

Data Refinement: model-oriented proof methods and their comparison

Willem-Paul de Roever
University of Kiel, Germany

- 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?

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

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

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

$$S \subseteq T.$$

Refinement (3)

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

var s : *finset of* \mathbb{N} ; $s := \emptyset$;

S_1 ;

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

S_2 ;

$y := a$ *member of* s

end

begin

var l : \mathbb{N}^* ; $l := \text{nil}$;

S_1 ;

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

S_2 ;

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

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 Σ and Σ^A by $\Sigma \stackrel{\text{def}}{=} [\bar{x} \rightarrow \mathbb{V}]$ and $\Sigma^A \stackrel{\text{def}}{=} [\bar{x} \cup \bar{a} \rightarrow \mathbb{V}]$. 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} : $A_j \subseteq \Sigma^A \times \Sigma^A$.

Program Skeletons

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 $P(\mathcal{A}) = A_1 ; A_2 \cup A_3$ and $P(\mathcal{C}) = C_1 ; C_2 \cup C_3$.

Obviously, there are **infinitely many** program skeletons (unless $J = \emptyset$).

Data refinement (1)

Compare two levels of abstraction:

A _____ of data type A
 C _____ of data type C

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

C should refine/implement A .

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

When defining that C **refines** A , the particular way a data type representation is defined should, therefore, **not be observable**:

Data refinement (2)

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

$$\underbrace{CI ; \dots ; CF}_{CI, CF \text{ hide the transformation of } \bar{c} \text{ by } \{C_j\}_{j \in J}} \subseteq \underbrace{AI ; \dots ; AF}_{AI, AF \text{ hide the transformation of } \bar{a} \text{ by } \{A_j\}_{j \in J}}$$

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

$$CI ; P(\mathcal{C}) ; CF \subseteq AI ; P(\mathcal{A}) ; AF,$$

for all program skeletons P concerned. \Rightarrow

This involves proving **infinitely many** proof obligations.

Data refinement (3)

Definition 2 Data type $\mathcal{C} = (CI, (C_j)_{j \in J}, CF)$ refines data type $\mathcal{A} = (AI, (A_j)_{j \in J}, AF)$ iff, for all program skeletons P :

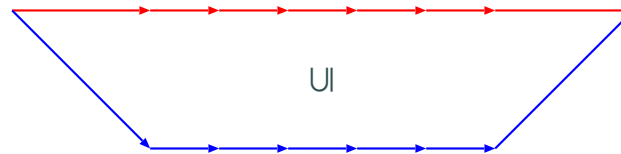
$$CI ; P(\mathcal{C}) ; CF \subseteq AI ; P(\mathcal{A}) ; AF$$

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

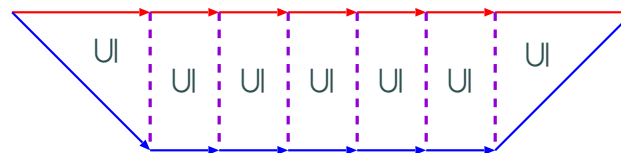
Hence, in order to prove data refinement, one has to prove **infinitely many proof obligations**.

Why simulation?

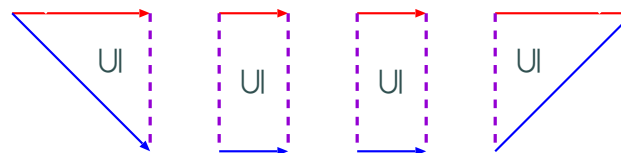
Instead of proving **infinitely many** proof obligations such as



directly, one would like to use **induction**. This requires invention of a relationship ρ between **abstract** and **concrete** level representation.



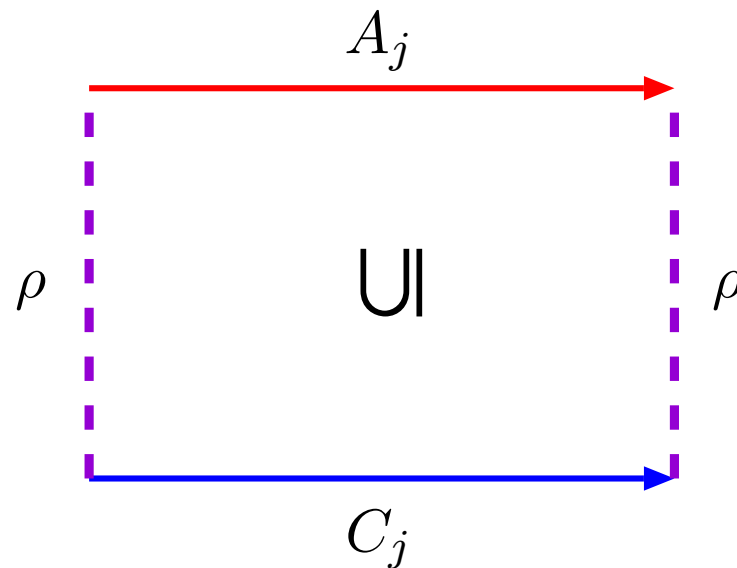
To focus on (the finite number of) base cases



one has to guarantee that *induction steps are for free*.

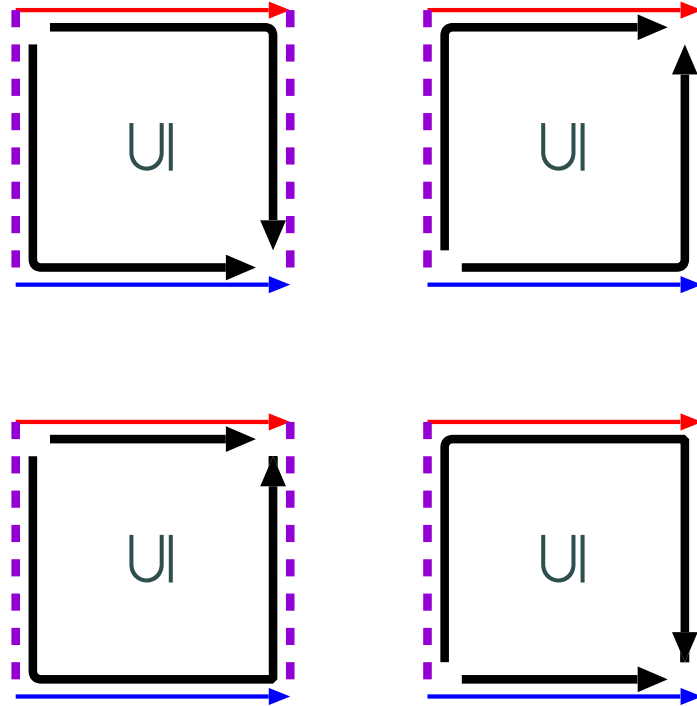
Local conditions for simulation (1)

Consider a relation $\rho \subseteq \Sigma^A \times \Sigma^C$ between abstract and concrete states. Then there are essentially **four** ways in which **weak commutativity** of diagram



can be defined, possibly using inverses of ρ .

Local conditions for simulation (2)

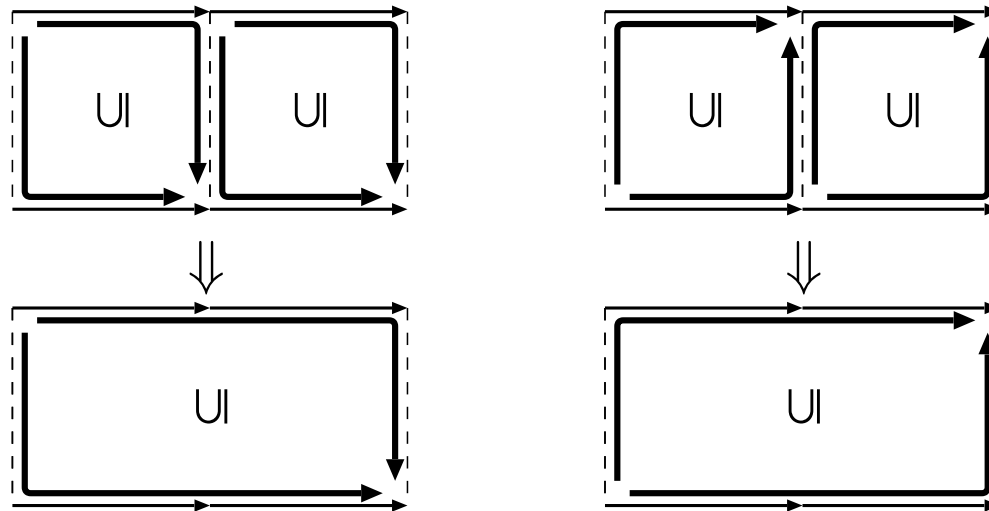


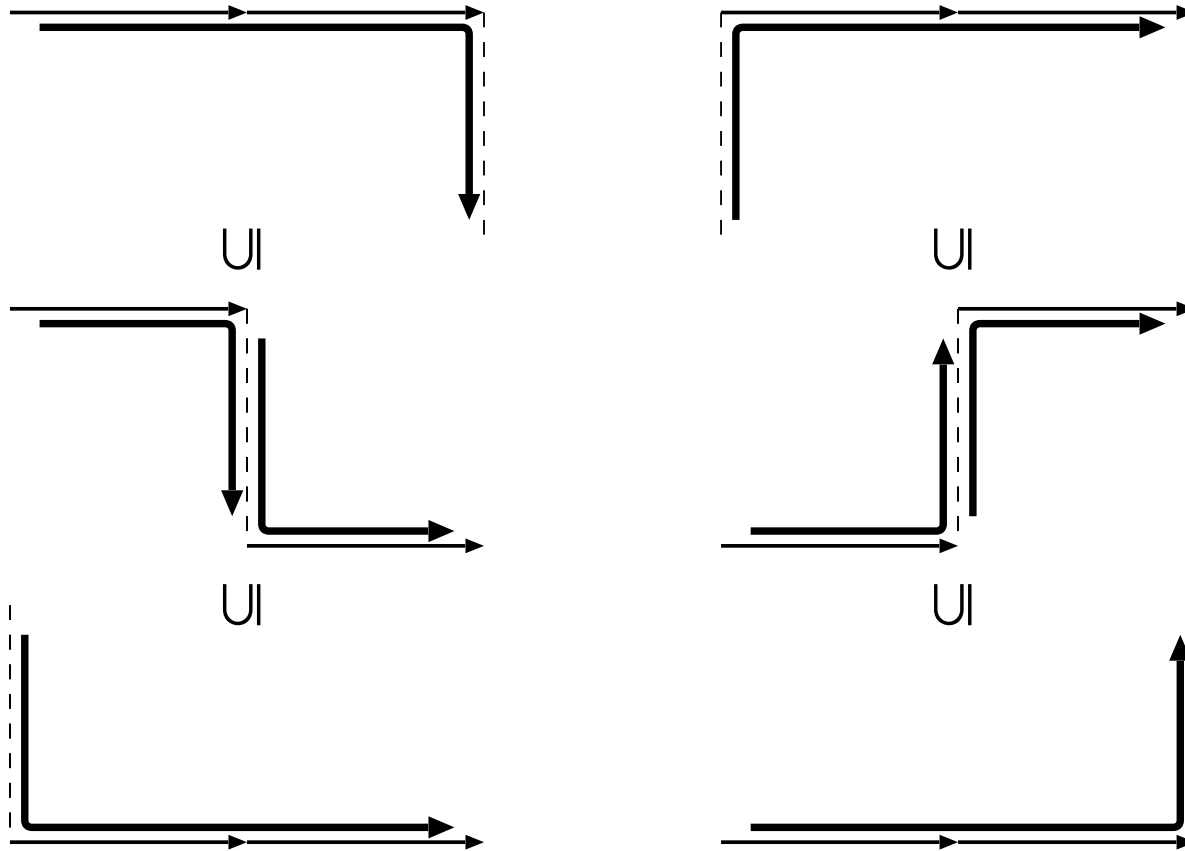
The induction step for sequential composition is free only for the first two, called downward and upward simulation, resp.

Technical note: The conditions for initialization and finalization are obtained by “identifying” either of the RHS/LHS pairs of corners in the diagrams above.

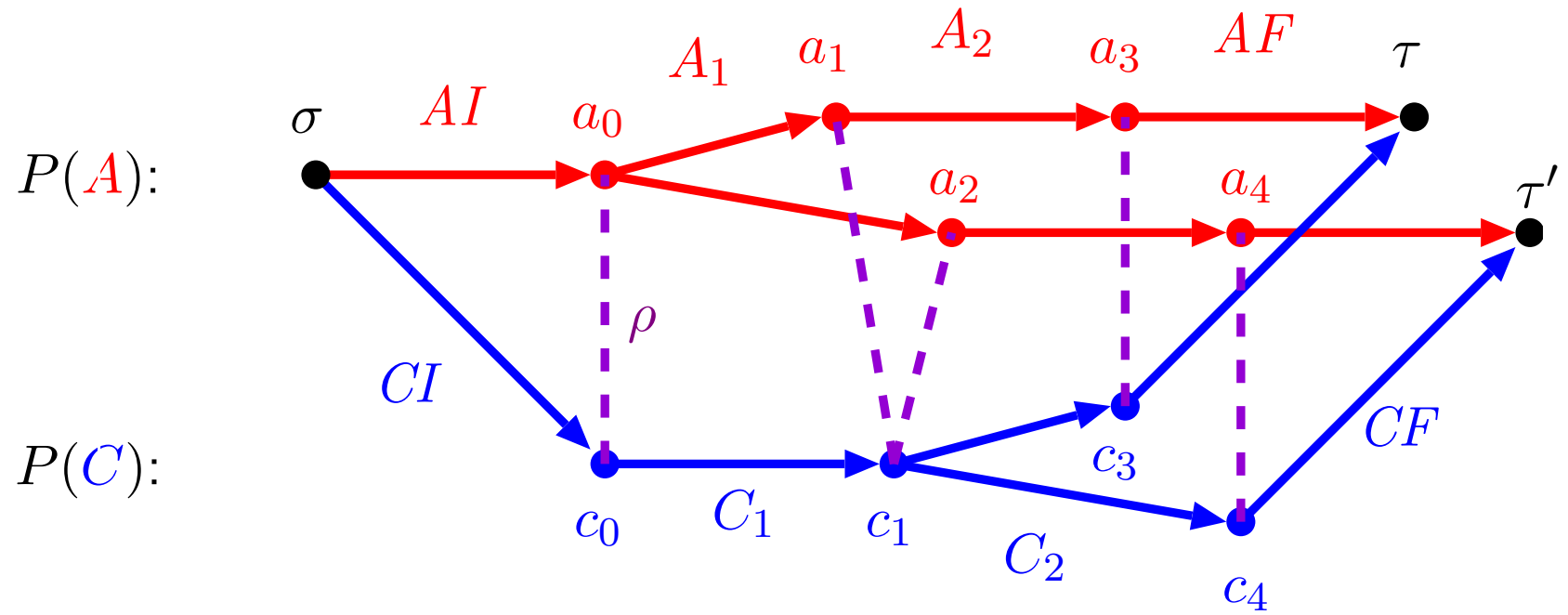
Soundness of simulation

Both downward and upward simulation are *sound* techniques for proving data refinement. The induction steps for sequential composition look as follows.





Incompleteness of downward simulation (1)



Incompleteness of downward simulation (2)

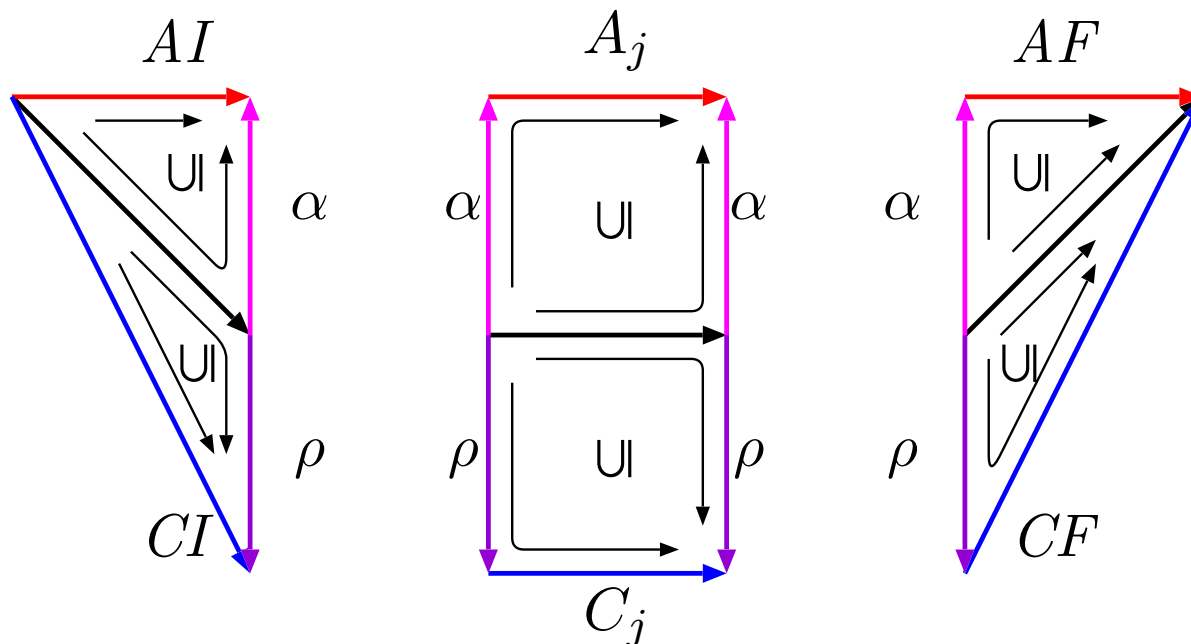
Assume ρ is a downward simulation relation between $(AI, (A_j)_{j \in \{1,2\}}, AF)$ and $(CI, (C_j)_{j \in \{1,2\}}, CF)$ where the relations in question are those depicted above.

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 ρ .
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 \mathcal{C} refines \mathcal{A} then there exist

- an intermediate data type \mathcal{B} ,
- a downward simulation relation ρ between \mathcal{B} and \mathcal{C} , and
- an upward simulation relation α between \mathcal{B} and \mathcal{A} .



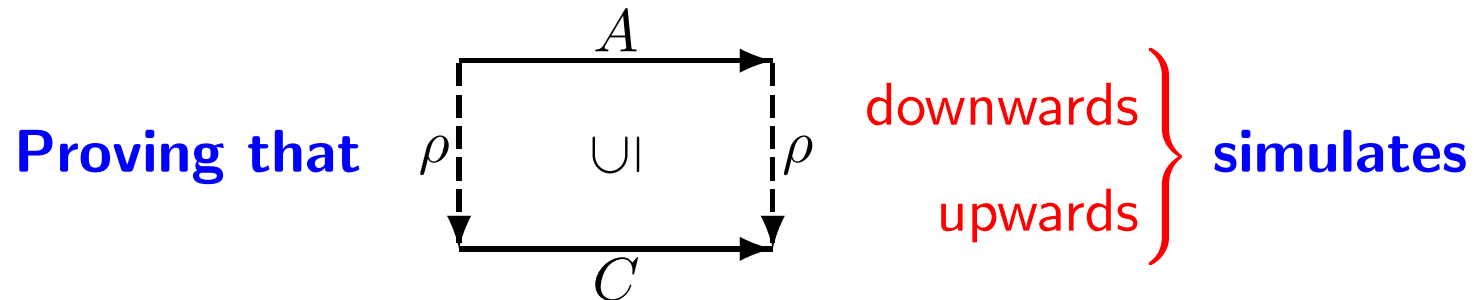
Numerous (formal) methods exist for writing specifications and refining those to implementations:

- 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.

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



So we have to prove **inclusion between relations**

$(\rho ; C \subseteq A ; \rho$ 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 $\{\varphi\} S \{\psi\}$ to specify operations, meaning:
predicate operation predicate

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

- *if* φ holds in initial state σ , *and if* S terminates for initial state σ in final state τ *then* ψ holds in τ .

Notation: $\models \{\varphi\} S \{\psi\}$ (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 *Logvar* of logical variables disjoint from *VAR*, the set of program variables, and to **logical variable states** $\Gamma \stackrel{\text{def}}{=} \text{Logvar} \rightarrow \text{VAL}, \gamma \in \Gamma$.

Using **logical variable states**, the meaning of $\models \{\varphi\} S \{\psi\}$ is:

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

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

$$\mathcal{C}[\varphi] \subseteq \Gamma \times \Sigma$$

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

$$\mathcal{P}[S] \subseteq \Sigma \times \Sigma$$

This implies: $\models \{\varphi\} S \{\psi\} \Leftrightarrow \models \varphi ; S \subseteq \psi$

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

- When operation op is specified by $\{\varphi\} op \{\psi\}$, we interpret op as the **maximal relation** r satisfying $\mathcal{C}[\varphi] ; r \subseteq \mathcal{C}[\psi]$.
- This max. relation is expressed by **specification statement** $\varphi \rightsquigarrow \psi$:

$$\mathcal{P}[\varphi \rightsquigarrow \psi] \stackrel{\text{def}}{=} \{(\sigma, \tau) \mid \forall \gamma \in \Gamma. (\gamma, \sigma) \in \mathcal{C}[\varphi] \Rightarrow (\gamma, \tau) \in \mathcal{C}[\psi]\}$$
- Since $\forall \sigma, \tau. \forall \gamma ((\gamma, \sigma) \in \mathcal{C}[\varphi] \wedge (\sigma, \tau) \in \mathcal{P}[S] \Rightarrow (\gamma, \tau) \in \mathcal{C}[\psi])$
 $\Leftrightarrow \forall \sigma, \tau. (\sigma, \tau) \in \mathcal{P}[S] \Rightarrow (\forall \gamma. (\gamma, \sigma) \in \mathcal{C}[\varphi] \Rightarrow (\gamma, \tau) \in \mathcal{C}[\psi])$
 $\Leftrightarrow \mathcal{P}[S] \subseteq \mathcal{P}[\varphi \rightsquigarrow \psi],$

one obtains

$$\{\varphi\} S \{\psi\} \Leftrightarrow \models S \subseteq \varphi \rightsquigarrow \psi.$$

- This clarifies why we interpret op as **maximal** relation:
 We **do not want to restrict** any **refinement** S of op **unnecessarily**.

Third connection

Let for $s \subseteq A \times C$ and $t \subseteq B \times C$

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

then $\models \{\varphi\} S \{\psi\} \Leftrightarrow \models \varphi \subseteq [S]\psi$

Proof: $\forall \sigma \tau. \forall \gamma. (\gamma, \sigma) \in \mathcal{C}[\varphi] \wedge (\sigma, \tau) \in \mathcal{P}[S] \Rightarrow (\gamma, \tau) \in \mathcal{C}[\psi]$
 $\Leftrightarrow \underbrace{\forall \gamma, \sigma. (\gamma, \sigma) \in \mathcal{C}[\varphi]} \Rightarrow \underbrace{(\forall \tau. (\sigma, \tau) \in \mathcal{P}[S] \Rightarrow (\gamma, \tau) \in \mathcal{C}[\psi])}$
 $\Leftrightarrow \forall \gamma, \sigma. (\gamma, \sigma) \in \mathcal{C}[\varphi] \Rightarrow (\gamma, \sigma) \in \mathcal{C}[[S]\psi]$
 $\Leftrightarrow \mathcal{C}[\varphi] \subseteq \mathcal{C}[[S]\psi]$

QED

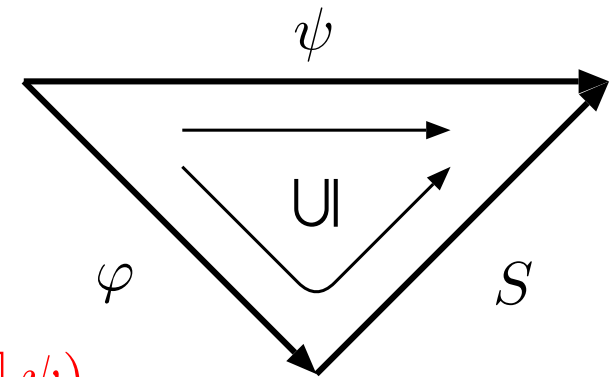
Galois connection

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

$$S \subseteq \varphi \rightsquigarrow \psi \quad \Leftrightarrow \quad \varphi ; S \subseteq \psi \quad (\Leftrightarrow \quad \varphi \subseteq [S] \psi)$$

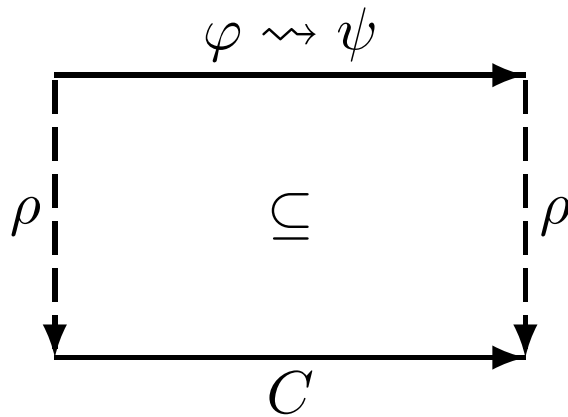
It depends on the program semantics chosen whether **total** or **partial correctness** is expressed; the equivalence of these inclusions holds in both cases.

This Galois connection is our main technical tool in relating relational to assertional characterizations of operations.



Assertional characterization of simulation

Problem: How to characterize the **maximal** relation C $?$ -simulating $\varphi \rightsquigarrow \psi$ under abstraction relation ρ as a **specification statement**:



$? = L$ or downwards / L^{-1} or upwards

\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 ρ as a specification statement.

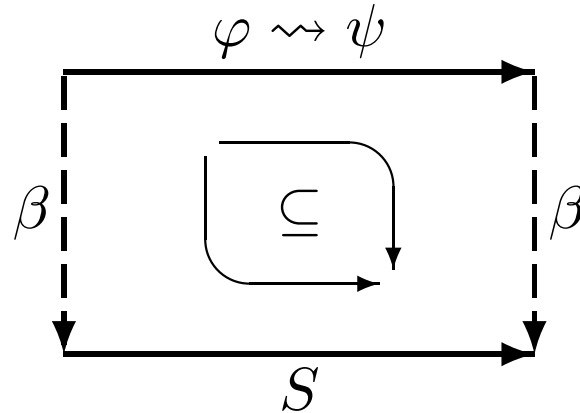
Solution for C :

(Partial correctness, relational semantics, downward simulation)

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

NB For the **total correctness** solution, **add** conjunct $\exists \bar{a} (\rho \wedge \exists \bar{x}_0 (\varphi))$ 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)

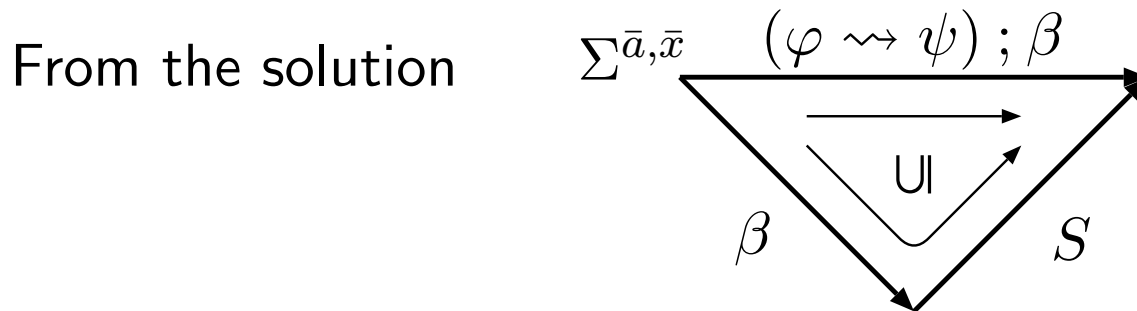


S downward simulates $\varphi \rightsquigarrow \psi$ w.r.t. β

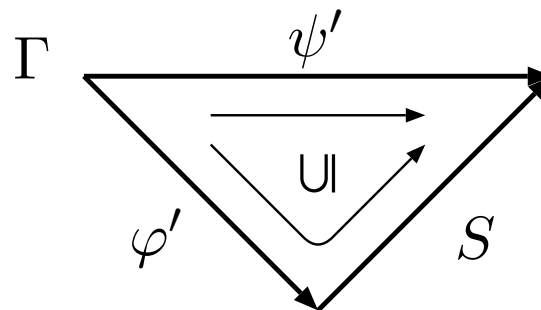
$$\iff \beta ; S \subseteq (\varphi \rightsquigarrow \psi) ; \beta$$

$$\iff S \subseteq \beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta$$

Proof sketch of the downward simulation theorem for partial correctness (2)



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 φ, ψ 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: $\mathcal{C}[\varphi] \subseteq \Sigma^{\bar{x}} \times \Sigma^{\bar{y}}$, $\mathcal{C}[\psi] \subseteq \Sigma^{\bar{x}} \times \Sigma^{\bar{z}}$, with $\Sigma^{\bar{u}} \stackrel{\text{def}}{=} [\bar{u} \rightarrow Val]$

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

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

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 σ by $v'(\sigma, \tau) = \sigma(v)$, and unprimed ones v in τ by $v(\sigma, \tau) = \tau(v)$, and define

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

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: $\mathcal{C}[\varphi \rightsquigarrow \psi] = \mathcal{C}[\varphi] \rightsquigarrow \mathcal{C}[\psi]$

Expressing “;” syntactically

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

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

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

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

Theorem: $\mathcal{C}[\varphi; \psi] = \mathcal{C}[\varphi]; \mathcal{C}[\psi]$

Expressing representation relations syntactically

Binary representation relation β is expressed by a first-order predicate ρ 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**:

$$\beta(\bar{a}', \bar{x}' ; \bar{c}, \bar{x}) \stackrel{\text{def}}{=} \rho(\bar{a}' ; \bar{c}) \wedge \bar{x}' = \bar{x}, \text{ for appr. } \rho$$

Proof of downwards simulation theorem for partial correctness (1)

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

$$\left. \begin{array}{l} \text{Given: } \varphi(x_0 ; x, a) \\ \psi(x_0 ; x, a) \end{array} \right\} \Rightarrow (\varphi \rightsquigarrow \psi) = (\forall x_0. \varphi[x', a'/x, a] \rightarrow \psi)$$

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

$$\left. \begin{array}{l} (\varphi \rightsquigarrow \psi)(x', a' ; x, a) \\ \underbrace{(\rho[a'/a] \wedge x' = x)(x', a' ; x, c)}_{=\beta} \end{array} \right\} \Rightarrow$$
$$\begin{aligned} (\varphi \rightsquigarrow \psi) ; (\rho[a'/a] \wedge x' = x) &= \underbrace{\exists u, a. (\varphi \rightsquigarrow \psi)[u/x] \wedge \rho \wedge u = x}_{=} \\ &= (\exists a. (\varphi \rightsquigarrow \psi) \wedge \rho)(x', a' ; x, c) \end{aligned}$$

Proof of downwards simulation theorem for partial correctness (2)

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

$$\underbrace{\rho[a'/a] \wedge x' = x}_{=\beta} \rightsquigarrow (\varphi \rightsquigarrow \psi); \underbrace{(\rho[a'/a] \wedge x' = x)}_{=\beta} = \quad (\text{by (2)})$$

$$\underbrace{\forall x'_0, a'_0. (\rho[a'_0/a] \wedge x'_0 = x)[x', c'/x, c] \rightarrow (\exists a. \rho \wedge \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi)}_{= \rho[a'_0/a] \wedge x'_0 = x \rightsquigarrow \exists a. \rho \wedge \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi}$$

QED

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

$$\models \left\{ \rho[a'_0/a] \wedge x'_0 = x \right\} S \left\{ \exists a. \rho \wedge \forall x_0. \varphi[x'_0, a'_0/x, a] \rightarrow \psi \right\}$$

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 φ and ψ , 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 ρ^{-1} is a total function S downward simulates $\varphi \rightsquigarrow \psi$ under representation relation ρ iff

$$\models \left\{ \exists a(\rho \wedge \varphi) \right\} S \left\{ \exists a(\rho \wedge \psi) \right\}.$$

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:

	relations	pred. transformers
partial corr.	Hoare (p.c.), Hehner	Gardiner
total corr.	VDM, Z, Reynolds, Hoare (t.c.), Abadi & Lamport, Lynch	Back & von Wright, Morgan

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

$$Prog \ni S ::= \varphi \rightsquigarrow \psi \mid X \mid S_1 ; S_2 \mid S_1 \square S_2 \mid \mu X.S$$

with relational semantics for partial correctness $\mathcal{P}[\cdot] : Prog \rightarrow 2^{\Sigma \times \Sigma}$
such that

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

Example 4 $(x, y, s = x_0, y_0, s_0) \rightsquigarrow (x, y, s = x_0, y_0, s_0 \cup \{x_0\})$ expresses $s := s \cup \{x\}$ in Example 1.

4 semantics of programs

	relations	pred. transformers
partial corr.	$\mathcal{P}[[S]] \subseteq \Sigma^2$	$wlp(S) : 2^\Sigma \xrightarrow{\text{mon}} 2^\Sigma$
total corr.	$\mathcal{P}_\perp[[S]] \subseteq \Sigma_\perp^2$	$wp(S) : 2^\Sigma \xrightarrow{\text{mon}} 2^\Sigma$

vertical connection: separation theorems

total corr. = partial corr. + termination

$$[\varphi]S[\psi] \Leftrightarrow \{\varphi\} S \{\psi\} \wedge [\varphi]S[\text{true}]$$

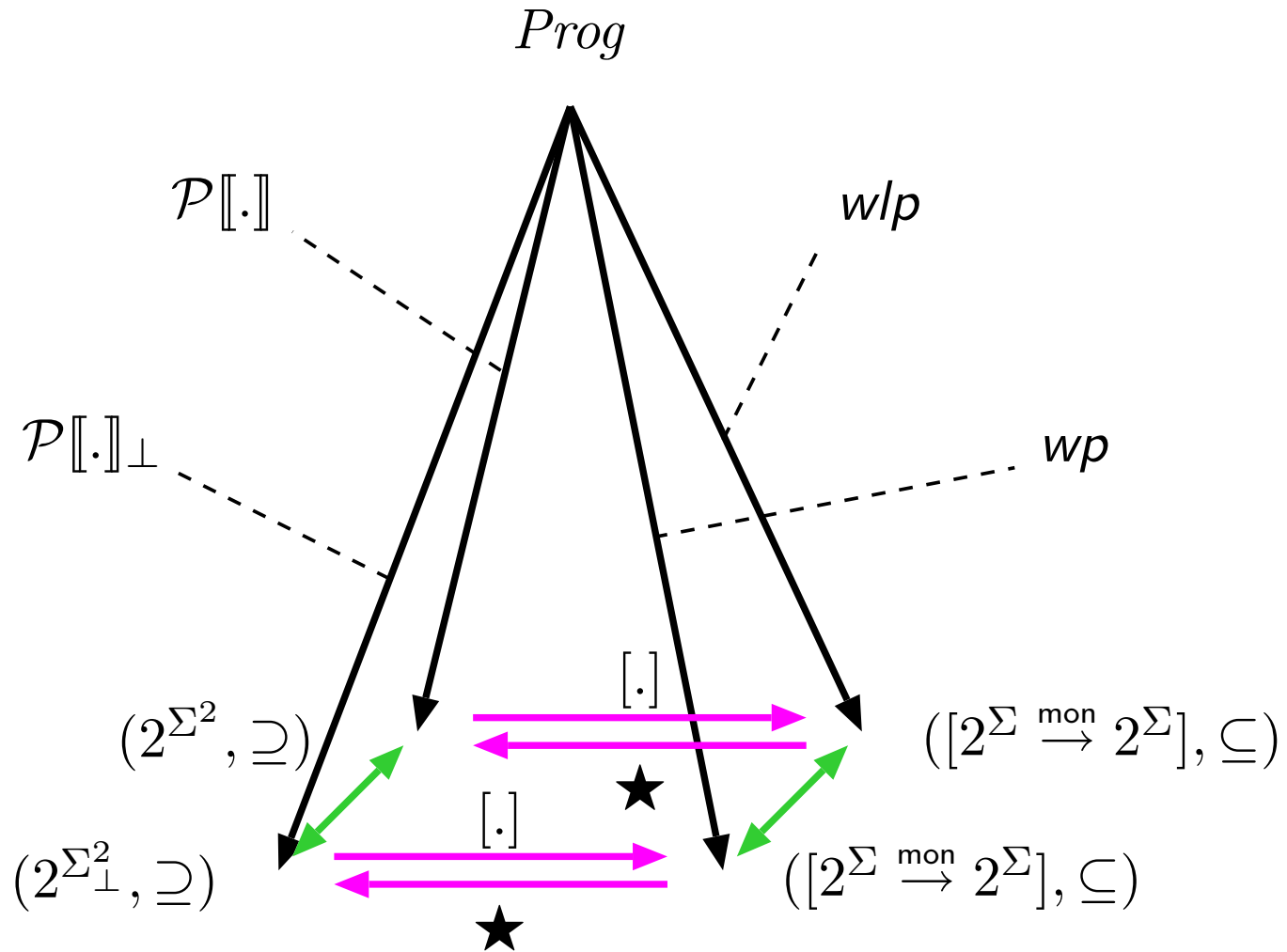
$$wp(S)\psi \Leftrightarrow wlp(S)\psi \wedge wp(S)\text{true}$$

horizontal connection: Galois connection ($\star, [\cdot]$)

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

$$(\sigma, \tau) \in \star P \Leftrightarrow \forall s (\sigma \in P(s) \Rightarrow \tau \in s)$$

Relating semantics (2)



adaptation axiom

$$\vdash \{\pi\} \varphi \rightsquigarrow \psi \{ \exists \bar{y}_0 (\pi[\bar{y}_0/\bar{x}] \wedge \forall \bar{x}_0 (\varphi[\bar{y}_0/\bar{x}] \Rightarrow \psi)) \}$$

\rightsquigarrow -substitution rule

$$\frac{\{\varphi\} S_1 \{\psi\}, \{\pi\} S_2[\varphi \rightsquigarrow \psi / X] \{\theta\}}{\{\pi\} S_2[S_1 / X] \{\theta\}}$$

recursion rule

$$\frac{\{\pi\} S[\pi \rightsquigarrow \theta / X] \{\theta\}}{\{\pi\} \mu X.S \{\theta\}}$$

composition rule

$$\frac{\{\pi\} S_1 \{\varphi\}, \{\varphi\} S_2 \{\rho\}}{\{\pi\} S_1 ; S_2 \{\rho\}}$$

choice rule

$$\frac{\{\pi\} S_1 \{\rho\}, \{\pi\} S_2 \{\rho\}}{\{\pi\} S_1 \square S_2 \{\rho\}}$$

consequence rule

$$\frac{\pi \Rightarrow \varphi, \{\varphi\} S \{\psi\}, \psi \Rightarrow \rho}{\{\pi\} S \{\rho\}}$$

= *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]

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.

Example with Reynolds' method: steps R1 and R2

begin

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

$s := \{5\}$;

{**geninv** l : $elems(l) = s$ }

S_1 ;

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

S_2 ;

$y :=$ *a member of* s ;

end

Example with Reynolds' method: step R3

begin

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

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

{**geninv** l : $elems(l) = s$ }

S_1 ;

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

S_2 ;

$y :=$ *a member of* s ;

end

Example with Reynolds' method: step R4

begin

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

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

{**geninv** l : $elems(l) = s$ }

S_1 ;

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

S_2 ;

$y := first(l)$;

end

Example with Reynolds' method: final step

begin

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

$l := \langle 5 \rangle$;

S_1 ;

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

S_2 ;

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

end

Example in VDM (1)

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

First the **state variable** is declared.

$s : \mathbf{set\ of\ } \mathbb{N}$		$l : \mathbb{N}^*$
-------------------------------------	--	--------------------

Then its **initial value** is fixed.

$s_0 = \{5\}$		$l_0 = [5]$
---------------	--	-------------

Example in VDM (2)

The operations are specified next.

ADDA ($x : \mathbb{N}$)

ext wr $s : \mathbf{set\ of}\ \mathbb{N}$

post $s = \overset{\leftarrow}{s} \cup \{x\}$

ADDc ($x : \mathbb{N}$)

ext wr $l : \mathbb{N}^*$

post $l = \overset{\leftarrow}{l} \frown x$

GETa ($y : \mathbb{N}$)

ext rd $s : \mathbf{set\ of}\ \mathbb{N}$

pre $s \neq \emptyset$

post $y \in s$

GETc ($y : \mathbb{N}$)

ext rd $l : \mathbb{N}^*$

pre $\text{len}(l) > 0$

post $y = \text{first}(l)$

VDM proof obligations (1)

The connection between the state spaces of the two levels under consideration is provided by *retrieve function*

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

The **concrete** data model (\mathbb{N}^*) shall be *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 **concrete initial** states must be **abstract initial** states.

$$\vdash \text{elems } ([5]) = \{5\}$$

VDM proof obligations (2)

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 *ADDc*)

$$\begin{aligned} \overleftarrow{l}, l : \mathbb{N}^*, \text{elems } (l) \neq \emptyset, l = \overleftarrow{l} \frown x \\ \vdash \text{elems } (l) = \text{elems } (\overleftarrow{l}) \cup \{x\} \end{aligned}$$

All references can be found in:

- Willem-Paul de Roever and Kai Engelhardt, **Data Refinement: Model-Oriented Proof Methods and their Comparison**, Cambridge Tracts in Theoretical Computer Science 47, Cambridge University Press, 1998.

Total correctness semantics for relations (1)

1. To express *nontermination*, add \perp to Σ : Σ_{\perp}
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 Σ_{\perp}*
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_{\perp} for the *total correctness semantics of relations* r :

- $\{\perp\} \times \Sigma_{\perp} \subseteq r$ (nontermination is **strict**)
- $(\sigma, \perp) \in r \Rightarrow (\sigma, \tau) \in r$ for all $\tau \in \Sigma_{\perp}$ (by (2b) and (3))

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

5. Introduce as **finitary** sets:

- all **finite** subsets of Σ (\emptyset included!)
- Σ_{\perp}

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

In $(R_{\mathcal{F}}, \supseteq)$ “;” and “U” are **continuous**.

Total correctness semantics for relations (3)

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:

$$\Sigma = wt(AF) \stackrel{\text{def}}{=} \{\sigma \mid (\sigma, \perp) \notin AF\}$$

Galois connection for total correctness (1)

7. Define **total correctness** of S w.r.t. **precondition** φ and **postcondition** ψ by:

For all σ and τ , if φ holds in σ , then:

(i) S terminates in ς , and

(ii) ψ holds in all final states of S thus obtained

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

We define φ and ψ s.t. the *Galois connection* holds again:

$$\models [\varphi] S [\psi] \Leftrightarrow \models \varphi ; S \subseteq \psi \Leftrightarrow \models S \subseteq (\varphi \rightsquigarrow \psi) \left(\Leftrightarrow \models \varphi \subseteq [S]\psi \right)$$

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: \perp_{VAL} and $\perp_{\mathbb{B}}$.

\Rightarrow Need for *3-valued logic* with “ \neg ” *monotone* on cpo \mathbb{B}_{\perp} and $tt \vee \perp = \perp \vee tt = tt$ (otherwise, $\exists x. \frac{1}{x} = 1$ denotes $\perp_{\mathbb{B}}$, and not tt as should be), i.e., our logic is non-strict [Jones & Middelburg '94].

9. The *total correctness semantics* of assertions $\mathcal{T}_{\perp}[\![\cdot]\!]$

$$\mathcal{T}_{\perp}[\![\cdot]\!] : Pred \rightarrow (\Sigma_{\perp} \rightarrow \mathbb{B}_{\perp})$$

is *strict* in $\sigma \in \Sigma_{\perp}$.

Galois connection for total correctness (3)

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

Otherwise, the intended interpretation

... if φ holds then S terminates and ...

is *violated!*

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

$\Rightarrow \mathcal{C}_{\perp}[\varphi] \stackrel{\text{def}}{=} \{(\gamma, \sigma) \mid \mathcal{T}_{\perp}[\varphi](\gamma, \sigma) = tt\}$

Because of strictness of \mathcal{T}_{\perp} in γ and σ one has: $(\sigma, \perp) \notin \mathcal{C}_{\perp}[\varphi]$.

Galois connection for total correctness (4)

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

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

has the following consequences:

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

- a. (σ, \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 (σ, \perp) *implies the addition of* (σ, τ) *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. β .
- b. Modulo possible differences in the evaluation of the predicates φ , ψ , and ρ *only state pairs (σ, \perp) must be added* to the partial correctness solution *exactly for those σ for which nontermination of $\beta \rightsquigarrow (\varphi \rightsquigarrow \psi); \beta$ is possible.*
- c. Define $wt(r) \stackrel{\text{def}}{=} \{\sigma \mid (\sigma, \perp) \notin r\}$.
Then $wt(\beta \rightsquigarrow (\varphi \rightsquigarrow \psi); \beta) = \mathcal{C}_{\perp} \llbracket \exists a. (\beta \wedge \exists x_0. \varphi) \rrbracket$, where x_0 lists the free logical variables of φ and ψ .

Proof sketch for simulation theorem (2) (total correctness, relational semantics, downward simulation)

13. a. **Lemma:** $\mathcal{P}_\perp \llbracket (\pi \wedge \varphi) \rightsquigarrow \psi \rrbracket =$
 $\{(\sigma, \tau) \in \Sigma_\perp \mid \sigma \in \mathcal{C}_\perp \llbracket \pi \rrbracket \Rightarrow (\sigma, \tau) \in \mathcal{P}_\perp \llbracket \varphi \rightsquigarrow \psi \rrbracket\}$

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

$$(\sigma, \tau) \in \mathcal{P}_\perp \llbracket max_{pc} \rrbracket \iff$$
$$\mathcal{A}_\perp \llbracket \beta \rrbracket ; \{(\sigma, \tau)\} \subseteq \mathcal{P}_\perp \llbracket \varphi \rightsquigarrow \psi \rrbracket ; \mathcal{A}_\perp \llbracket \beta \rrbracket,$$

where max_{pc} expresses the partial correctness solution for the downwards simulation problem, and \mathcal{A}_\perp is the appropriate total correctness semantics for abstraction relations.

Proof sketch for simulation theorem (3)

(total correctness, relational semantics, downward simulation)

14. **Theorem:** The following specification statement expresses the maximal solution for total correctness and downward simulation of our simulation problem:

$$\underbrace{\exists a(\beta \wedge \exists x_0.\varphi)} \wedge \rho[a'_0/a] \wedge x = x'_0 \rightsquigarrow \exists a(\rho \wedge \forall x_0(\varphi[x'_0 a'_0/x a] \Rightarrow \psi))$$

expresses $wt(\beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta)$

Proof:

- By (12c) we express the domain of weak termination $wt(\beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta)$ by $\exists a(\beta \wedge \exists x_0.\varphi)$.
- By (13a) we include in $\mathcal{P}_\perp[\exists a.(\beta \wedge \exists x_0.\varphi) \wedge \varphi' \rightsquigarrow \psi']$ *all pairs* (σ, \perp) *for which nontermination of* $\beta \rightsquigarrow (\varphi \rightsquigarrow \psi) ; \beta$ *is possible.*
- By (13b) it suffices to take as $\varphi' = \rho[a'_0/a] \wedge x = x'_0$ and $\psi' = \exists a(\rho \wedge \forall x_0(\varphi[x'_0 a'_0/x a] \Rightarrow \psi))$ in the above expression, thus *adding* the appropriate *pairs* (σ, \perp) to max_{pc} . □