Integrating VSE's Refinement in HETS

Mihai Codescu

FAU Erlangen-Nürnberg

November 6th, 2012

Outline

- Heterogeneous specifications:
  - Motivation
  - CASL and The Heterogenous Tool Set Hets
  - Mathematical foundations
- Integration of VSE in Hets
  - Refinement in VSE
  - Institution of dynamic logic
  - VSE's refinement as a comorphism
  - Tool demo: implementing natural numbers as lists of bits

Formal Specification and Verification of Software

- formal methods are a scientific approach to software engineering, aiming towards
  - a clear and precise description (*specification*)
  - a proof of correct behavior (*verification*)

  of a software system
- while difficult, formal verification is employed in areas
  - where *safety* and *security* are critical: e.g. medical systems, aicraft systems
  - with high cost of failure: e.g. hardware industry
- *algebraic specifications*:
  - programs are modelled as models of some logical system
  - CASL is a de-facto standard language for specification of functional requirements
  - HETS provides tool support for *heterogeneous* specifications

Formal Specification and Verification of Software

- formal methods are a scientific approach to software engineering, aiming towards
  - a clear and precise description (*specification*)
  - a proof of correct behavior (*verification*)

  of a software system
- while difficult, formal verification is employed in areas
  - where *safety* and *security* are critical: e.g. medical systems, aicraft systems
  - with high cost of failure: e.g. hardware industry
- *algebraic specifications*:
  - programs are modelled as models of some logical system
  - CASL is a de-facto standard language for specification of functional requirements
  - HETS provides tool support for *heterogeneous* specifications

Formal Specification and Verification of Software

- formal methods are a scientific approach to software engineering, aiming towards
  - a clear and precise description (*specification*)
  - a proof of correct behavior (*verification*)

  of a software system
- while difficult, formal verification is employed in areas
  - where *safety* and *security* are critical: e.g. medical systems, aicraft systems
  - with high cost of failure: e.g. hardware industry
- *algebraic specifications*:
  - programs are modelled as models of some logical system
  - CASL is a de-facto standard language for specification of functional requirements
  - HETS provides tool support for *heterogeneous* specifications

Heterogeneous Specifications

- allow the user to flexibly choose one of the multitude of logical systems at hand according to his knowledge and to the current problem
- are particularly used in specification of complex systems:
  - change of formalism between different levels of developement
  - viewpoint specification: different aspects of the same component of a logical system are specified in different formalisms
- various logic-specific tools can be employed in the verification of the system

HETS

- parsing, static analysis and proof management tool for heterogeneous multi-logic specification
- sound integration of heterogeneity, using institutions
- flexible selection of tool-supported sublanguages suitable for subproblems
- systematic connection of new formalisms to tools via translations
- easy plug-in of new formalisms and translations

Logics Supported in HETS

- **general-purpose logics:**
  Propositional, QBF, SoftFOL, CASL, HasCASL, HOL-Light, FPL
- **logical frameworks:**
  Isabelle, LF, DFOL, Framework
- **ontologies and constraint languages:**
  CASL-DL, OWL2, CommonLogic, RelScheme, ConstraintCASL
- **logics of reactive systems:**
  CspCASL, CoCASL, ModalCASL, ExtModal, Maude
- **programming languages:**
  Haskell, VSE
- **logics of specific tools:**
  Reduce, DMU (CATIA), Adl, EnCL, FreeCAD

HetCASL

specification libraries
architectural refinements
structured specifications
Grothendieck institution

- the graph of logics is a parameter of the language
- syntax for specifying the current logic and for translations between logics
- model-theoretic semantics

HETS supports other logic-specific (e.g. Maude, Twelf, Common Logic, Haskell) or even heterogeneous (DOL) module systems.

Heterogeneous Development Graphs

Heterogeneous structured specifications are mapped into heterogeneous development graphs [Mossakowski/Autexier/Hutter 2001]:

- nodes correspond to individual specification modules
- definition links correspond to imports of modules
- theorem links express proof obligations

Development graphs

- are a tool for management and reuse of proofs
- come with a sound and complete (up to an oracle for conservative extensions) proof calculus:
  - decompose global theorem links semi-automatically into local ones
  - choose logic specific provers for local proof goals

Logics as Institutions

Institutions are a model-theoretical formalization of logical systems [Goguen/Burstall 1984]
An *institution* consists of:

- a category **Sign** of *signatures*;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a *satisfaction relation* $\models_\Sigma \ \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad [\textit{Satisfaction condition}]$$

Logics as Institutions

Institutions are a model-theoretical formalization of logical systems [Goguen/Burstall 1984]
An *institution* consists of:

- a category **Sign** of *signatures*;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a *satisfaction relation* $\models_\Sigma \; \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad \textit{[Satisfaction condition]}$$

Logics as Institutions

Institutions are a model-theoretical formalization of logical systems [Goguen/Burstall 1984]
An *institution* consists of:

- a category **Sign** of *signatures*;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a *satisfaction relation* $\models_\Sigma \ \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad [\textit{Satisfaction condition}]$$

Logics as Institutions

Institutions are a model-theoretical formalization of logical systems [Goguen/Burstall 1984]
An *institution* consists of:

- a category **Sign** of *signatures*;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a *satisfaction relation* $\models_\Sigma \;\subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$
  such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad [\textit{Satisfaction condition}]$$

Logics as Institutions

Institutions are a model-theoretical formalization of logical systems [Goguen/Burstall 1984]
An *institution* consists of:

- a category **Sign** of *signatures*;
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$, giving a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* for each signature $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Sen}(\sigma)(\varphi)$ is written $\sigma(\varphi)$;
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \to \mathbf{Cat}$, giving a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* for each $\Sigma \in |\mathbf{Sign}|$. Notation: $\mathbf{Mod}(\sigma)(M')$ is written $M'|_\sigma$;
- for each $\Sigma \in |\mathbf{Sign}|$, a *satisfaction relation* $\models_\Sigma \ \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$

  such that for any $\sigma\colon \Sigma \to \Sigma'$, $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_{\Sigma'} \sigma(\varphi) \iff M'|_\sigma \models_\Sigma \varphi \qquad [\textit{Satisfaction condition}]$$

First-order logic: Syntax

- A signature Σ consists of:
  - a set of sorts S,
  - family $F_{w,s}$ of sets of function symbols indexed by arity $w \in S^*$ and result sort $s \in S$,
  - family P_w of sets of predicate symbols with arity $w \in S^*$.
- terms $T_\Sigma(X)$ with variables from $(X_s)_{s \in S}$:
  - $x \in X_s \implies x \in T_\Sigma(X)_s$,
  - $f(t_1, \ldots, t_n) \in T_\Sigma(X)_s$, for each $f \in F_{w,s}$ and $t_i \in T_\Sigma(X)_{w_i}$.
- atomic sentences:
  - $p(t_1, \ldots, t_n)$,
  - $t = t'$.
- sentences:
  - quantification and usual Boolean connectives on top of atomic sentences

First-order logic: Syntax

- A signature Σ consists of:
  - a set of sorts S,
  - family $F_{w,s}$ of sets of function symbols indexed by arity $w \in S^*$ and result sort $s \in S$,
  - family P_w of sets of predicate symbols with arity $w \in S^*$.
- terms $T_\Sigma(X)$ with variables from $(X_s)_{s \in S}$:
  - $x \in X_s \implies x \in T_\Sigma(X)_s$,
  - $f(t_1, \ldots, t_n) \in T_\Sigma(X)_s$, for each $f \in F_{w,s}$ and $t_i \in T_\Sigma(X)_{w_i}$.
- atomic sentences:
  - $p(t_1, \ldots, t_n)$,
  - $t = t'$.
- sentences:
  - quantification and usual Boolean connectives on top of atomic sentences

First-order logic: Syntax

- A signature Σ consists of:
  - a set of sorts S,
  - family $F_{w,s}$ of sets of function symbols indexed by arity $w \in S^*$ and result sort $s \in S$,
  - family P_w of sets of predicate symbols with arity $w \in S^*$.
- terms $T_\Sigma(X)$ with variables from $(X_s)_{s \in S}$:
  - $x \in X_s \implies x \in T_\Sigma(X)_s$,
  - $f(t_1, \ldots, t_n) \in T_\Sigma(X)_s$, for each $f \in F_{w,s}$ and $t_i \in T_\Sigma(X)_{w_i}$.
- atomic sentences:
  - $p(t_1, \ldots, t_n)$,
  - $t = t'$.
- sentences:
  - quantification and usual Boolean connectives on top of atomic sentences

First-order logic: Model Theory

- A model M of a signature gives:
  - for each sort s, a non-empty carrier set M_s,
  - for each function symbol $f : w \to s$, a function $M_f : M_w \to M_s$ and
  - for each predicate symbol $p : w$, a relation $M_p \subseteq M_w$.
- For $\Sigma_{Monoid} = (\{univ\}, \{e : univ, \circ : univ \times univ \to univ\})$, the monoid of natural numbers with addition N is given by
  - $N_{univ} = \{0, 1, \ldots\}$
  - $N_\circ = +$
  - $N_e = 0$
- satisfaction is defined inductively, using intepretation of terms:
  - $M_{f(t_1,\ldots,t_n)} = M_f(M_{t_1}, \ldots, M_{t_n})$
  - $M \models t = t'$ iff $M_t = M'_t$
  - $M \models p(t_1, \ldots, t_n)$ iff $M_p(M_{t_1}, \ldots, M_{t_n})$ holds
  - the usual way for Boolean connectives and quantification

First-order logic: Model Theory

- A model M of a signature gives:
  - for each sort s, a non-empty carrier set M_s,
  - for each function symbol $f : w \to s$, a function $M_f : M_w \to M_s$ and
  - for each predicate symbol $p : w$, a relation $M_p \subseteq M_w$.
- For $\Sigma_{Monoid} = (\{univ\}, \{e : univ, \circ : univ \times univ \to univ\})$, the monoid of natural numbers with addition N is given by
  - $N_{univ} = \{0, 1, \ldots\}$
  - $N_\circ = +$
  - $N_e = 0$
- satisfaction is defined inductively, using intepretation of terms:
  - $M_{f(t_1,\ldots,t_n)} = M_f(M_{t_1}, \ldots, M_{t_n})$
  - $M \models t = t'$ iff $M_t = M'_t$
  - $M \models p(t_1, \ldots, t_n)$ iff $M_p(M_{t_1}, \ldots, M_{t_n})$ holds
  - the usual way for Boolean connectives and quantification

First-order logic: Model Theory

- A model M of a signature gives:
  - for each sort s, a non-empty carrier set M_s,
  - for each function symbol $f : w \to s$, a function $M_f : M_w \to M_s$ and
  - for each predicate symbol $p : w$, a relation $M_p \subseteq M_w$.
- For $\Sigma_{Monoid} = (\{univ\}, \{e : univ, \circ : univ \times univ \to univ\})$, the monoid of natural numbers with addition N is given by
  - $N_{univ} = \{0, 1, \ldots\}$
  - $N_\circ = +$
  - $N_e = 0$
- satisfaction is defined inductively, using intepretation of terms:
  - $M_{f(t_1,\ldots,t_n)} = M_f(M_{t_1}, \ldots, M_{t_n})$
  - $M \models t = t'$ iff $M_t = M'_t$
  - $M \models p(t_1, \ldots, t_n)$ iff $M_p(M_{t_1}, \ldots, M_{t_n})$ holds
  - the usual way for Boolean connectives and quantification

Translations as Institution Comorphisms

An institution comorphism [Goguen/Rosu 2000] from I to J is a triple (Φ, α, β) where

- Φ is a functor mapping signatures of I to signatures of J
- α_Σ naturally maps sentences in I over Σ to sentences in J over $\Phi(\Sigma)$
- β_Σ naturally reduces $\Phi(\Sigma)$-models in J to Σ-models in I

such that *truth is invariant under translation*:

$$\beta_\Sigma(M) \models^I_\Sigma e \iff M \models^J_{\Phi(\Sigma)} \alpha_\Sigma(e)$$

$$\begin{array}{ccc}
I & & J \\
\Sigma & \xmapsto{\quad\Phi\quad} & \Phi(\Sigma) \\
Sen^I(\Sigma) & \xrightarrow{\quad\alpha_\Sigma\quad} & Sen^J(\Phi(\Sigma)) \\
Mod^I(\Sigma) & \xleftarrow{\quad\beta_\Sigma\quad} & Mod^J(\Phi(\Sigma))
\end{array}$$

Grothendieck Institutions

Given an indexed coinstitution $\mathcal{I}\colon Ind^{op} \longrightarrow \mathbf{CoIns}$, we define the *Grothendieck institution* [Diaconescu 2002, Mossakowski 2002] $\mathcal{I}^{\#}$ as follows:

$$
\begin{tikzcd}
(i, \Sigma_1) \arrow[r, "d:j\to i"] & (j, \Phi^d(\Sigma_1)) \arrow[r, "\sigma"] & (j, \Sigma_2) \\
\mathbf{Sen}^i(\Sigma_1) \arrow[r, "\alpha^d_{\Sigma_1}"] & \mathbf{Sen}^j(\Phi^d(\Sigma_1)) \arrow[r, "\mathbf{Sen}^j(\sigma)"] & \mathbf{Sen}^j(\Sigma_2) \\
\models^i_{\Sigma_1} & & \models^j_{\Sigma_2} \\
\mathbf{Mod}^i(\Sigma_1) & \mathbf{Mod}^j(\Phi^d(\Sigma_1)) \arrow[l, "\beta_{\Sigma_1}"'] & \mathbf{Mod}^j(\Sigma_2) \arrow[l, "\mathbf{Mod}^j(\sigma)"']
\end{tikzcd}
$$

Verification Support Environment (VSE)

- industrial-strength methodology for specification and verification of large scale software systems, based on a refinement process
- provides an interactive deductive component, based on a Gentzen style natural deduction calculus for dynamic logic and supports automatic code generation
- successfully used [HutterEtAl. 2000] in projects such as the control system of a heavy robot facility, a formal security policy model conforming to the German signature law and protocols for chip card based biometric identification.

Institution of Dynamic Logic (DynL)

- signatures: first-order signatures + procedure symbols
- models: first-order structures + procedures as relations
- sentences:
  - the usual dynamic logic formulas, with imperative programs as modalities
  - procedure definitions assigning programs to procedure symbols
  - sort generation constraints with restrictions
- satisfaction:
  - Kripke-like satisfaction for dynamic logic formulas
  - replacing a procedure call with its body should not change the result of a program + some minimality condition for recursive functions
  - the elements of the sort satisfying the restriction are generated by the constructors

Institution of Dynamic Logic - signatures

- A signature $\Sigma = (S, F, P, PR)$ consists of:
  - a FOL signature (S, F, P)
  - a family $PR_{w,v}$ of procedure symbols with input arguments $w \in S^*$ and output arguments $v \in S^*$
  - some procedure symbols in $PR_{w,s}$, where $s \in S$, are marked as functional, denoted FP
  - a subsignature for Boolean values
- A signature morphism maps corresponding symbols such that functional symbols are mapped to functional symbols and the map between procedure symbols is injective. Booleans are mapped identically by signature morphisms.

Institution of Dynamic Logic - terms and programs

- for a signature $\Sigma = (S, F, P, PR)$ and a sorted set of variables X, the terms are the usual first-order terms over the signature $(S, F \cup FP, P)$ with variables in X
- for a signature $\Sigma = (S, F, P, PR)$, the set of Σ-programs is the smallest set containing
  - **abort**, **skip**
  - $x := t$
  - **declare** $x : s = t$, **declare** $x : s = t$
  - $\alpha; \beta$
  - **if** Φ **then** α **else** β **fi**
  - **while** Φ **do** α **od**
  - $pr(x_1, \ldots, x_n; y_1, \ldots, y_n)$
  - **return** t

Institution of Dynamic Logic - sentences

For a signature $\Sigma = (S, F, P, PR)$, $\mathbf{Sen}(\Sigma)$ contains:

- dynamic logic formulas:
  - T and F
  - first-order (S, F, P)-formulas
  - $[\alpha]e$, $\langle\alpha\rangle e$, $\neg e$, $e_1 \wedge e_2$ and $\forall x : s \bullet e$
- procedure definitions:

  defprocs
  procedure $pr_1(x_1^1, \ldots, x_{n_1}^1; y_1^1, \ldots, y_{m_1}^1)\alpha_1$
  $\ldots$
  procedure $pr_k(x_1^k, \ldots, x_{n_k}^k; y_1^k, \ldots, y_{m_k}^k)\alpha_k$
  defprocsend
- restricted sort generation constraints:

  generated types
  $s_1 ::= p_1^1(\ldots)|p_2^1(\ldots)|\ldots|p_n^1(\ldots)$ **restricted by** r^1
  $\ldots$
  $s_k ::= p_1^k(\ldots)|p_2^k(\ldots)|\ldots|p_m^k(\ldots)$ **restricted by** r^k

Institution of Dynamic Logic - models and satisfaction

For a signature $\Sigma = (S, F, P, PR)$, a model is a first-order structure such that procedure symbols are interpreted as relations, functional procedures as total functions and Booleans in the standard way.
A model M satisfies:

- each definition of a procedure pr_i if

$$M \models \forall x_1^i, \ldots, x_{n_i}^i, r_1^i, \ldots, r_{m_i}^i :$$
$$(\langle pr_i(x_1^i, \ldots, x_n^i; y_1^i, \ldots, y_m^i) \rangle y_1^i = r_1^i \wedge \cdots \wedge y_{m_i}^i = r_{m_i}^i)$$
$$\Leftrightarrow \langle \alpha \rangle y_1^i = r_1^i \wedge \cdots \wedge y_{m_i}^i = r_{m_i}^i$$

- a RSGC $s ::= p_1(\ldots)|p_2(\ldots)| \ldots |p_n(\ldots)$ **restricted by** r if the subset of M_s on which r terminates is generated by the constructors p_i

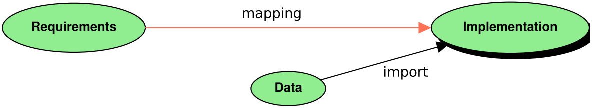
Satisfaction (continued)

Kripke-like semantics for dynamic logic formulas in a model M:

- states are partial functions taking variables to values
- interpretation of a term in a state is defined as expected
- semantics of a program is a predicate on two states, denoted $\_[\![\alpha]\!]^M\_$, e.g.:
  - $q[\![x := \tau]\!]^M r \Leftrightarrow r = q[x : s \leftarrow \tau^{M,q}]$ and $\tau^{M,q}$ is defined, where $s = sort(\tau)$
  - $q[\![\mathbf{if}\,\varepsilon\,\mathbf{then}\,\alpha\,\mathbf{else}\,\beta\,\mathbf{fi}]\!]^M r \Leftrightarrow (q \models \varepsilon$ and $q[\![\alpha]\!]^M r)$ or $(q \models \neg\varepsilon$ and $q[\![\beta]\!]^M r)$
- satisfaction is first defined on a program state r:
  - $M, r \models p(\tau_1, \ldots \tau_n) \Leftrightarrow$ for all $i = 1, \ldots, n$, $\tau_i^{M,r}$ is defined and $M_p(\tau_1^{M,r}, \ldots, \tau_n^{M,r})$
  - $M, r \models [\alpha]e \Leftrightarrow$ for all program states q with $r[\![\alpha]\!]^M q$: $M, q \models e$
- finally $M \models e$ iff $M.r \models e$ for each state r

Refinement in VSE

- first-order specification of requirements
- DynL specification of the implementation:
  - imports first-order specification of data
  - procedure definitions
  - for each sort, designated restriction and observational congruence
- a mapping assigns to each symbol the procedure symbol implementing it
- VSE proves correctness semi-automatically



VSE's Refinement in HETS

The refinement notion of VSE is represented as a comorphism from CASL to DynL such that:

- signatures:
  - for each sort we introduce procedure symbols for equality and restriction formula and axioms for their expected behavior
  - for each function/predicate symbol we introduce new procedure symbols, loosely specified
- translation of first-order sentences is based on translation of terms into programs implementing the representation of the term
- dynamic-logic models are reduced by performing the submodel-quotient construction.

CASL2VSERefine - syntax

- for each sort s:
  - sort s
  - $eq_s \in PR_{[s,s],[Bool]}$
  - $r_s \in PR_{[s],[]}$

  and sentences:
  - $\langle r_s(x)\rangle T \ \wedge \langle r_s(y)\rangle T \Rightarrow \langle eq_s(x,y;e)\rangle T$
  - $\langle r_s(x)T\rangle \Rightarrow \langle eq_s(x,x;e)\rangle e = T$
  - $\langle r_s(x)\rangle T \wedge \langle r_s(y)\rangle T \wedge \langle eq_s(x,y;e)\rangle e = T \Rightarrow \langle eq_s(y,x;e)\rangle e = T$
  - $\langle r_s(x)\rangle T \ \wedge \ \langle r_s(y)\rangle T \ \wedge \ \langle r_s(z)\rangle T \ \wedge \ \langle eq_s(x,y;e)\rangle e = T \ \wedge \ \langle eq_s(y,z;e)\rangle e = T \ \Rightarrow \langle eq_s(x,z;e)\rangle e = T$
- for each $f \in F_{s \to t}$, $f \in PR_{[s],[t]}$ and sentences
  - $\langle r_s(x)\rangle T \wedge \langle r_s(y)\rangle T \wedge \langle eq_s(x,y;e)\rangle e = T \Rightarrow \langle y1 := f(x)\rangle \langle y2 := f(y)\rangle \langle eq_t(y1,y2;e)\rangle e = T$
  - $\langle r_s(x)\rangle T \implies \langle f(x;y)\rangle \langle r_t(y)\rangle T$
- for each $p \in P_s$, $p \in PR_{[s],[Bool]}$ and sentences
  - $\langle r_s(x)\rangle T \wedge \langle r_s(y)\rangle T \wedge \langle eq_s(x,y;e)\rangle e = T \Rightarrow \langle p(x;r1)\rangle \langle p(y;r2)\rangle r1 = r2$
  - $\langle r_s(x)\rangle T \Rightarrow \langle p(x;e)\rangle T$

CASL2VSERefine - models

Given a CASL signature $\Sigma = (S, F, P)$ and a model M' of $\Phi(\Sigma) = ((S, \emptyset, \emptyset, PR), E)$, let $M = \beta_\Sigma(M')$:

- $M_s = M_{r_s}/_\equiv$ where:
  - M_{r_s} is the subset of M'_s for which r_s holds
  - $a \equiv b$ is equivalent to $M', t \models \langle eq_s(x_1, x_2; y)\rangle y = true$ whenever $t(x_1) = a$ and $t(x_2) = b$
- for each function symbol f, $M_f(a_1, \ldots, a_n) = b$ iff $M', t \models \langle f(x_1, \ldots, x_n; y)\rangle y = z$ when $t(x_i) = a_i$ and $t(z) = b$.
- for each predicate symbol p, $M_p(a_1, \ldots, a_n)$ holds iff $M', t \models \langle p(x_1, \ldots, x_n; y)\rangle y = true$.

CASL2VSERefine - sentences

- terms are translated into programs that compute their representation:
  - $x \mapsto x := x$
  - $f(t_1, \ldots, t_n) \mapsto \alpha_1; \ldots; \alpha_n; a := f(y_1, \ldots, y_n)$
- sentences are translated inductively:
  - $t_1 = t_2 \mapsto \langle \alpha_1; \alpha_2; eq_s(y_1, y_2; y) \rangle y = T$
  - $\forall x : s.e \mapsto \forall x : s.\langle r_s(x) \rangle true \Rightarrow \alpha(e)$

Natural Numbers as Lists of Bits

view BINARY :
SIMPNATS **to** NATS_IMPL =
logic $\rightarrow$ *CASL2VSERefine*,
$nats \mapsto bin$,
$gn\_restr\_nats \mapsto nlz$,
$gn\_eq\_nats \mapsto eq$,
$gn\_zero\_n \mapsto i\_zero$,
$gn\_succ\_n \mapsto i\_succ$,
$gn\_prdc\_n \mapsto i\_prdc$,
$gn\_add\_n \mapsto i\_add$