Data Refinement:
model-oriented proof methods and their comparison

Willem-Paul de Roever
University of Kiel, Germany
Overview

- Refinement
- Data refinement
- Simulation
- Equivalence between assertional and relational characterizations of downward simulation
- Sound and relatively complete proof system for a minimal Hoare logic
- Theorems: Reynolds’ method, VDM reduced to downward simulation for total correctness
Questions answered in this talk

- What is a (data) refinement step?
- How to find and prove such a step?
- How to judge the solutions given by others?
Refinement (1)

Given a pair of programs called concrete and abstract, the concrete program refines the abstract program correctly whenever the use of the concrete program does not lead to an observation which is not also an observation of the abstract program.  

[Gardiner & Morgan, 1993]

So, what is observable?
So, what is observable?

In our setting of sequential, imperative programs, only the binary relation between initial and final states is considered observable.

Given a class $Prog$ of programs and a function

$$\mathcal{P}_[.] : Prog \rightarrow 2^{\Sigma \times \Sigma}$$

that maps each program to its initial/final state relation, program $S \in Prog$ refines $T \in Prog$ is defined by

$$\mathcal{P}[S] \subseteq \mathcal{P}[T],$$

abbreviated to

$$S \subseteq T.$$
Example 1 Let $S_1$ and $S_2$ denote statements not involving variables $s$ and $l$. Compare the following two programs; they refine each other.

begin
\begin{align*}
\text{var } s & : \text{finset of } \mathbb{N} ; s := \emptyset; \\
S_1; \\
s & := s \cup \{x\}; \\
S_2; \\
y & := \text{a member of } s
\end{align*}
end

begin
\begin{align*}
\text{var } l & : \mathbb{N}^* ; l := \text{nil}; \\
S_1; \\
l & := \text{append}(l, x); \\
S_2; \\
y & := \text{first}(l)
\end{align*}
end

This refinement step comprises of replacing the variable $s$ (ranging over finite subsets of the natural numbers) and operations on it by the sequence-valued variable $l$ and corresponding operations.
Initial/final state behaviour of $S_1$ and $S_2$ in terms of value-transformations of $x$, $y$ are global w.r.t. $S_1$ and $S_2$: $x$ and $y$ are called normal variables.

In contrast $s$, $t$ are data-representation variables. Their values are only visible inside $S_1$ and $S_2$, because these variables vary according to the abstraction level.

Representation variables are not observable outside a program.
How to formalize the interesting part of two programs such as those in the example from the refinement point of view?

**Definition 1 [data type]** Given a finite set of variables $\bar{x}$, called normal variables, another (disjoint) finite set of variables $\bar{a}$, called representation variables, and a finite index set $J$, define state spaces $\Sigma$ and $\Sigma^A$ by $\Sigma \equiv [\bar{x} \rightarrow \mathbb{V}]$ and $\Sigma^A \equiv [\bar{x} \cup \bar{a} \rightarrow \mathbb{V}]$. Let $A_j \subseteq \Sigma^A \times \Sigma^A$ for $j \in J$. Let initialization $AI \subseteq \Sigma \times \Sigma^A$, and finalization $AF \subseteq \Sigma^A \times \Sigma$. Then we call

$$\mathcal{A} = (AI, (A_j)_{j \in J}, AF)$$

a data type.

Note relational characterization of $\mathcal{A}$: $A_j \subseteq \Sigma^A \times \Sigma^A$. 
A program skeleton maps each data type to a relation constructed from
the operations $A_j$ and operations on the normal variables using
sequential composition, non-deterministic choice and recursion.

Example 2 $P(A) = A_1 ; A_2 \cup A_3$ and $P(C) = C_1 ; C_2 \cup C_3$.

Obviously, there are infinitely many program skeletons (unless $J = \emptyset$).
Compare two levels of abstraction:

\[ \mathcal{A} \quad \text{of data type } \mathcal{A} \]
\[ \mathcal{C} \quad \text{of data type } \mathcal{C} \]

with \( \mathcal{A} \) and \( \mathcal{C} \) compatible (index sets \( J \) plus set \( \bar{x} \) of normal variables the same).

\( \mathcal{C} \) should refine/implement \( \mathcal{A} \).

As mentioned before, the data type representation variables (e.g., \( s \) and \( l \)) themselves are NOT observable.  \( \Rightarrow \)

When defining that \( \mathcal{C} \) refines \( \mathcal{A} \), the particular way a data type representation is defined should, therefore, not be observable:
When defining that $C$ refines $A$, the particular way a data type representation is defined should, therefore, not be observable:

\[ CI ; \ldots ; CF \subseteq AI ; \ldots ; AF \]

$CI$, $CF$ hide the transformation of $\bar{c}$ by $\{C_j\}_{j \in J}$

$AI$, $AF$ hide the transformation of $\bar{a}$ by $\{A_j\}_{j \in J}$

Moreover, the fact that one data type refines another should hold for all program skeletons using those data types:

\[ CI ; P(C) ; CF \subseteq AI ; P(A) ; AF, \]

for all program skeletons $P$ concerned.  \Rightarrow

This involves proving infinitely many proof obligations.
Definition 2  Data type $C = (CI, (C_j)_{j \in J}, CF)$ refines data type $A = (AI, (A_j)_{j \in J}, AF)$ iff, for all program skeletons $P$:

$$CI ; P(C) ; CF \subseteq AI ; P(A) ; AF$$

Technical note: $C$ uses $\bar{c}$ (disjoint from $\bar{x}$ and $\bar{a}$) and $\Sigma^C = [\bar{x} \cup \bar{c} \rightarrow \mathbb{V}]$ instead of $\bar{a}$ and $\Sigma^A$. Moreover, $C$ and $A$ use the same index set $J$. I.e., $C$ and $A$ are compatible.

Hence, in order to prove data refinement, one has to prove infinitely many proof obligations.
Why simulation?

Instead of proving \textit{infinitely many} proof obligations such as

directly, one would like to use \textit{induction}. This requires invention of a relationship $\rho$ between \textit{abstract} and \textit{concrete} level representation.

To focus on (the finite number of) base cases

one has to guarantee that \textit{induction steps are for free}. 
Consider a relation \( \rho \subseteq \Sigma^A \times \Sigma^C \) between abstract and concrete states. Then there are essentially \( \text{four} \) ways in which \( \text{weak commutativity} \) of the diagram can be defined, possibly using inverses of \( \rho \).
Local conditions for simulation (2)

The induction step for sequential composition is free only for the first two, called downward and upward simulation, resp.

Technical note: The conditions for initialization and finalization are obtained by “identifying” either of the RHS/LHS pairs of corners in the diagrams above.
Both downward and upward simulation are *sound* techniques for proving data refinement. The induction steps for sequential composition look as follows.
Proofs
Incompleteness of downward simulation (1)

\[ P(A) : \]

\[ P(C') : \]

\[ \sigma \quad A_1 \quad a_0 \quad a_1 \quad A_2 \quad a_3 \quad A F \quad \tau \]

\[ C I \quad C F \]

\[ c_0 \quad C_1 \quad c_1 \quad C_2 \quad c_4 \]

\[ \rho \quad a_2 \]

\[ \tau' \]
Incompleteness of downward simulation (2)

Assume $\rho$ is a downward simulation relation between $(AI, (A_j)_{j \in \{1,2\}}, AF)$ and $(CI, (C_j)_{j \in \{1,2\}}, CF)$ where the relations in question are those depicted above.

1. $CI \subseteq AI$; $\rho$, thus, $(a_0, c_0) \in \rho$.

2. $\rho; C_1 \subseteq A_1$; $\rho$, thus, one of $(a_1, c_1)$ and $(a_2, c_1)$ is in $\rho$.
   W.l.o.g. assume that $(a_1, c_1) \in \rho$.

3. $\rho; C_2 \subseteq A_2$; $\rho$, thus, $(a_3, c_4) \in \rho$.

4. $\rho; CF \subseteq AF$, which implies, that $(a_3, \tau') \in AF$,
   however, $CF$ is only $\{(c_3, \tau), (c_4, \tau')\}$! **Contradiction!**
The combination of downward and upward simulation is complete for proving refinement between data types.

**Theorem 1 [HHS]** If $C$ refines $A$ then there exist

- an intermediate data type $B$,
- a downward simulation relation $\rho$ between $B$ and $C$, and
- an upward simulation relation $\alpha$ between $B$ and $A$. 

![Diagram showing relationships between $A_I$, $A_j$, $A_F$, $C_I$, $C_j$, and $C_F$ with simulation relations $\rho$ and $\alpha$.]
What’s out there?

Numerous (formal) methods exist for writing specifications and refining those to implementations:

- VDM (Raise, Z, B)
- Reynolds’ method
- Refinement Calculi of Back & von Wright, Gardiner & Morgan, Morris
- Hehner’s method
- Abadi & Lamport’s refinement mappings
- Lynch’s possibilities mappings

Major development technique: stepwise refinement

All these methods are proved to be related in the Data Refinement book by Kai Engelhardt and me.
Key problem

- The soundness and completeness results of [HHS86] reduce the task of proving data refinement to:

  \[
  \begin{array}{c}
  A \\
  \rho \\
  \cup \\
  C \\
  \rho \\
  \end{array}
  \begin{array}{c}
  \downarrow \rho \\
  \uparrow \rho \\
  \{ \text{downwards} \} \quad \{ \text{upwards} \} \quad \text{simulates}
  \end{array}
  \]

  So we have to prove inclusion between relations

  \((\rho ; C \subseteq A ; \rho \text{ and } C ; \rho^{-1} \subseteq \rho^{-1} ; A)\).

  I.e., we have a relational characterization of simulation.

- This relational characterization we want to compare with methods which use assertional characterizations of operations and simulation (Hoare logics, VDM, Reynolds, refinement calculi).

- Key problem: How to relate these two characterizations?
Assertional vs. relational characterizations of an operation

Assertional methods characterize operations by first-order logic assertions called pre- and postconditions. Questions:

1: Given an assertional characterization of an operation, which relation is determined by it?

2: Given a relational characterization of an operation: can this operation be expressed using pre- and postconditions?

Ad 2: Solved affirmatively in [Zwiers '89, LNCS 321] on the basis of recursion theory.

Ad 1: Solved using Galois connections as developed below.
Use Hoare formulae $\{\phi\} \quad S \quad \{\psi\}$ to specify operations, meaning:

$\{\phi\} S \{\psi\}$ is valid (holds) iff

- if $\phi$ holds in initial state $\sigma$, and if $S$ terminates for initial state $\sigma$ in final state $\tau$ then $\psi$ holds in $\tau$.

Notation: $\models \{\phi\} S \{\psi\}$ (validity)
Specifying operations by Hoare formulae introduces the need for logical variables \( v \), i.e., variables \( v \) whose values are not changed during program execution:

\[
\{ x = v \} \ x := x + 1 \ \{ x = v + 1 \},
\]

because, otherwise, no single axiom for \( x := x + 1 \).

Leads to introduction of set \( \text{Logvar} \) of logical variables disjoint from \( \text{VAR} \), the set of program variables, and to logical variable states \( \Gamma \overset{\text{def}}{=} \text{Logvar} \rightarrow \text{VAL}, \gamma \in \Gamma \).
Using **logical variable states**, the meaning of $\models \{ \varphi \} S \{ \psi \}$ is:

$$\forall \sigma, \tau \in \Sigma. \forall \gamma \in \Gamma. (\gamma, \sigma) \in C[\varphi] \land (\sigma, \tau) \in P[S] \Rightarrow (\gamma, \tau) \in C[\psi],$$

with the meaning of assertions given by a relation between logical states and program states:

$$C[\varphi] \subseteq \Gamma \times \Sigma$$

and the meaning of operation $S$ as a relation between program states:

$$P[S] \subseteq \Sigma \times \Sigma$$

This implies:

$$\models \{ \varphi \} S \{ \psi \} \iff \models \varphi ; S \subseteq \psi$$

using $r_1 ; r_2 \overset{\text{def}}{=} \{(\sigma, \tau) \mid \exists \theta. (\sigma, \theta) \in r_1 \land (\theta, \tau) \in r_2\}$. 

Second connection

- When operation $op$ is specified by $\{\varphi\} op \{\psi\}$, we interpret $op$ as the maximal relation $r$ satisfying $C[\varphi]; r \subseteq C[\psi]$.

- This max. relation is expressed by specification statement $\varphi \leadsto \psi$: $P[\varphi \leadsto \psi] \overset{\text{def}}{=} \{(\sigma, \tau) \mid \forall \gamma. (\gamma, \sigma) \in C[\varphi] \Rightarrow (\gamma, \tau) \in C[\psi]\}$

- Since $\forall \sigma, \tau. (\gamma, \sigma) \in C[\varphi] \land (\sigma, \tau) \in P[S] \Rightarrow (\gamma, \tau) \in C[\psi]$)
  $\iff \forall \sigma, \tau. (\sigma, \tau) \in P[S] \Rightarrow (\forall \gamma. (\gamma, \sigma) \in C[\varphi] \Rightarrow (\gamma, \tau) \in C[\psi])$
  $\iff P[S] \subseteq P[\varphi \leadsto \psi]$, one obtains

  \[
  \{\varphi\} S \{\psi\} \iff \models S \subseteq \varphi \leadsto \psi.
  \]

- This clarifies why we interpret $op$ as maximal relation: We do not want to restrict any refinement $S$ of $op$ unnecessarily.
Let for \( s \subseteq A \times C \) and \( t \subseteq B \times C \)

\[
[t]s \overset{\text{def}}{=} \{(a,b) \in A \times B \mid \forall c \in C.(b,c) \in t \Rightarrow (a,c) \in s\}
\]

then

\[\models \{\varphi\} S \{\psi\} \iff \models \varphi \subseteq [S]\psi\]

**Proof:** \(\forall \sigma. \forall \gamma. (\gamma, \sigma) \in C[\varphi] \land (\sigma, \tau) \in P[S] \Rightarrow (\gamma, \tau) \in C[\psi]\)

\[\iff \forall \gamma, \sigma. (\gamma, \sigma) \in C[\varphi] \Rightarrow (\forall \tau. (\sigma, \tau) \in P[S] \Rightarrow (\gamma, \tau) \in C[\psi])\]

\[\iff \forall \gamma, \sigma. (\gamma, \sigma) \in C[\varphi] \Rightarrow (\gamma, \sigma) \in C[[S]\psi]\]

\[\iff C[\varphi] \subseteq C[[S]\psi]\]

QED
Express maximal solutions for each of the relations on the LHS of $\varphi ; S \subseteq \psi$ in terms of the remaining two relations.

$S \subseteq \varphi \hookrightarrow \psi \iff \varphi ; S \subseteq \psi \ (\iff \varphi \subseteq [S] \psi)$

It depends on the program semantics chosen whether total or partial correctness is expressed; the equivalence of these inclusions holds in both cases.

This Galois connection is our main technical tool in relating relational to assertional characterizations of operations.
**Problem:** How to characterize the maximal relation $C$ $?$-simulating $\varphi \rightsquigarrow \psi$ under abstraction relation $\rho$ as a specification statement:

\[
\varphi \rightsquigarrow \psi \quad ? \subseteq \rho \quad ? = L \text{ or downwards} / L^{-1} \text{ or upwards}
\]

\[
\rho \quad C \quad \rho
\]

$\Rightarrow$ Once solved, $?$-simulation is characterized, and therefore provable, within Hoare Logic

We solved this problem for both $L$ and $L^{-1}$-simulation and for partial correctness and total correctness relational semantics [de Roever & Engelhardt, MFCS ’96].
**Problem:** How to characterize the maximal relation $C$ downwards simulating $\varphi \rightsquigarrow \psi$ under abstraction relation $\rho$ as a specification statement.

**Solution for $C$:**

(Partial correctness, relational semantics, downward simulation)

$$
\exists \bar{a} \left( \rho \land (\bar{x}, \bar{a}) = (\bar{y}_0, \bar{b}_0) \right) \rightsquigarrow \exists \bar{a} \left( \rho \land \forall \bar{x}_0 \left( \varphi \left[ \frac{(\bar{y}_0, \bar{b}_0)}{(\bar{x}, \bar{a})} \right] \Rightarrow \psi \right) \right)
$$

**NB** For the total correctness solution, add conjunct $\exists \bar{a} \left( \rho \land \exists \bar{x}_0 \left( \varphi \right) \right)$ to the precondition. (This term expresses the domain of convergence of the total correctness solution.) This conjunct is essential in justifying the reduction of Reynolds’ method for data refinement to downwards simulation for total correctness.
Proof sketch of the downward simulation theorem for partial correctness (1)

\[ S \text{ downward simulates } \varphi \rightsquigarrow \psi \text{ w.r.t. } \beta \]

\[ \iff \beta ; S \subseteq (\varphi \rightsquigarrow \psi) ; \beta \]

\[ \iff S \subseteq \beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta \]
Proof sketch of the downward simulation theorem for partial correctness (2)

From the solution

\[ \sum \bar{a}, \bar{x} \ (\varphi \rightsquigarrow \psi) ; \beta \]

provided by the Galois connection \( \beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta \) (which is not a specification statement but a relational term) we construct a specification statement \( \varphi' \rightsquigarrow \psi' \)

with the same meaning for \( S \) by expressing binary relations, "\( \rightsquigarrow \)"“, and ";" syntactically and replacing abstract program states in the upper left corner by concrete logical states.
Expressing binary relations and $\rightsquigarrow$ syntactically (1)

Given first-order logic predicates $\varphi$, $\psi$ with free variables:

\[ fv(\varphi) = \{ \bar{x}, \bar{y} \} \quad \text{notation: } \varphi(\bar{x}; \bar{y}), \quad \bar{x} \cap \bar{y} = \emptyset \]
\[ fv(\psi) = \{ \bar{x}, \bar{z} \} \quad \text{notation: } \psi(\bar{x}; \bar{z}), \quad \bar{x} \cap \bar{z} = \emptyset \]

Then: $C[\varphi] \subseteq \Sigma^{\bar{x}} \times \Sigma^{\bar{y}}$, $C[\psi] \subseteq \Sigma^{\bar{x}} \times \Sigma^{\bar{z}}$, with $\Sigma^{\bar{u}} \overset{\text{def}}{=} [\bar{u} \to \text{Val}]$

and $C[\varphi] \rightsquigarrow C[\psi] = \{ (\sigma, \tau) \mid \forall \theta. (\theta, \sigma) \in C[\varphi] \Rightarrow (\theta, \tau) \in C[\psi] \} \subseteq \Sigma^{\bar{y}} \times \Sigma^{\bar{z}}$

Since $\bar{y}$ and $\bar{z}$ in general not disjoint, indicate syntactically which variables are evaluated in initial state $\sigma$ and which ones in final state $\tau$, for $(\sigma, \tau) \in C[\varphi] \rightsquigarrow C[\psi]$.

Substitute primed versions $v'$ for variables $v$ evaluated in the initial state, with unprimed versions evaluated in the final state.
Expressing binary relations and \(\rightsquigarrow\) syntactically (2)

**Convention:** primed variables \(v'\) evaluated in \(\sigma\) by \(v'(\sigma, \tau) = \sigma(v)\), and unprimed ones \(v\) in \(\tau\) by \(v(\sigma, \tau) = \tau(v)\), and define

\[
\varphi \rightsquigarrow \psi \overset{\text{def}}{=} (\forall \bar{y}. \varphi[\bar{y}' / \bar{y}] \rightarrow \psi)(\bar{y}; \bar{z}),
\]

possibly renaming \(\bar{x}\) in case \(\bar{x} \cap \bar{y}' \neq \emptyset\).

**Example 3** \((x = x_0 \rightsquigarrow x = x_0 + 1) = \forall x_0. (x' = x_0 \rightarrow x = x_0 + 1)\) characterizes \(x := x + 1\).

**Theorem:** \(C[\varphi \rightsquigarrow \psi] = C[\varphi] \rightsquigarrow C[\psi]\)
Expressing “;” syntactically

Given first-order logic predicates $\varphi(\bar{x}; \bar{y})$ and $\psi(\bar{y}; \bar{z})$, “;” is usually defined by

$$(\exists \bar{y}. \varphi(\bar{x}; \bar{y}) \land \psi(\bar{y}; \bar{z})) (\bar{x}; \bar{z})$$

However this does NOT cater for our primed variable convention:

So one has
$$\varphi(\bar{x}' ; \bar{y}), \bar{x}' \cap \bar{y} = \emptyset$$
$$\psi(\bar{y}' ; \bar{z}), \bar{y}' \cap \bar{z} = \emptyset$$

and defines

$$\varphi(\bar{x}' ; \bar{y}) ; \psi(\bar{y}' ; \bar{z}) \overset{\text{def}}{=} \exists \bar{u}. \varphi[\bar{u}/\bar{y}] \land \psi[\bar{u}/\bar{y}'] \quad \text{with} \quad \bar{u} \cap \bar{x}' = \bar{u} \cap \bar{z} = \emptyset$$

**Theorem:** $C[\varphi ; \psi] = C[\varphi] ; C[\psi]$
Expressing representation relations syntactically

Binary representation relation $\beta$ is expressed by a first-order predicate $\rho$ relating values of abstract representation variables $\bar{a}$ to those of concrete representation variables $\bar{c}$, and lets the values of normal variables $\bar{x}$ – i.e., of non-representation variables – unchanged:

$$\beta(\bar{a}', \bar{x}'; \bar{c}, \bar{x}) \overset{\text{def}}{=} \rho(\bar{a}'; \bar{c}) \land \bar{x}' = \bar{x}, \text{ for appr. } \rho$$
Proof of downwards simulation theorem for partial correctness (1)

1. Case $\varphi \Rightarrow \psi$:

Given: $\varphi(x_0 ; x, a)$

$\psi(x_0 ; x, a)$

$\Rightarrow (\varphi \Rightarrow \psi) = (\forall x_0 . \varphi[x', a'/x, a] \rightarrow \psi)$

2. Case $(\varphi \Rightarrow \psi) ; \beta$:

$(\varphi \Rightarrow \psi)(x', a' ; x, a)$

$(\rho[a'/a] \land x' = x)(x', a' ; x, c)$

$\Rightarrow$

$(\varphi \Rightarrow \psi) ; (\rho[a'/a] \land x' = x) = \exists u, a . (\varphi \Rightarrow \psi)[u/x] \land \rho \land u = x$

$= (\exists a . (\varphi \Rightarrow \psi) \land \rho)(x', a' ; x, c)$
Proof of downwards simulation theorem for partial correctness (2)

3. Case $\beta \leadsto (\varphi \leadsto \psi) ; \beta$:

$$\rho[a'/a] \land x' = x \leadsto (\varphi \leadsto \psi) ; (\rho[a'/a] \land x' = x) = \text{ (by (2))}$$

$$= \beta$$

$$\forall x'_0, a'_0. (\rho[a'_0/a] \land x'_0 = x)[x', c'/x, c] \rightarrow (\exists a. \rho \land \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi)$$

$$= \rho[a'_0/a] \land x'_0 = x \leadsto \exists a. \rho \land \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi$$

QED

I.e., $S \subseteq \beta \leadsto (\varphi \leadsto \psi) ; \beta$ iff

$$\models \left\{ \rho[a'/a] \land x'_0 = x \right\} S \left\{ \exists a. \rho \land \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi \right\}$$
Simplification possible in some cases

**Theorem:** For $\bar{x}$ list of program variables, $\bar{x}_0$ a list of logical variables occurring free in assertions $\varphi$ and $\psi$, let $\bar{y}_0$ be a list of fresh logical variables of the same length as of $\bar{x}$. Then:

$$\varphi \rightsquigarrow \psi = \bar{x} = \bar{y}_0 \rightsquigarrow \forall \bar{x}_0 (\varphi[\bar{y}_0/\bar{x}] \rightarrow \psi)$$

**Theorem:** For preconditions of form $\bar{x} = \bar{y}_0$ of if $\rho^{-1}$ is a total function $S$ downward simulates $\varphi \rightsquigarrow \psi$ under representation relation $\rho$ iff

$$|= \left\{ \exists a (\rho \land \varphi) \right\} S \left\{ \exists a (\rho \land \psi) \right\}.$$
Unfortunately, the relational model for partial correctness is not appropriate for all of the methods we would like to discuss.

Instead we need **four** of them:

<table>
<thead>
<tr>
<th>partial corr.</th>
<th>relations</th>
<th>pred. transformers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoare (p.c.), Hehner</td>
<td>Gardiner</td>
<td></td>
</tr>
<tr>
<td>total corr.</td>
<td>VDM, Z, Reynolds, Hoare (t.c.), Abadi &amp; Lamport, Lynch</td>
<td>Back &amp; von Wright, Morgan</td>
</tr>
</tbody>
</table>
Syntax & semantics

*Prog* is a reasonably broad language to express the essential features of the treated methods (from the data refinement point of view).

\[
Prog \ni S ::= \varphi \leadsto \psi \mid X \mid S_1 ; S_2 \mid S_1 \parallel S_2 \mid \mu X . S
\]

with relational semantics for partial correctness \( \mathcal{P}[.] : Prog \to 2^{\Sigma \times \Sigma} \) such that

\[
\{ \varphi \} S \{ \psi \} \text{ is valid iff } \mathcal{P}[S] \subseteq \mathcal{P}[\varphi \leadsto \psi] .
\]

**Example 4** \((x, y, s = x_0, y_0, s_0) \leadsto (x, y, s = x_0, y_0, s_0 \cup \{x_0\})\) expresses \(s := s \cup \{x\}\) in Example 1.
4 semantics of programs

<table>
<thead>
<tr>
<th></th>
<th>relations</th>
<th>pred. transformers</th>
</tr>
</thead>
<tbody>
<tr>
<td>partial corr.</td>
<td>$\mathcal{P}[S] \subseteq \Sigma^2$</td>
<td>$\mathcal{wlp}(S) : 2^\Sigma \xrightarrow{\text{mon}} 2^\Sigma$</td>
</tr>
<tr>
<td>total corr.</td>
<td>$\mathcal{P}<em>\perp[S] \subseteq \Sigma</em>\perp^2$</td>
<td>$\mathcal{wp}(S) : 2^\Sigma \xrightarrow{\text{mon}} 2^\Sigma$</td>
</tr>
</tbody>
</table>
vertical connection: separation theorems

**total corr.** = **partial corr.** + **termination**

\[ [\varphi] S[\psi] \iff \{\varphi\} S \{\psi\} \land [\varphi] S[\text{true}] \]

\[ wp(S')\psi \iff wlp(S')\psi \land wp(S')\text{true} \]

horizontal connection: Galois connection \((\star, [.]\)\)

\[ \sigma \in [r] s \iff \forall \tau ((\sigma, \tau) \in r \Rightarrow \tau \in s) \]

\[ (\sigma, \tau) \in \star P \iff \forall s (\sigma \in P(s) \Rightarrow \tau \in s) \]
Relating semantics (2)

\[ \Sigma \text{mon} \rightarrow 2^\Sigma \subseteq [2^\Sigma \overset{\text{mon}}{\rightarrow} 2^\Sigma], \subseteq \]

\[ \Sigma \text{mon} \rightarrow 2^\Sigma \subseteq [2^\Sigma \overset{\text{mon}}{\rightarrow} 2^\Sigma], \subseteq \]

\[ (2^{\Sigma^2}, \supseteq) \]

\[ (2^{\Sigma^2}, \supseteq) \]

\[ \mathcal{P}[.] \]

\[ \mathcal{P}[.] \]

\[ \mathcal{P}[.] \]

\[ (2^{\Sigma^2}, \supseteq) \]

\[ (2^{\Sigma^2}, \supseteq) \]

\[ wlp \]

\[ wp \]
adaptation axiom

\[ \vdash \{ \pi \} \varphi \rightsquigarrow \psi \{ \exists y_0 \left( \pi \left[ y_0 / x \right] \land \forall x_0 \left( \varphi \left[ y_0 / x \right] \Rightarrow \psi \right) \right) \} \]

\rightsquigarrow\text{-substitution rule}

\[ \begin{array}{c}
\{ \varphi \} \; \{ \psi \}, \{ \pi \} \; S_2 \left[ \varphi \rightsquigarrow \psi / x \right] \{ \theta \} \\
\{ \pi \} \; S_2 \left[ S_1 / x \right] \{ \theta \}
\end{array} \]

recursion rule

\[ \begin{array}{c}
\{ \pi \} \; S \left[ \pi \rightsquigarrow \theta / x \right] \{ \theta \} \\
\{ \pi \} \; \mu X. S \{ \theta \}
\end{array} \]
Proof system (2)

composition rule

\[
\begin{array}{c}
\{\pi\} S_1 \{\varphi\}, \{\varphi\} S_2 \{\rho\} \\
\{\pi\} S_1 ; S_2 \{\rho\}
\end{array}
\]

choice rule

\[
\begin{array}{c}
\{\pi\} S_1 \{\rho\}, \{\pi\} S_2 \{\rho\} \\
\{\pi\} S_1 \parallel S_2 \{\rho\}
\end{array}
\]

consequence rule

\[
\pi \Rightarrow \varphi, \{\varphi\} S \{\psi\}, \psi \Rightarrow \rho \\
\{\pi\} S \{\rho\}
\]

= sound and (relatively) complete proof system (in the sense of Cook).
Reynolds’ method

... we must transform our program to replace the abstract variable by a concrete variable representing its value. To do this, we will use the following general method:

R1. One or more concrete variables are introduced to store the representation of one or more abstract variables.

R2. A general invariant called the representation invariant is introduced, which describes the relationship between the abstract and concrete variables.

R3. Each assignment to an abstract variable (or more generally, each assignment that affects the representation invariant) is augmented with assignments to the concrete variables that re-establish the representation invariant (or achieve it, in case of an initialization).

R4. Each expression that contains an abstract variable but occurs outside of an assignment to an abstract variable is replaced by an expression that does not contain abstract variables but is guaranteed by the representation invariant to have the same value.

The last step will render the abstract variables auxiliary, so that their declarations and assignments can be eliminated. [Reynolds 1981]
Theorems

**Theorem 2** Each data refinement step following Reynolds’ recipe induces a case of total correctness downward simulation in the relational setting.

**Theorem 3** Each data refinement step in VDM induces a case of total correctness downward simulation in the relational setting.
begin

\textbf{var} \ s : \text{set of } \mathbb{N} ; \ l : \mathbb{N}^* ;

\ s := \{5\} ;

\{\textbf{geninv l: } \text{elems}(l) = s\}

\ S_1 ;

\ s := s \cup \{x\} ;

\ S_2 ;

\ y := \text{a member of } s ;

\textbf{end}
Example with Reynolds’ method: step R3

begin

var $s : \text{set of } \mathbb{N} ; l : \mathbb{N}^*$;

$s := \{5\} ; l := \langle 5 \rangle$;

$\{\text{geninv } l: \text{elems}(l) = s\}$

$S_1$;

$\langle s := s \cup \{x\} ; l := \text{append}(l, x) \rangle$;

$S_2$;

$y := \text{a member of } s$;

end
begin

var $s : \text{set of } \mathbb{N} ; l : \mathbb{N}^*$;

$s := \{5\} ; l := \langle 5 \rangle$;

$\{\text{geninv } l: \text{elems}(l) = s\}$

$S_1$;

$\langle s := s \cup \{x\} ; l := \text{append}(l, x) \rangle$;

$S_2$;

$y := \text{first}(l)$;

end
begin

\textbf{var} \ l : \mathbb{N}^* ;

\ l := \langle 5 \rangle ;

\ S_1 ;

\ l := \text{append}(\ l , \ x) ;

\ S_2 ;

\ y := \text{first}(\ l ) ;

end
We specify the abstract and concrete level operations of our example using VDM:

First the state variable is declared.

\[ s : \text{set of } \mathbb{N} \quad | \quad l : \mathbb{N}^* \]

Then its initial value is fixed.

\[ s_0 = \{5\} \quad | \quad l_0 = [5] \]
The operations are specified next.

\[ \text{Adda} \ (x : \mathbb{N}) \]
\[ \text{ext wr } s : \text{set of } \mathbb{N} \]
\[ \text{post } s = s \cup \{x\} \]

\[ \text{Geta} \ (y : \mathbb{N}) \]
\[ \text{ext rd } s : \text{set of } \mathbb{N} \]
\[ \text{pre } s \neq \emptyset \]
\[ \text{post } y \in s \]

\[ \text{Addc} \ (x : \mathbb{N}) \]
\[ \text{ext wr } l : \mathbb{N}^* \]
\[ \text{post } l = l \cup x \]

\[ \text{Getc} \ (y : \mathbb{N}) \]
\[ \text{ext rd } l : \mathbb{N}^* \]
\[ \text{pre } \text{len}(l) > 0 \]
\[ \text{post } y = \text{first}(l) \]
The connection between the state spaces of the two levels under consideration is provided by \textit{retrieve function} 

\[
\text{elems} : \mathbb{N}^* \rightarrow \text{set of } \mathbb{N}.
\]

The \textit{concrete} data model ($\mathbb{N}^*$) shall be \textit{adequate}, i.e., every abstract value has a corresponding concrete value:

\[
l : \mathbb{N}^* \vdash \exists s : \text{set of } \mathbb{N} \ (\text{elems} (l) = s)
\]

All images of \textit{concrete initial} states must be \textit{abstract initial} states.

\[
\vdash \text{elems} ([5]) = \{5\}
\]
The concrete precondition shall hold whenever the abstract precondition does. (*domain rule* for *GETa* and *GETc*)

\[ l : \mathbb{N}^* , \text{elems} (l) \neq \emptyset \vdash \text{len}(l) > 0 \]

The concrete operation should not break the abstract postcondition. (*result rule* for *ADDa* and *ADDCc*)

\[ \leadsto \] \[ \hat{l} , l : \mathbb{N}^* , \text{elems} (l) \neq \emptyset , l = \leadsto \hat{l} \hat{\leftarrow} x \], \[ \vdash \text{elems} (\hat{l}) = \text{elems} (\hat{l}) \cup \{x\} \]
All references can be found in:

Total correctness semantics for relations (1)

1. To express nontermination, add \( \bot \) to \( \Sigma \): \( \Sigma \bot \)

2. In the context of total correctness, termination is better than nontermination: a terminating computation should refine a nonterminating one \( \Rightarrow \)
   a. Interpret refinement by \( \subseteq \)
   b. Model nontermination by the whole of \( \Sigma \bot \)

3. From the viewpoint of total correctness, the nondeterministic possibility of nontermination is as bad as guaranteed nontermination. Consistent with (2b).
Total correctness semantics for relations (2)

4. (2b) implies the following domain \( R_\bot \) for the total correctness semantics of relations \( r \):
   \[
   \begin{align*}
   & \{ \bot \} \times \Sigma_\bot \subseteq r \quad \text{(nontermination is strict)} \\
   & (\sigma, \bot) \in r \Rightarrow (\sigma, \tau) \in r \text{ for all } \tau \in \Sigma_\bot \quad \text{(by (2b) and (3))}
   \end{align*}
   \]

   \((R_\bot, \supseteq)\) is a cpo with least element \( U \overset{\text{def}}{=} \Sigma_\bot \times \Sigma_\bot \) and “;” and “∪” not continuous but monotonic.

5. Introduce as finitary sets:
   \[
   \begin{align*}
   & \text{all finite subsets of } \Sigma \text{ (} \emptyset \text{ included!)} \\
   & \Sigma_\bot
   \end{align*}
   \]

   and consider the SMYTH cpo \((R_F, \supseteq)\) of finitary binary relations \( r \) s.t., for all \( \sigma \in \Sigma \), \( \{ \tau \mid (\sigma, \tau) \in r \} \) is finitary.

   In \((R_F, \supseteq)\) “;” and “∪” are continuous.
6. **Soundness** of upward and downward simulation for total correctness only holds when *all relations on the abstract level are finitary*.

**Completeness** of upward and downward simulation additionally requires that the *finalization operator* $AF$ on the abstract level *terminates weakly*, i.e., $AF$ does not involve diverging computations on undefined operations:

$$\Sigma = \text{wt}(AF) \overset{\text{def}}{=} \{ \sigma \mid (\sigma, \bot) \notin AF \}$$
7. Define total correctness of $S$ w.r.t. precondition $\varphi$ and postcondition $\psi$ by:

For all $\sigma$ and $\tau$, if $\varphi$ holds in $\sigma$, then:

(i) $S$ terminates in $\varsigma$, and

(ii) $\psi$ holds in all final states of $S$ thus obtained

Notation: $\models [\varphi] S [\psi]$

We define $\varphi$ and $\psi$ s.t. the Galois connection holds again:

$\models [\varphi] S [\psi] \iff \models \varphi ; S \subseteq \psi \iff \models S \subseteq (\varphi \hookrightarrow \psi) \left( \iff \models \varphi \subseteq [S] \psi \right)$
Galois connection for total correctness (2)

8. In the context of total correctness one must define the meaning of expressions s.a. \( y \div 0 \) and \( y \div (x - x) > z \).
There is only one choice: \( \bot_{VAL} \) and \( \bot_{B} \).

\[ \Rightarrow \text{Need for 3-valued logic with "¬" monotone on cpo } B_{\bot} \]
and \( tt \lor \bot = \bot \lor tt = tt \) (otherwise, \( \exists x. \frac{1}{x} = 1 \) denotes \( \bot_{B} \), and not \( tt \) as should be), i.e., our logic is non-strict [Jones & Middelburg '94].

9. The total correctness semantics of assertions \( \mathcal{T}_{\bot}[\cdot] \)

\[ \mathcal{T}_{\bot}[\cdot] : Pred \rightarrow (\Sigma_{\bot} \rightarrow B_{\bot}) \]

is strict in \( \sigma \in \Sigma_{\bot} \).
10. **Assertions** in total correctness formulae $[\varphi] S [\psi]$ should be interpreted as *subsets of* $\Gamma \times \Sigma$ of $\Gamma_\bot \times \Sigma_\bot$.

Otherwise, the intended interpretation

\[
\ldots \text{ if } \varphi \text{ holds then } S \text{ terminates and } \ldots
\]

is **violated**!

$\Rightarrow$ *In the context of total correctness pre- and postconditions are 2-valued!*

$\Rightarrow$ $C_\bot [\varphi] \overset{\text{def}}{=} \{(\gamma, \sigma) \mid T_\bot [\varphi](\gamma, \sigma) = tt\}$

Because of strictness of $T_\bot$ in $\gamma$ and $\sigma$ one has: $(\sigma, \bot) \notin C_\bot [\varphi]$. 
Galois connection for total correctness (4)

11. Defining for $\sigma, \tau \in \Sigma_\perp$

$$(\sigma, \tau) \in \mathcal{P}_\perp[\varphi \sim \psi] \overset{\text{def}}{=} \forall \gamma \in \Gamma((\gamma, \sigma) \in \mathcal{C}_\perp[\varphi] \Rightarrow (\gamma, \tau) \in \mathcal{C}_\perp[\psi])$$

has the following consequences:

The total correctness semantics $\mathcal{P}_\perp[\varphi \sim \psi]$ differs from its partial correctness semantics only in that:

a. $(\sigma, \perp)$ added in the total correctness semantics only if
   $$\forall \gamma. (\gamma, \sigma) \notin \mathcal{C}_\perp[\varphi]$$
   since $(\gamma, \perp) \notin \mathcal{C}_\perp[\psi]$ by (10).

b. The addition of $(\sigma, \perp)$ implies the addition of $(\sigma, \tau)$ for all $\tau \in \Sigma_\perp$.

c. $\{\perp\} \times \Sigma_\perp$ is added (a special case of (a)).
Proof sketch for simulation theorem (1)  
(total correctness, relational semantics, downward simulation)

12. a. The clue is the observation that *the partial correctness solution is almost the total correctness solution* for the problem of finding the *maximal* relation \( \varphi' \rightsquigarrow \psi' \) s.t. \( \varphi' \rightsquigarrow \psi' \) downward simulates \( \varphi \rightsquigarrow \psi \) w.r.t. \( \beta \).

b. Modulo possible differences in the evaluation of the predicates \( \varphi, \psi, \) and \( \rho \) *only state pairs* \( (\sigma, \bot) \) *must be added* to the partial correctness solution *exactly for those* \( \sigma \) *for which nontermination of* \( \beta \rightsquigarrow (\varphi \rightsquigarrow \psi); \beta \) *is possible*.

c. Define \( wt(r) \equiv \{ \sigma | (\sigma, \bot) \notin r \} \).
Then \( wt(\beta \rightsquigarrow (\varphi \rightsquigarrow \psi); \beta) = C_\bot[\exists a. (\beta \land \exists x_0. \varphi)] \), where \( x_0 \) lists the free logical variables of \( \varphi \) and \( \psi \).
Proof sketch for simulation theorem (2) (total correctness, relational semantics, downward simulation)

13. a. **Lemma:** \( P_\perp[(\pi \land \varphi) \leadsto \psi] = \)

\[ \{(\sigma, \tau) \in \Sigma_\perp | \sigma \in C_\perp[\pi] \Rightarrow (\sigma, \tau) \in P_\perp[\varphi \leadsto \psi]\} \]

b. **Lemma:** For \( \sigma, \tau \in \Sigma \),

\[ (\sigma, \tau) \in P_\perp[max_{pc}] \iff A_\perp[\beta] ; \{(\sigma, \tau)\} \subseteq P_\perp[\varphi \leadsto \psi] ; A_\perp[\beta], \]

where \( max_{pc} \) expresses the partial correctness solution for the downwards simulation problem, and \( A_\perp \) is the appropriate total correctness semantics for abstraction relations.
14. **Theorem:** The following specification statement expresses the maximal solution for total correctness and downward simulation of our simulation problem:

\[
\exists a(\beta \land \exists x_0.\varphi) \land \rho[a'_0/a] \land x = x'_0 \leadsto \exists a(\rho \land \forall x_0(\varphi[x'_0 a'_0/x a] \Rightarrow \psi))
\]

expresses \( wt(\beta \leadsto (\varphi \leadsto \psi) ; \beta) \)

**Proof:**

- By (12c) we express the domain of weak termination \( wt(\beta \leadsto (\varphi \leadsto \psi) ; \beta) \) by \( \exists a(\beta \land \exists x_0.\varphi). \)

- By (13a) we include in \( \mathcal{P}_\perp[\exists a.(\beta \land \exists x_0.\varphi) \land \varphi' \leadsto \psi'] \) **all pairs** \((\sigma, \perp)\) **for which nontermination of** \( \beta \leadsto (\varphi \leadsto \psi) ; \beta \) **is possible.**

- By (13b) it suffices to take as \( \varphi' = \rho[a'_0/a] \land x = x'_0 \) and \( \psi' = \exists a(\rho \land \forall x_0(\varphi[x'_0 a'_0/x a] \Rightarrow \psi)) \) in the above expression, thus **adding** the appropriate pairs \((\sigma, \perp)\) to \( \text{max}_{pc} \). □