff all the conjuncts are satisfied
- similar for disjunction etc.

Satisfaction of compound formulae

A standard in first-order logic, i.e.

- a conjuction is satisfied iff all the conjuncts are satisfied
- similar for disjunction etc.
- a universal (existential) quantification is satisfied when all (some) of the changes of the valuation for the quantified variable lead to satisfcation in the model:
  $M, \nu \models \forall x : s \,.\, \phi$ iff $M, \xi \models \phi$ for all valuation ξ that differ from ν only on $x : s$

Satisfaction of closed formulae

A closed formula (sentences) is satisfied in a model iff it is satisfied w.r.t. the empty valuation:

$$M \models \varphi \text{ iff } M, \emptyset \models \varphi$$

Sort generation constraints

A Σ-sort-generation constraint (S', F') consists of

Sort generation constraints

A Σ-sort-generation constraint (S', F') consists of

- a set of sorts $S' \subseteq S$

Sort generation constraints

A Σ-sort-generation constraint (S', F') consists of

- a set of sorts $S' \subseteq S$
- a set of (qualified) operation symbols $F' \subseteq TF \cup PF$

Sort generation constraints

A Σ-sort-generation constraint (S', F') consists of

- a set of sorts $S' \subseteq S$
- a set of (qualified) operation symbols $F' \subseteq TF \cup PF$

$M \models (S', F')$ iff the carriers of sorts in S' are generated by terms in F' (with variables of sorts outside S')

Sort generation constraints

A Σ-sort-generation constraint (S', F') consists of

- a set of sorts $S' \subseteq S$
- a set of (qualified) operation symbols $F' \subseteq TF \cup PF$

$M \models (S', F')$ iff the carriers of sorts in S' are generated by terms in F' (with variables of sorts outside S')
i.e. for each $s \in S'$, $a \in s^M$, there is some term t (with variables of sorts outside S') and some valuation ν with $\nu^{\#}(t) = a$.

Example Σ^{Nat}-models

- $Nat^M = I\!\!N$, 0^M=0, $suc^M(x) = x + 1$,
  $$pre^M(x) = \begin{cases} x - 1, x > 0 \\ \text{undefined, otherwise} \end{cases}$$

- $Nat^N = I\!\!N \cup \{\infty\}$, 0^N=0,
  $$suc^N(x) = \begin{cases} \infty, & \text{if } x = \infty \\ x + 1, & \text{otherwise} \end{cases},$$
  $$pre^N(x) = \begin{cases} x - 1, & \text{if } 0 < x \neq \infty \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- $Nat^T = \{*\}$, $0^T = *$, $suc^T(*) = *$, $pre^T(*) = *$

- $Nat^K = I\!N$, $0^N = K$, $suc^K(x) = x$,
  $$pre^K(x) = \begin{cases} y, & \text{if TM } x \text{ outputs } y \text{ on input } x \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- $Nat^F = I\!N \to I\!N$, $0^F(x) = 0$, $suc^F(f)(x) = f(x) + 1$,
  $pre^F(f)$ undefined for each f

Σ^{Nat}-models revisited

- $M \models (\{Nat\}, \{0, succ\})$ because
  Nat^M consists solely of interpretations of successor terms
  0, $succ(0)$, $succ(succ(0))$, . . . ("no junk")

Σ^{Nat}-models revisited

- $M \models (\{Nat\}, \{0, succ\})$ because Nat^M consists solely of interpretations of successor terms 0, $succ(0)$, $succ(succ(0))$, . . . ("no junk")
- $N \not\models (\{Nat\}, \{0, succ\})$ because ∞ is not obtained by any successor term

Σ^{Nat}-models revisited

- $M \models (\{Nat\}, \{0, succ\})$ because Nat^M consists solely of interpretations of successor terms 0, $succ(0)$, $succ(succ(0))$, . . . ("no junk")
- $N \not\models (\{Nat\}, \{0, succ\})$ because ∞ is not obtained by any successor term
- $T \models (\{Nat\}, \{0, succ\})$ because $*$ is $\emptyset^{\#}(0)$

Σ^{Nat}-models revisited

- $M \models (\{Nat\}, \{0, succ\})$ because Nat^M consists solely of interpretations of successor terms 0, $succ(0)$, $succ(succ(0))$, . . . ("no junk")
- $N \not\models (\{Nat\}, \{0, succ\})$ because ∞ is not obtained by any successor term
- $T \models (\{Nat\}, \{0, succ\})$ because $*$ is $\emptyset^{\#}(0)$
- $K \not\models (\{Nat\}, \{0, succ\})$: $\emptyset^{\#}(suc^n(0))$ is 0 for any n

Σ^{Nat}-models revisited

- $M \models (\{Nat\}, \{0, succ\})$ because Nat^M consists solely of interpretations of successor terms 0, $succ(0)$, $succ(succ(0))$, . . . ("no junk")
- $N \not\models (\{Nat\}, \{0, succ\})$ because ∞ is not obtained by any successor term
- $T \models (\{Nat\}, \{0, succ\})$ because $*$ is $\emptyset^{\#}(0)$
- $K \not\models (\{Nat\}, \{0, succ\})$: $\emptyset^{\#}(suc^n(0))$ is 0 for any n
- $F \not\models (\{Nat\}, \{0, succ\})$ because Nat^F is uncountable (but the set of terms is countable)

Semantics of basic specifications

- Basic specifications denote a signature and a set of sentences

Semantics of basic specifications

- Basic specifications denote a signature and a set of sentences
- Ultimatetly, the semantics of a basic specification consists of that signature together with the class of models satisfying the sentences

Format of semantic judgements

- $\Sigma \vdash \texttt{BASIC-SPEC} \rhd (\Sigma', \Psi)$

Format of semantic judgements

- $\Sigma \vdash \texttt{BASIC-SPEC} \rhd (\Sigma', \Psi)$
- Σ is the local environment of previously-declared symbols

Format of semantic judgements

- $\Sigma \vdash \texttt{BASIC-SPEC} \rhd (\Sigma', \Psi)$
- Σ is the local environment of previously-declared symbols
- Σ' extends Σ with new symbols

Format of semantic judgements

- $\Sigma \vdash \texttt{BASIC-SPEC} \rhd (\Sigma', \Psi)$
- Σ is the local environment of previously-declared symbols
- Σ' extends Σ with new symbols
- Ψ is a set of Σ-sentences

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$

  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$
  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$

- $\Sigma' = (\{Elem, List\},$
  $\{Nil : Elem, Cons \colon Elem * List \longrightarrow List\}, \emptyset, \emptyset)$

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$
  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$
- $\Sigma' = (\{Elem, List\},$
  $\{Nil : Elem, Cons\colon Elem * List \longrightarrow List\}, \emptyset, \emptyset)$
- Ψ consists of axioms stating that

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$

  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$

- $\Sigma' = (\{Elem, List\},$
  $\{Nil : Elem, Cons\colon Elem * List \longrightarrow List\}, \emptyset, \emptyset)$

- Ψ consists of axioms stating that

  - $Cons$ is injective

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$
  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$
- $\Sigma' = (\{Elem, List\},$
  $\{Nil : Elem, Cons\colon Elem * List \longrightarrow List\}, \emptyset, \emptyset)$
- Ψ consists of axioms stating that
  - $Cons$ is injective
  - the ranges of Nil and $Cons$ are disjoint

Example

- $(\{Elem\}, \emptyset, \emptyset, \emptyset)$
  $\vdash$ **free type** $List ::= Nil | Cons(Elem, List) \rhd (\Sigma', \Psi)$
- $\Sigma' = (\{Elem, List\},$
  $\{Nil : Elem, Cons : Elem * List \longrightarrow List\}, \emptyset, \emptyset)$
- Ψ consists of axioms stating that
  - $Cons$ is injective
  - the ranges of Nil and $Cons$ are disjoint
  - $List$ is generated by Nil and $Cons$ (i.e. $(\{List\}, \{Nil : Elem, Cons : Elem * List \longrightarrow List\})$

This means that the models

This means that the models

- may interpret $Elem$ with any set

This means that the models

- may interpret $Elem$ with any set
- must interpret $List$ with lists over that set (up to isomorphism)

This means that the models

- may interpret $Elem$ with any set
- must interpret $List$ with lists over that set (up to isomorphism)
- i.e. terms Nil, $Cons(e, Nil)$ with $e \in Elem^M$, ...

Proof system for the CASL institution

(Absurdity) $$\frac{\mathit{false}}{\varphi}$$

(Tertium non datur) $$\frac{\begin{array}{cc}[\varphi] & [\varphi \Rightarrow \mathit{false}] \\ \vdots & \vdots \\ \psi & \psi \end{array}}{\psi}$$

(⇒-intro) $$\frac{\begin{array}{c}[\varphi] \\ \vdots \\ \psi \end{array}}{\varphi \Rightarrow \psi}$$

(⇒-elim) $$\frac{\begin{array}{c}\varphi \\ \varphi \Rightarrow \psi \end{array}}{\psi}$$

(Reflexivity) $\dfrac{}{x_s \stackrel{e}{=} x_s}$ if x_s is a variable

(∀-elim) $\dfrac{\forall x : s \,.\, \varphi}{\varphi}$

(∀-intro) $\dfrac{\varphi}{\forall x : s \,.\, \varphi}$ where x_s occurs freely only in local assump.

(Congruence) $\dfrac{\varphi}{(\bigwedge_{x_s \in FV(\varphi)} x_s \stackrel{e}{=} \nu(x_s)) \Rightarrow \varphi[\nu]}$ if $\varphi[\nu]$ defined

(Totality) $\dfrac{}{def\, f(f_{w,s}(x_{s_1}, \ldots, x_{s_n}))}$ if $w = s_1 \ldots s_n$, $f \in TF_{w,s}$

(Substitution) $\dfrac{\varphi}{(\bigwedge_{x_s \in FV(\varphi)} def(\nu(x_s))) \Rightarrow \varphi[\nu]}$

if $\varphi[\nu]$ defined and $FV(\varphi)$ occur freely only in local assumpt.

(Function Strictness) $\dfrac{t_1 \stackrel{e}{=} t_2}{def(t)}$ t some subterm of t_1 or t_2

(Predicate Strictness) $\dfrac{p_w(t_1, \ldots, t_n)}{def(t_i)}$ $i \in \{1, \ldots, n\}$

(Induction)
$$\frac{(S', F') \qquad \varphi_1 \wedge \cdots \wedge \varphi_k}{\bigwedge_{s \in S'} \forall x : s \,.\, \Psi_s(x)}$$

$F' = \{f_1 \colon s^1_1 \ldots s^1_{m_1} \longrightarrow s^1; \; \ldots; \; f_k \colon s^k_1 \ldots s^k_{m_k} \longrightarrow s^k\}$,

Ψ_s is a formula with one free variable of sort s, for $s \in S'$,

$$\varphi_j = \forall x_1 : s^j_1, \ldots, x_{m_j} : s^j_{m_j} \,.\, \Big(def\, f_j(x_1, \ldots, x_{m_j})) \wedge \bigwedge_{i \in \{1,\ldots,m_j\};\, s^j_i \in S'} \Psi_{s^j_i}(x_i) \Big) \Rightarrow \Psi_{s_j}\left(f_j(x_1, \ldots, x_{m_j})\right)$$

(List-Induction)

$$\frac{\begin{array}{c}(\{List\}, \{nil : List, cons\colon Elem * List \longrightarrow List\}) \\ \Psi(nil) \wedge \forall e : Elem;\ L : List\,.\ \Psi(L) \Rightarrow \Psi(cons(e, L))\end{array}}{\forall x : List\,.\ \Psi(x)}$$

Ψ is a formula with one free variable of sort $List$

Start induction proof

(Sortgen-intro) $\dfrac{\varphi_1 \wedge \cdots \wedge \varphi_k \Rightarrow \bigwedge_{s \in S'} \forall x : s \,.\, p_s(x)}{(S', F')}$

$F' = \{f_1\colon s_1^1 \ldots s_{m_1}^1 \longrightarrow s^1;\ \ldots;\ f_k\colon s_1^k \ldots s_{m_k}^k \longrightarrow s^k\}$,
the predicates $p_s : s$ $(s \in S')$ occur only in local assumpt.,

$$\varphi_j = \forall x_1 : s_1^j, \ldots, x_{m_j} : s_{m_j}^j \,. \left(def\, f_j(x_1, \ldots, x_{m_j})) \wedge \bigwedge_{i \in \{1,\ldots,m_j\};\ s_i^j \in S'} p_{s_i^j}(x_i) \right) \Rightarrow p_{s_j}\left(f_j(x_1, \ldots, x_{m_j})\right)$$

List-Sortgen-intro

$$\frac{\begin{array}{c} p(nil) \wedge (\forall e : Elem;\ L : List \,.\, p(L) \Rightarrow p(cons(e, L))) \\ \Rightarrow \forall x : List \,.\, p(x) \end{array}}{(\{List\}, \{nil : List, cons \colon Elem * List \longrightarrow List\})}$$

the predicate $p : List$ occurs only in local assumptions

Exercise

- Prove that your favourite sorting algorithm actually computes a permutation, using induction on lists

Course on values-Induction

(List-Induction)

$$\frac{\begin{array}{c}(\{List\}, \{nil : List, cons\colon Elem * List \longrightarrow List\}) \\ \Psi(nil) \wedge \forall e : Elem;\ L : List\,. \\ (\forall L1\,.\ \#L1 < \#L \Rightarrow \Psi(L1)) \Rightarrow \Psi(cons(e, L))\end{array}}{\forall x : List\,.\ \Psi(x)}$$

Ψ is a formula with one free variable of sort $List$

Soundness and Completeness

Let Ψ be a set of Σ-sentences, φ be a Σ-sentence.

- $\Psi \vdash_\Sigma \varphi$ if φ can derived from Ψ using the calculus rules
- $\Psi \models_\Sigma \varphi$ if for every Σ-model, $M \models_\Sigma \Psi$ implies $M \models \varphi$
- The calculus is sound: $\Psi \vdash_\Sigma \varphi$ implies $\Psi \models_\Sigma \varphi$
- The calculus is complete: $\Psi \models_\Sigma \varphi$ implies $\Psi \vdash_\Sigma \varphi$ only if sort generation constraints are excluded

CASL subsorted signatures

A subsorted signature $\Sigma = (S, TF, PF, P, \leq)$ consists of

CASL subsorted signatures

A subsorted signature $\Sigma = (S, TF, PF, P, \leq)$ consists of

- a many-sorted signature (S, TF, PF, P)

CASL subsorted signatures

A subsorted signature $\Sigma = (S, TF, PF, P, \leq)$ consists of

- a many-sorted signature (S, TF, PF, P)
- a pre-oder (reflexive transitive relation) $\leq$ on S

CASL subsorted signatures

A subsorted signature $\Sigma = (S, TF, PF, P, \leq)$ consists of

- a many-sorted signature (S, TF, PF, P)
- a pre-oder (reflexive transitive relation) $\leq$ on S

Signature morphisms are many-sorted signature morphisms preserving the pre-order and the overloading relations

The overloading relations

$f_{w',s'} \sim_F f_{w'',s''}$ iff

The overloading relations

$f_{w',s'} \sim_F f_{w'',s''}$ iff

- there exists w with $w \leq w'$ and $w \leq w''$, and

The overloading relations

$f_{w',s'} \sim_F f_{w'',s''}$ iff

- there exists w with $w \leq w'$ and $w \leq w''$, and
- there exists s with $s' \leq s$ and $s'' \leq s$

The overloading relations

$f_{w',s'} \sim_F f_{w'',s''}$ iff

- there exists w with $w \leq w'$ and $w \leq w''$, and
- there exists s with $s' \leq s$ and $s'' \leq s$

Operation symbols that are in the overloading relation have to be interpreted in the "same" way.

The overloading relations

$f_{w',s'} \sim_F f_{w'',s''}$ iff

- there exists w with $w \leq w'$ and $w \leq w''$, and
- there exists s with $s' \leq s$ and $s'' \leq s$

Operation symbols that are in the overloading relation have to be interpreted in the "same" way.
Similarly for predicate symbols.

Translation from subsorted to many-sorted signatures

Construct many-sorted $\Sigma^{\#}$ out of subsorted Σ as follows:

- Add injections $emb\colon s \longrightarrow s'$ for $s \leq s'$

Translation from subsorted to many-sorted signatures

Construct many-sorted $\Sigma^{\#}$ out of subsorted Σ as follows:

- Add injections $emb \colon s \longrightarrow s'$ for $s \leq s'$
- Add partial projections $proj \colon s' \longrightarrow ?s$ for $s \leq s'$

Translation from subsorted to many-sorted signatures

Construct many-sorted $\Sigma^{\#}$ out of subsorted Σ as follows:

- Add injections $emb\colon s \longrightarrow s'$ for $s \leq s'$
- Add partial projections $proj\colon s' \longrightarrow ?s$ for $s \leq s'$
- Add membership predicates $elem^s : s'$ for $s \leq s'$

Translation from subsorted to many-sorted signatures

Construct many-sorted $\Sigma^{\#}$ out of subsorted Σ as follows:

- Add injections $emb\colon s \longrightarrow s'$ for $s \leq s'$
- Add partial projections $proj\colon s' \longrightarrow ?s$ for $s \leq s'$
- Add membership predicates $elem^{s} : s'$ for $s \leq s'$

$\Sigma^{\#}$ is complemented by a set of axioms J, stating injectivity of embeddings and various compatibility conditions (including preservation of overloading)

Subsorted models and sentences

- are just many-sorted $\Sigma^{\#}$-models and -sentences
- satisfaction is just many-sorted satisfaction