A Summary of Refinement Theory

Maarten Fokkinga

Version of September 16, 1994

When can a program part be replaced by another? We discuss this question formally, and in doing so we summarise the work of Zwiers *et al* ("A Note on Compositional Refinement" and some private communication) and part of the work of Hoare *et al* ("The weakest prespecification" and "Prespecification in data refinement").

1 Preliminaries: relations

We don't want to talk about the syntax of programs or specifications. So there is no need to fix a language, whether it has "mixed terms" or not. We shall only discuss the semantics of programs and specifications, and for us these semantic entities are just *relations* (binary relations between input and output, or between pre-state and post-state, or between sets of traces).

We use several (more or less known) operations on relations, notably:

composition	$R \mathbin{;} S$
converse	$R^{\cup}$
factors	$R/S, \quad R\backslash S$
inclusion	$R \subseteq S$
identity	$I \quad .$

Of these, only the factors need some explanation. The *left factor of S in R*, denoted R/S, is the largest relation X such that $X \mathbin{;} S$ approximates R, or, in sloppy words, it is R with S cut off at the back (as suggested by the notation):

$$R/S \quad = \quad \cup(X : \quad X \mathbin{;} S \subseteq R : \quad X) \quad .$$

From this, one can prove a characterisation by means of a Galois-connection:

$$X \subseteq R/S \quad \equiv \quad X \mathbin{;} S \subseteq R \quad .$$

Thus, the factoring function $\_/S$ is the upper adjoint of the composing function $\_ \mathbin{;} S$. In view of the characterisation one might call R/S the weakest precondition of S with respect to R, since $X \mathbin{;} S \subseteq R$ expresses that after precondition X, program S gives no other results than allowed by R. We do not need this interpretation in the sequel, so we shall not elaborate this view any further. The interested reader should consult Hoare et al [3, 4].

The importance of the above Galois connection for formal proofs about relations in general cannot be overemphasised. One should be entirely familiar with its use: rewriting (blindfolded, I would say) a statement of the form $xxxx \mathbin{;} S \subseteq yyyy$ into the equivalent statement $xxxx \subseteq yyyy/S$, and vice versa. Note that in both statements S occurs at the same side of the term.

Here are several corollaries, easily provable from the characterisation. Those enclosed in square brackets are not used in the sequel. Note the similarities and differences with the laws for multiplication and division (viewing '$;$' as multiplication and '/' as division):

$$
\begin{array}{llcll}
 & R/S \mathbin{;} S & \subseteq & R & \text{“cancellation”} \\
 & I/S \mathbin{;} S & \subseteq & I & \text{“cancellation”} \\
 & R \mathbin{;} S/T & \subseteq & (R \mathbin{;} S)/T & \text{“fusion into numerator”} \\
 & R/S & \subseteq & R'/S \quad \text{if } R \subseteq R' & \text{“monotony in numerator”} \\
 & R/S' & \subseteq & R/S \quad \text{if } S \subseteq S' & \text{“antimonotony in denominator”} \\
 & (R/S)/T & = & R/(T \mathbin{;} S) & \text{“cascaded division”} \\
{[} & R/I & = & R &] \\
{[} & I & \subseteq & R/R &] \\
{[} & (R \cap S)/T & = & R/T \cap S/T &] \\
{[} & R/(S \cup T) & = & R/S \cap R/T \quad . &]
\end{array}
$$

In view of the first two of these properties, a left factor I/S (or R/S) is sometimes called an approximate left inverse of S (relative to R).

Right factors are defined similarly to left factors:

$$
\begin{array}{lcl}
S\backslash R & = & \text{the right factor of } S \text{ in } R \\
 & = & \text{the largest } X \text{ such that } S \mathbin{;} X \text{ approximates } R \\
 & = & \cup(X : \quad S \mathbin{;} X \subseteq R : \quad X) \\
 & = & \text{“} R \text{ with } S \text{ cut off at the front”} \\
X \subseteq S\backslash R & \equiv & S \mathbin{;} X \subseteq R \quad .
\end{array}
$$

So, $S\backslash\_$ is the upper adjoint of $S \mathbin{;} \_$. The properties are similar to those for left factors: interchange the arguments of each $;$ and replace $/$ by $\backslash$ while interchanging its arguments. The two kinds of factors associate with each other:

$$
R\backslash(S/T) \quad = \quad (R\backslash S)/T \quad .
$$

So we can write $R\backslash S/T$ without semantic ambiguity.

Recall further the following definitions:

$$\begin{array}{lcl}
R \text{ is functional} & \equiv & R^{\cup} ; R \subseteq I \\
R \text{ is injective} & \equiv & R ; R^{\cup} \subseteq I \\
R \text{ is total} & \equiv & I \subseteq R ; R^{\cup} \\
R \text{ is surjective} & \equiv & I \subseteq R^{\cup} ; R \quad .
\end{array}$$

As an aid to the memory, note that the statement $\ldots ; S \subseteq I$ asserts that there is no choice for the outcome of S — this occurs in the definition of functionality and injectivity. Similarly, the statement $I \subseteq S ; \ldots$ asserts that S accepts all input — this occurs in the definition of totality and surjectivity.

We shall not attempt to formulate all (or most?) of the theory in the language of category theory, order-enriched categories to be precise (since, at the moment, I'm not entirely familiar with order-enriched categories). The advantage would be a gain in generality, and possibly a simplification of various proofs by invoking well-known categorical theorems. Notice in particular that we apply associativity of composition always tacitly.

1.1 Comparison with Zwiers' language. Zwiers [6] considers n-ary rather than binary relations. However, binary relations seem to cover $n{+}1$-ary relations as well if one of its arguments is considered an n-ary vector. And unary relations can be identified with the binary relations that are included in the identity relation I.
[*CHECK: Do I loose something here?*]

2 Refinement and Simulation

We say that R *refines* S if $R \subseteq S$. This means that R can only produce outcomes that S can produce; it is a form of partial correctness of R with respect to S. The statement that R produces something whenever S does, is not captured by the refinement relation and need be expressed separately ("R is total on the domain of S").

In practice refinement is a very strong condition; a weaker condition is obtained by allowing for an additional interpretation and representation of the data handled by R and S, respectively. Then we speak of *simulation.*

So, consider the following relations:

$$\begin{array}{ccc}
 & \xrightarrow{S} & \\
\rho \uparrow & & \uparrow \rho \\
 & \xrightarrow{R} &
\end{array}$$

At the upper level, S stands for an abstract program or specification, whereas, at the lower level, R stands for a concrete program or specification. Relation ρ stands for a *retrieve* relation, interpreting concrete representations as abstract values.

2.1 Definition. There are at least four ways to define the notion "R ρ-simulates S":

$$\begin{array}{llll} \rho^{\cup} ; R ; \rho & \subseteq & S & \text{strong simulation} \\ R ; \rho & \subseteq & \rho ; S & \text{up simulation} \\ \rho^{\cup} ; R & \subseteq & S ; \rho^{\cup} & \text{down simulation} \\ R & \subseteq & \rho ; S ; \rho^{\cup} & \text{weak simulation} \quad . \end{array}$$

We call ρ here the *retrieve relation*. The adjective 'up' signifies the presence of ρ, which is drawn upwards in the picture; similarly, 'down' signifies the presence of $\rho^{\cup}$. The adjectives 'strong' and 'weak' come from the fact, proved below, that strong simulation implies the other simulations under mild conditions on ρ. (Since retrieve relations ρ are intended to interpret concrete representations as abstract values, totality of ρ is a weaker requirement than functionality.) We shall sometimes omit the adjectives if we refer to all of them jointly, and at other times we shall speak about s/u/d/w-simulation for clarity.

If the retrieve relation is the identity relation, then all four notions coincide with the notion that R refines S.

The following theorem asserts that the simulation notions are all equivalent if the retrieve relation is total and functional; in that case we might also say "ρ is a homomorphism from R to S".

2.2 Theorem. The simulations imply each other in the following way:

$$\begin{array}{llllll} \text{strong} & \Rightarrow & \text{up} & \Rightarrow & \text{weak,} & \text{if } \rho \text{ total} \\ \text{strong} & \Rightarrow & \text{down} & \Rightarrow & \text{weak,} & \text{if } \rho \text{ total} \\ \text{weak} & \Rightarrow & \text{up} & \Rightarrow & \text{strong,} & \text{if } \rho \text{ functional} \\ \text{weak} & \Rightarrow & \text{down} & \Rightarrow & \text{strong,} & \text{if } \rho \text{ functional} \quad . \end{array}$$

Proof. For the implication "weak $\Rightarrow$ down $\Rightarrow$ strong" we argue as follows:

$$\begin{array}{lll} & \rho^{\cup} ; R ; \rho \subseteq S & \text{(strong)} \\ \Leftarrow & \quad \text{transitivity } \subseteq \text{, and functionality } \rho \ (\rho^{\cup} ; \rho \subseteq I) & \\ & \rho^{\cup} ; R ; \rho \subseteq S ; \rho^{\cup} ; \rho & \\ \Leftarrow & \quad \text{Leibniz} & \\ & \rho^{\cup} ; R \subseteq S ; \rho^{\cup} & \text{(down)} \\ \Leftarrow & \quad \text{transitivity } \subseteq \text{, and functionality } \rho \ (\rho^{\cup} ; \rho \subseteq I) & \\ & \rho^{\cup} ; R \subseteq \rho^{\cup} ; \rho ; S ; \rho^{\cup} & \\ \Leftarrow & \quad \text{Leibniz} & \\ & R \subseteq \rho ; S ; \rho^{\cup} \quad . & \text{(weak)} \end{array}$$

The other parts are proved similarly.

2.3 Explicit quantification. The formulation of the simulation relations in first-order predicate logic may reveal more of the relative strength by making the quantifications explicit. For strong simulation we have (using x, x' for data at the lower R-level, and y, y' at the upper S-level):

$$
\begin{array}{ll}
 & R \text{ strongly } \rho\text{-simulates } S \\
\equiv & \\
 & \rho^{\cup} ; R ; \rho \subseteq S \\
\equiv & \\
 & \forall y, y' :: \quad (\exists x, x' :: \ y \, \langle \rho^{\cup} \rangle \, x \, \langle R \rangle \, x' \, \langle \rho \rangle \, y') \quad \Rightarrow \quad y \, \langle S \rangle \, y' \\
\equiv & \\
 & \forall y, y' :: \quad \forall x, x' :: \quad (y \, \langle \rho^{\cup} \rangle \, x \, \langle R \rangle \, x' \, \langle \rho \rangle \, y' \Rightarrow y \, \langle S \rangle \, y') \\
\equiv & \\
 & \forall x, x', y, y' :: \quad y \, \langle \rho \rangle \, x \, \langle R \rangle \, x' \, \langle \rho \rangle \, y' \Rightarrow y \, \langle S \rangle \, y' \\
\equiv & \\
 & \forall R\text{-run } \vec{x} :: \quad \forall \vec{y} \ \rho\text{-related to } \vec{x} :: \quad \vec{y} \text{ is a } S\text{-run} \quad .
\end{array}
$$

Similarly we find for weak simulation:

$$
\begin{array}{ll}
 & R \text{ weakly } \rho\text{-simulates } S \\
\equiv & \\
 & R \subseteq \rho ; S ; \rho^{\cup} \\
\equiv & \\
 & \forall x, x' :: \quad x \, \langle R \rangle \, x' \quad \Rightarrow \quad (\exists y, y' :: \ x \, \langle \rho \rangle \, y \, \langle S \rangle \, y' \, \langle \rho^{\cup} \rangle \, x') \\
\equiv & \\
 & \forall x, x' :: \quad \exists y, y' :: \quad x \, \langle R \rangle \, x' \Rightarrow x \, \langle \rho \rangle \, y \, \langle S \rangle \, y' \, \langle \rho^{\cup} \rangle \, x' \\
\equiv & \\
 & \forall R\text{-run } \vec{x} :: \quad \exists \vec{y} \ \rho\text{-related to } \vec{x} :: \quad \vec{y} \text{ is a } S\text{-run} \quad .
\end{array}
$$

Finally, for up and down simulation we find:

$$
\begin{array}{ll}
 & R \text{ up } \rho\text{-simulates } S \\
\equiv & \\
 & \forall x, x', y' : \ x \, \langle R \rangle \, x' \, \langle \rho \rangle \, y' : \quad \exists y :: \ x \, \langle \rho \rangle \, y \, \langle S \rangle \, y' \quad ,
\end{array}
$$

and:

$$
\begin{array}{ll}
 & R \text{ down } \rho\text{-simulates } S \\
\equiv & \\
 & \forall x, x,' y : \ y \, \langle \rho^{\cup} \rangle \, x \, \langle R \rangle \, x' : \quad \exists y' :: \ y \, \langle S \rangle \, y' \, \langle \rho^{\cup} \rangle \, x' \quad .
\end{array}
$$

Pictorially, the simulations have the following quantifications:

$$
\begin{array}{ccc}
\forall & \ldots & \forall \\
\downarrow & & \uparrow \\
\forall & \longrightarrow & \forall \\
 & \text{strong} &
\end{array}
\qquad
\begin{array}{ccc}
\exists & \ldots & \forall \\
\vdots & & \uparrow \\
\forall & \longrightarrow & \forall \\
 & \text{up} &
\end{array}
\qquad
\begin{array}{ccc}
\forall & \ldots & \exists \\
\downarrow & & \vdots \\
\forall & \longrightarrow & \forall \\
 & \text{down} &
\end{array}
\qquad
\begin{array}{ccc}
\exists & \ldots & \exists \\
\vdots & & \vdots \\
\forall & \longrightarrow & \forall \\
 & \text{weak} &
\end{array}
$$

2.4 Properties of simulation We wish to investigate the notion of simulation, in particular closure properties such as preservation under composition, and transitivity:

- Suppose R_i ρ-simulates S_i. Does the composite $R_0 ; R_1$ also ρ-simulate the composite $S_0 ; S_1$?

- Suppose R ρ-simulates S, and S σ-simulates T. Does R simulate T in any sense?

It is not hard to give theorems and direct proofs for each kind of simulation separately. But we shall follow another approach: we shall first generalise the notion of simulation, and then prove the claims in one go for all of the above notions. The additional advantage (and original motivation) is that the new notion of simulation is easily extended to a notion of simulation between *contexts*, or parameterised relations, rather than *stand alone* relations.

2.5 Generalised simulation. Using the notion of factors, we can bring each of the four notions of simulation in the form $R \subseteq F(S)$ for a suitable function F (which we call St, Up, Dn, Wk, respectively):

$$
\begin{array}{lcll}
R \text{ strongly } \rho\text{-simulates } S & \equiv & \rho^{\cup} ; R ; \rho \subseteq S & \\
 & \equiv & R \subseteq \rho^{\cup} \backslash S / \rho & \text{so } St_\rho(S) = \rho^{\cup} \backslash S / \rho \\
R \text{ up } \rho\text{-simulates } S & \equiv & R ; \rho \subseteq \rho ; S & \\
 & \equiv & R \subseteq (\rho ; S)/\rho & \text{so } Up_\rho(S) = (\rho ; S)/\rho \\
R \text{ down } \rho\text{-simulates } S & \equiv & \rho^{\cup} ; R \subseteq S ; \rho^{\cup} & \\
 & \equiv & R \subseteq \rho^{\cup} \backslash (S ; \rho^{\cup}) & \text{so } Dn_\rho(S) = \rho^{\cup} \backslash (S ; \rho^{\cup}) \\
R \text{ weakly } \rho\text{-simulates } S & \equiv & R \subseteq \rho ; S ; \rho^{\cup} & \text{so } Wk_\rho(S) = \rho ; S ; \rho^{\cup} \quad .
\end{array}
$$

Thus we will speak of F-simulation. We shall use juxtaposition to denote functional composition (of simulation functions): $F\,G(X) = F(G(X))$.

2.6 Definition. F-simulation is denoted by $\subseteq_F$ and defined by:

$$R \subseteq_F S \quad \equiv \quad R \subseteq F(S) \quad .$$

It remains to be investigated whether some constraints are to be imposed upon F in order that the notion of F-simulation has nice properties and is of practical use.

For the theorem below we also need the concept of subdistributivity.

2.7 Definition. Function F is called *subdistributive over relational composition* if:

$$F(R) ; F(S) \quad \subseteq \quad F(R ; S) \quad .$$

2.8 Theorem.
For monotone F (and arbitrary G):

$$R \subseteq_F S \subseteq_G T \quad \Rightarrow \quad R \subseteq_{FG} T \quad .$$

For subdistributive F:

$$R_i \subseteq_F S_i \;{\scriptstyle (i=0,1)} \quad \Rightarrow \quad R_0 ; R_1 \subseteq_F S_0 ; S_1 \quad .$$

The proofs are immediate by unfolding the definitions.

In order to apply the theorem to s/u/d/w-simulation, we need to investigate whether the s/d/u/w/-simulation functions are monotone and distributive, and what their compositions look like.

2.9 Theorem.
Each F_ρ of $St_\rho, Up_\rho, Dn_\rho, Wk_\rho$ is monotone (so all s/u/d/w-simulations are transitive) and satisfies $F_\rho\, F_\sigma = F_{\rho;\sigma}$.
Subdistributivity (hence preservation of s/u/d/w-simulation) holds in the following cases:

for St_ρ	if ρ is total
for Up_ρ, Dn_ρ	for all ρ
for Wk_ρ	if ρ is functional.

Proof. The monotonicity of the simulation functions is immediate since they are composed of factorisation (with a fixed denominator) and relational composition (with one fixed argument), and factorisation is monotone in the numerator and relational composition is monotone in both arguments.

That the simulation functions compose nicely, is shown for St as follows:

$$
\begin{array}{ll}
 & St_\rho\, St_\sigma(S) \\
= & \\
 & \rho^\cup\backslash(\sigma^\cup\backslash S/\sigma)/\rho \\
= & \\
 & (\sigma^\cup \,;\, \rho^\cup)\backslash S/(\rho \,;\, \sigma) \\
= & \\
 & (\rho \,;\, \sigma)^\cup\backslash S/(\rho \,;\, \sigma) \\
= & \\
 & St_{\rho;\sigma}(S) \quad .
\end{array}
$$

Similarly for Up, Dn, Wk .

Subdistributivity is slightly less easy to prove. For St_ρ and total ρ we argue:

$$
\begin{array}{ll}
 & St_\rho\,(R) \;;\; St_\rho(S) \\
= & \qquad \text{definition } St \\
 & \rho^\cup\backslash S/\rho \;;\; \rho^\cup\backslash S/\rho \\
\subseteq & \qquad \text{totality } \rho :\ I \subseteq \rho \,;\, \rho^\cup \\
 & \rho^\cup\backslash S/\rho \,;\, \rho \,;\, \rho^\cup \,;\; \rho^\cup\backslash S/\rho \\
\subseteq & \qquad \text{cancellation (twice, in the middle)} \\
 & \rho^\cup\backslash R \;;\; S/\rho \\
\subseteq & \qquad \text{fusion into numerator (twice, in the middle)} \\
 & \rho^\cup\backslash(R \,;\, S)/\rho \\
\subseteq & \qquad \text{definition } St \\
 & St_\rho(R \,;\, S) \quad .
\end{array}
$$

For Up_ρ and arbitrary ρ we argue:

$$\begin{array}{ll}
& Up_\rho(R) \,;\, Up_\rho(S) \\
= & \quad \text{definition } Up \\
& (\rho ; R)/\rho \,;\, (\rho ; S)/\rho \\
\subseteq & \quad \text{fusion into numerator} \\
& ((\rho ; R)/\rho \,;\, \rho ; S)/\rho \\
\subseteq & \quad \text{cancellation} \\
& (\rho ; R ; S)/\rho \\
= & \quad \text{definition } Up \\
& Up_\rho(R ; S) \quad .
\end{array}$$

The reasoning for Dn is similar. Finally, for Wk_ρ and functional ρ we argue:

$$\begin{array}{ll}
& Wk_\rho(R) ; Wk_\rho(S) \\
= & \quad \text{definition } Wk \\
& \rho ; R ; \rho^\cup \,;\, \rho ; S ; \rho^\cup \\
\subseteq & \quad \text{functionality } \rho :\ \rho^\cup ; \rho \subseteq I \\
& \rho ; R \,;\, S ; \rho^\cup \\
\subseteq & \quad \text{definition } Wk \\
& Wk_\rho(R ; S) \quad .
\end{array}$$

This completes the proof.

3 Compositionality

We are interested in compositionality: how properties (simulation properties, here) of the semantics of the constituents carry over to the semantics of the composite. Theorem 2.8 already asserts that simulation is preserved by combining constituents into one whole by means of relational composition. But this doesn't cover all ways to combine several program parts into a whole. In its full generality we need to consider *contexts*, that is, programs with a hole in it where other program parts (contexts maybe) can be plugged in. So a context A is nothing else than a function mapping relations to relations: $A(R)$ is a relation when R is. We shall use $A, B, C, \ldots$ for contexts.

> *Remark.* It is not to be expected that contexts are fully arbitrary functions. In particular, they must treat their argument as a black box, not looking at their inside. For example, for relations R with equal domain and range, the following function A is probably not what one would call a context:
>
> $$A(R) \quad = \quad \text{if } R \text{ is bijective, then } R^\cup \text{ else } R \quad .$$

Thus we expect contexts to be polymorphic in their parameter, and thus satisfying a so-called free theorem; see Wadler [5]. In a categorical setting, we expect that contexts should be defined as a kind of *transformations* in the sense of Fokkinga [1, 2]. One must be more explicit about the *typing* of relations in order to make these remarks precise. In the sequel we do not need these possible extra properties of contexts.

There is also a difference between contexts A and simulation functions F: it seems that the result type of $A(R)$ may differ much more from the argument type of R than is possible in the case of $F(R)$ and R. Again, in this paper we have no opportunity to exploit the difference.

A context A with another context B plugged into the hole is again a context: the nesting of B within A. Formally it is the functional composition of A with B, which we denote by juxtaposition:

$$\begin{array}{lcl} A\,B\,(X) & = & A(B(X)) \\ A\,F\,(X) & = & A(F(X)) \quad . \end{array}$$

Relational operations (composition, inclusion) are defined for contexts by pointwise extension:

$$\begin{array}{lcll} A \,;\, B & = & \lambda X :: & A(X) \,;\, B(X) \\ A \subseteq B & \equiv & \forall X :: & A(X) \subseteq B(X) \quad . \end{array}$$

We have already defined the notion of F-simulation for relations. Now we're going to define such a notion for contexts. For practical applicability we need *two* retrieve functions; F at the outside of the context, and G at the inside, that is, at the hole.

3.1 Definition. F,G-simulation is denoted $\subseteq_{F,G}$, and defined by:

$$\begin{array}{lcll} A \subseteq_{F,G} B & \equiv & A\,G \subseteq F\,B & \\ & \equiv & A(G(X)) \subseteq F(B(X)) & \text{for all } X \quad . \end{array}$$

Note the swap in the position of F,G. (In this way the notation seems consistent with the notation in my own work, and my thesis in particular.)

Of course the definition generalises in the obvious way to n-ary contexts. This gives no surprises (at least, if you think for a while...), neither in the definition nor in the theorems and proofs, but the explicit notation of the tuples of functions and arguments might be quite awkward. We'll postpone this to the appendix.

In the following theorem, the first two claims are the obvious generalisations of Theorem 2.8. The third claim is new; it asserts that separate simulations between contexts on the one hand and what is to be filled in on the other hand, carries over to a simulation between the composite systems.

3.2 Theorem.
For monotone F (and arbitrary G, H, J):

$$A \subseteq_{F,G} B \subseteq_{H,J} C \quad \Rightarrow \quad A \subseteq_{FH,GJ} B \quad .$$

For subdistributive F (and arbitrary G):

$$A_i \subseteq_{F,G} B_i \ {\scriptstyle (i=0,1)} \quad \Rightarrow \quad A_0 \,;\, A_1 \subseteq_{F,G} B_0 \,;\, B_1 \quad .$$

For monotone A (and arbitrary B, A', B'):

$$A \subseteq_{F,G} B, \quad A' \subseteq_{G,H} B' \quad \Rightarrow \quad A\,A' \subseteq_{F,H} B\,B' \quad .$$

Proof. Immediate by unfolding the definitions.

3.3 Research topic. *The main question raised by Zwiers et al is this: are there simple criteria for checking the conditions of the theorem in a practical situation, that is, for concrete programming and specification languages?*

References

[1] M.M. Fokkinga. Datatype laws without signatures. In *Computing Science in The Netherlands*, pages 231–248. Stichting Mathematisch Centrum, Amsterdam, 1991. Also Tech Report CS-R9133, CWI. Submitted for publication.

[2] M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[3] C.A.R. Hoare and J. He. The weakest prespecification. *Information Processing Letters*, 24:127–132, 1987.

[4] C.A.R. Hoare, J. He, and J.W. Sanders. Prespecification in data refinement. *Information Processing Letters*, 25:71–76, 1987.

[5] P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM Press, September 1989. FPCA '89, Imperial College, London.

[6] J. Zwiers, J. Coenen, and W.-P. de Roever. A note on compositional refinement. In *Proc. of the 5th Refinement Workshop, Springer Workshops in Computing*, pages 342–366. Springer Verlag, 1992.

A Contexts with multiple arguments

We generalise contexts with one hole to contexts with several holes. The problem that we want to solve is the awkward notation of explicit tuples of functions and arguments. So we set out to develop a notation that allows for a compact expression of the theorem, as well as a compact expression of the proof. We also want to express the theory "at the function level" rather than "at the point level" with explicit arguments.

Let us first give the notation for functions in general, not knowing that they are contexts or whatever.

A.1 Definition. The operations $\circ$ and $\bullet$ extend normal function composition in two different ways to composition of tuples of functions, m-tuple f and n-tuple g say:

$$\begin{array}{lcl} (f \circ g)(x) & = & (f_0(g_0(x), \ldots, g_{n-1}(x)),\ \ldots,\ f_{m-1}(g_0(x), \ldots, g_{n-1}(x))) \\ (f \bullet g)(x) & = & (f_0(g_0(x_0), \ldots, g_{n-1}(x_{n-1})),\ \ldots,\ f_{m-1}(g_0(x_0), \ldots, g_{n-1}(x_{n-1}))) \quad . \end{array}$$

In order that the definition is well formed, some arity restrictions have to be imposed on the tuples f and g. This is similar to simple typing constraints. We shall not deal with this aspect any further. Both $f \circ g$ and $f \bullet g$ can be considered as a tuple of functions:

$$\begin{array}{lcccl} (f \circ g)_i & = & f_i \circ g & = & \lambda x :: \ f_i(g_0(x), \ldots, g_{n-1}(x)) \\ (f \bullet g)_i & = & f_i \bullet g & = & \lambda x :: \ f_i(g_0(x_0), \ldots, g_{n-1}(x_{n-1})) \quad . \end{array}$$

The ordering $\subseteq$ on functions is extended to tuples of functions:

$$f \subseteq f' \quad \equiv \quad \forall i :: \quad f_i \subseteq f'_i \quad .$$

We shall identify a single function with the 1-tuple that has that function as sole member.

With these definitions there are a number of properties that facilitate a smooth calculation with tuples of functions (avoiding the need for explicit arguments).

A.2 Theorem. The identity function id is a right unit for $\circ$, operation $\circ$ is associative, and operations $\circ$ and $\bullet$, in that order, associate:

$$\begin{array}{lcl} f \circ id & = & f \\ f \circ (g \circ h) & = & (f \circ g) \circ h \\ f \circ (g \bullet h) & = & (f \circ g) \bullet h \quad . \end{array}$$

The tuple with components $f_i \circ g_i$ (for varying i) can be expressed compactly using the identity function id:

$$(id \bullet f) \circ g \quad = \quad (f_0 \circ g_0, \ldots, f_{m-1} \circ g_{m-1}) \quad .$$

Finally, m-ary function f is monotone iff for all m-tuples g, g':

$$g \subseteq g' \quad \Rightarrow \quad f \circ g \subseteq f \circ g' \quad .$$

Let us now generalise the notions of context and simulation to deal with contexts of arbitrary arity.

A.3 Definition. A context is an m-ary function mapping m-tuples of relations to a relation. Variables $A, B, \ldots$ range over contexts.

Nesting of an m-ary context A with an m-tuple of n-ary contexts A' is the n-ary context $A \circ A'$.

A simulation function is a unary function from relations to relations. For m-ary contexts A, B and simulation function F and m-tuple of simulation functions G, we put:

$$\begin{array}{lcl} A \subseteq_{F,G} B & \equiv & A \bullet G \subseteq F \circ B \\ & \equiv & \forall X :: A(G_0(X_0), \ldots, G_{m-1}(X_{m-1})) \subseteq F(B(X_0, \ldots, X_{m-1})) \quad . \end{array}$$

A.4 Theorem. (Compare with the last part of Theorem 3.2.)
For monotone A (and arbitrary B, A', B'):

$$A \subseteq_{F,G} B, \quad A' \subseteq_{G,H} B' \quad \Rightarrow \quad A \circ A' \subseteq_{F,H} B \circ B' \quad .$$

Here, A, B are m-ary contexts, A', B' are m-tuples of n-ary contexts, and F, G, H are a $1, m, n$-tuple, respectively, of simulation functions. Moreover, the second premiss has to be read as: $\forall i :: \quad A'_i \subseteq_{G_i,H} B'_i$.
Proof. The second premiss can be compactly formulated as: $A' \bullet H \subseteq (id \bullet G) \circ B'$. Now, using the association properties implicitly:

$$\begin{array}{cl} & A \circ A' \bullet H \\ \subseteq & \qquad \text{monotonicity of } A \text{, and the second premiss} \\ & A \circ (id \bullet G) \circ B' \\ = & \qquad \text{association } \circ\text{-}\bullet,\ id \text{ is right unit of } \circ \\ & (A \bullet G) \circ B' \\ \subseteq & \qquad \text{first premiss: } A \bullet G \subseteq F \circ B \text{ (“for all arguments”)} \\ & F \circ B \circ B' \quad . \end{array}$$

This is almost the same as the proof of the last part of Theorem 3.2.