Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 259 (2009) 165–176
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Data Refinement with Probability in Mind

T.M. Rabehaja[1,2] J.W. Sanders[1,3]

International Institute for Software Technology
United Nations University
Macao, SAR China

Abstract

The definition of data refinement between datatypes is expressed in terms of all programs that invoke procedures of the types. As a result it is laborious to check. Simulations provide sound conditions that, being 'static', facilitate checking; but then their soundness is important. In this paper we extract a technique from the heart of the theory and show it to be equivalent to data refinement; it plays a key role in establishing properties about simulations in any of the computational models. We survey the difficulties confronting the theory when the procedures and invoking programs may contain probabilistic choices, and show that then each of the two simulation conditions is alone not complete as a rule for data refinement, even if the datatypes are deterministic (in contrast to the standard case). The last part of the paper discusses work in progress.

Keywords: Data refinement, simulations, probability, incompleteness

1 Introduction

The results in this paper arose from an investigation into data refinement in the context of probabilistic programs. Here they have been grouped into results that do not concern probability and those that do.

1.1 Standard data refinement

For standard (*i.e.* non probabilistic) programs the relational model of Hoare *et al.* [6] consists of binary relations (between initial and final states) made 'healthy' to ensure that nontermination and nondeterminism interact so as to provide a sound development methodology. For example a nondeterministic choice, from a given

[1] The authors acknowledge assistance from the Macao Science and Technology Development Fund under the PEARL project, grant number 041/2007/A3.

[2] Email: tahiry@iist.unu.edu

[3] Email: jeff@iist.unu.edu

doi:10.1016/j.entcs.2009.12.023

initial state, between nontermination and termination in a given subsequent state is deemed equivalent to nontermination

$$\mathbf{abort} \sqcap P \;=\; \mathbf{abort}\,.$$

With such a methodological principle, error in the form of nontermination occurs only if it is the only option: the design calculus is 'pessimistic' and hence is reliable.

A simple semantic model is obtained by augmenting program states with a 'virtual' state $\perp$. Then the corresponding healthiness condition for the relational denotation $[P]_{\mathcal{R}}$ of computation P is, using infix notation for binary relations,

$$x\,[P]_{\mathcal{R}}\,\perp \;\Rightarrow\; \forall y \cdot x\,[P]_{\mathcal{R}}\,y\,.$$

(Although it is not our concern here, we recall that the setting for a systematic study of the relationship between laws and semantic healthiness conditions is 'Unifying Theories of Programming' [7].) Refinement $P \sqsubseteq Q$ between computations is modelled as containment of semantic relations: $[P]_{\mathcal{R}} \supseteq [Q]_{\mathcal{R}}$.

The more comprehensive predicate-transformer model of Dijkstra [1] contains the relational model via the Galois embedding wp: for relational denotation $[P]_{\mathcal{R}}$, postcondition q and initial state x,

$$wp.[P]_{\mathcal{R}}.q.x \;:=\; \forall y \cdot x\,[P]_{\mathcal{R}}\,y \;\Rightarrow\; y \neq \perp \wedge q.y\,.$$

Notice that termination is 'built in' to that (total correctness) definition which as a result does not need a virtual element. Refinement between computations is captured by the lifted implication ordering; writing $[P]_{\mathcal{T}}$ for the transformer denotation of computation P,

$$[P]_{\mathcal{T}} \sqsubseteq [Q]_{\mathcal{T}} \;:=\; \forall q, x \cdot [P]_{\mathcal{T}}.q.x \;\Rightarrow\; [Q]_{\mathcal{T}}.q.x\,.$$

Such refinement between computations suffices for the incremental derivation of implementations from specifications over the *same* state space. But to incorporate data representations, 'data refinement' is required. One datatype is *refined by* another iff use (by method invocation) of the former by any program P is refined by that program P' which instead uses the corresponding methods of the second type. Though that definition is theoretically simple, its verification requires an induction over all 'uses' of the type. So *simulation conditions* are used in practice [14].

In the relational model downwards and upwards simulations are readily shown to be sound methods for data refinement. The reason is that those definitions 'encapsulate' the induction required by the definition of data refinement. In Section 2 we study techniques that facilitate the proof of properties of data refinement by bridging the gap between the 'dynamic' definition of data refinement and the 'static' definition of simulations. A condition is extracted from the heart of the theory and shown to be equivalent to data refinement. It appears to play a key role in establishing properties of simulations in any of the computational models.

In the relational model, neither type of simulation by itself provides a complete rule for data refinement, although together they do so [5,14].

In the transformer model the two forms of simulation are again sound, but this time upwards simulation is by itself complete. The reason is that downwards simulation is able to be expressed by a Galois connection. That is the basic technique used here. The proof in the relational setting that the two simulations generate all data refinements reduces in the transformer setting to the stronger statement that a composition of upwards simulations (again an upwards simulation) does so.

1.2 Probabilistic data refinement

Probabilistic programs consist of standard programs augmented with a binary choice $P_r\oplus Q$ which chooses P with probability r and Q with the deficit probability $\overline{r} := 1-r$. In that setting, the counterpart of the relational model is the distributional model of He *et al.* [4] and that of the transformer model is the expectation-transformer model of Morgan *et al.* [12]. Again there is a Galois connection between them [9].

In the probabilistic setting, datatype procedures and invoking programs all may contain probabilistic choices. The definition of data refinement is similar but now simulation computations are in general probabilistic. For examples we refer to 'the steam boiler' in [9], Chapter 4 and to the proof of Theorem 3.

In Section 3 we consider data refinement for probabilistic programs. We begin by establishing structure on the spaces of simulations then show that in the distributional model, neither kind of simulation is alone complete for data refinement. Further progress is beset by the weakened laws of probabilistic programming. We conclude with a discussion.

The explicit contributions of the paper are:

(i) A characterisation of data refinement, that spans the divide between the 'static' definition (of simulation) and its 'dynamic' role in achieving data refinement and so captures an essential proof technique.

(ii) A summary of the structures, in the probabilistic model, of the spaces of upwards and downwards simulations. Those determine what can be said of data refinement in the probabilistic setting.

(iii) An example to show that in the distributional model of probabilistic programs, neither upwards nor downwards simulation is complete, *even if the abstract datatype is deterministic*. Approaches to the open question of completeness are discussed.

2 Data refinement

The topic of this section is standard (non-probabilistic) programs with the predicate-transformer model.

2.1 Preliminaries

The conjunctivity of standard programs x,

$$\forall y, z \cdot x \mathbin{;} (y \sqcap z) \;=\; x \mathbin{;} y \sqcap x \mathbin{;} z \tag{1}$$

is a remarkably strong property, supporting the definition of a Galois connection (provided x terminates) in the predicate-transformer model.

Lemma 1 *For any terminating computation x the function $y \mapsto x \mathbin{;} y$ is universally $(\sqsupseteq, \sqsupseteq)$-junctive and so possesses an adjoint $\alpha.x$ satisfying*

i. $\mathbf{skip} \sqsubseteq x \mathbin{;} \alpha.x$

ii. $\alpha.x \mathbin{;} x \sqsubseteq \mathbf{skip}$

iii. $\alpha.(x \mathbin{;} y) = \alpha.y \mathbin{;} \alpha.x$.

The proof is routine, using the fact that the adjoint of a universally $(\sqsupseteq, \sqsupseteq)$-junctive map is unique and the composition $sp \circ rp$ satisfies Lemma 1 where rp is the adjoint of wp and sp stands for the 'strongest post-condition' in the relational-transformer translation.

2.2 Data refinement

Recall that a datatype $T = (L; in, \mathcal{O}, fin)$ has type L of states, family $\mathcal{O}$ of named procedures (each, in general, with input and output), an initialisation in and a finalisation fin [5]. Each procedure $O : \mathcal{O}$ is assumed to be terminating. We let $\mathcal{L}$ denote the space of computations written in the guarded-command language, on some global state space which we leave implicit, that invoke procedures of type T.

Datatypes $T = (L; in, \mathcal{O}, fin)$ and $T' = (L'; in', \mathcal{O}', fin')$ are *conformal* iff the function $O \mapsto O'$ from $\mathcal{O}$ to $\mathcal{O}'$ is a signature-preserving bijection from $\mathcal{O}$ to $\mathcal{O}'$. If $P : \mathcal{L}$ invokes procedures from T, then $P' : \mathcal{L}'$ denotes the program that correspondingly invokes O' when P invokes O.

A datatype T is data-refined by a conformal type T' iff

$$\forall P \in \mathcal{L} \cdot in \mathbin{;} P \mathbin{;} fin \;\sqsubseteq\; in' \mathbin{;} P' \mathbin{;} fin' .$$

2.3 Data refinement characterised

In this section we use Lemma 1 to characterise data refinement in the predicate-transformer model, reasoning as follows. If P is a computation then $[P]$ denotes its semantic denotation, in this case a predicate transformer.

If $P, Q : \mathcal{L}$ are terminating uses of T then $P \mathbin{;} Q$ is also a terminating use of T for whose corresponding use $P' \mathbin{;} Q'$ of T' we have

$$[in \mathbin{;} P] \mathbin{;} [Q \mathbin{;} fin] \;\sqsubseteq\; [in' \mathbin{;} P'] \mathbin{;} [Q' \mathbin{;} fin']$$

$\Rrightarrow$ monotonicity of $;$ and Lemma 1.*ii.*

$$\alpha.[in' \mathbin{;} P'] \mathbin{;} [in \mathbin{;} P] \mathbin{;} [Q \mathbin{;} fin] \;\sqsubseteq\; [Q' \mathbin{;} fin']$$

$\Rrightarrow$ monotonicity of $;$ and Lemma 1.*i.*

$$\alpha.[in' \mathbin{;} P'] \mathbin{;} [in \mathbin{;} P] \;\sqsubseteq\; [Q' \mathbin{;} fin'] \mathbin{;} \alpha.[Q \mathbin{;} fin] .$$

The left and right hand sides of the last refinement are functions, respectively, of P and Q which we denote by I and F

$$I(P) := \alpha.[in' \mathbin{;} P'] \mathbin{;} [in \mathbin{;} P]$$

$$F(Q) := [Q' \mathbin{;} fin'] \mathbin{;} \alpha.[Q \mathbin{;} fin]$$

$$\forall P, Q : \mathcal{L} \cdot I(P) \ \sqsubseteq \ F(Q)\,.$$

Equivalently I and F satisfy, with maximum $\sqcup$ and minimum $\sqcap$ taken pointwise in $\mathcal{L}$,

$$\sqcup I \ \sqsubseteq \ \sqcap F\,. \tag{2}$$

Therefore, with each pair (T, T') of conformal datatypes is associated a pair (I, F) of functions. We now see that refinement (2) is in fact strong enough to characterise data refinement between the types.

Theorem 1 *For conformal datatypes T and T', refinement $T \sqsubseteq T'$ holds iff the pair (I, F) of functions satisfies (2).*

Proof. The forward implication has been established. For the converse, if $P : \mathcal{L}$ then

$$I(\mathbf{skip}) \ \sqsubseteq \ \sqcup I \ \sqsubseteq \ \sqcap F \ \sqsubseteq \ F(P)$$

and so we reason

$\alpha.[in'] \mathbin{;} [in] \ \sqsubseteq \ [P' \mathbin{;} fin'] \mathbin{;} \alpha.[P \mathbin{;} fin]$

$\Rrightarrow$ monotonicity of $;$ and Lemma 1.*ii*.

$\alpha.[in'] \mathbin{;} [in \mathbin{;} P \mathbin{;} fin] \ \sqsubseteq \ [P' \mathbin{;} fin']$

$\Rrightarrow$ monotonicity of $;$ and Lemma 1.*i*.

$[in \mathbin{;} P \mathbin{;} fin] \ \sqsubseteq \ [in' \mathbin{;} P' \mathbin{;} fin']$

which completes the proof. □

The following representation of (I, F) indicates the relationship between data refinement and simulations.

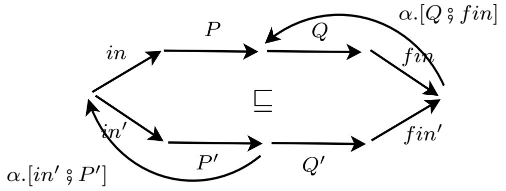

Fig. 1. Depiction of the pair (I, F) of functions characterising data refinement.

2.4 Simulations

Recall the definitions of simulations. An *upwards* simulation between T and T' is a *continuous* (nonhomogeneous and possibly nondeterministic) computation u with initial state of type L' and final state of type L such that

$$in \sqsubseteq in' \mathbin{;} u \tag{3}$$
$$\forall O : \mathcal{O} \cdot u \mathbin{;} O \sqsubseteq O' \mathbin{;} u \tag{4}$$
$$u \mathbin{;} fin \sqsubseteq fin' \,. \tag{5}$$

A *downward* (or *forward*) simulation is defined dually without the requirement of being continuous. The set of upwards (respectively downwards) simulations between T and T' is denoted $usim(T, T')$ (respectively $dsim(T, T')$). A *bisimulation* is an element of $bsim := usim \cap dsim$.

Upwards (and dually downwards) simulations are complete within the predicate transformer model [3,14]. In other words, the existence of a simulation is a necessary and sufficient condition for data-refinement. (By comparison, a combination of both kinds of simulation is required for completeness in the less comprehensive relational model.)

The pair (I, F) enjoys further properties. For instance $\sqcup I$ and $\sqcap F$ bound upwards simulation. For convenience we write $[\sqcup I, \sqcap F]$ for the set of functions satisfying refinement (2) (with no intention to imply that it is totally ordered).

Lemma 2 *If $u \in usim(T, T')$ then $u \in [\sqcup I, \sqcap F]$.*

Proof. For any $P : \mathcal{L}$

$u \in usim(T, T')$

$\Rrightarrow$

$[in \mathbin{;} P] \sqsubseteq [in' \mathbin{;} P'] \mathbin{;} u$

$\Rrightarrow$ monotonicity of $;$ and Lemma 1.*ii*.

$\alpha.[in' \mathbin{;} P'] \mathbin{;} [in \mathbin{;} P] \sqsubseteq u$

$\equiv$

$I(P) \sqsubseteq u\,,$

so $\sqcup I \sqsubseteq u$. The refinement $u \sqsubseteq \sqcap F$ is analogous. □

Theorem 1 in fact supplies the soundness of upwards simulations. Its converse is also of interest since then $usim(T, T') = [\sqcup I, \sqcap F]$ and so in particular completeness in the predicate-transformer model is a simple consequence, since the last set is empty when $\sqcup I \sqsubseteq \sqcap F$ does not hold. As we shall see, any such u satisfies the simulation laws (3) and (5) for initialisation and finalisation.

Lemma 3 *If $u \in [\sqcup I, \sqcap F]$ then equations (3) and (5) hold.*

Proof. For initialisation,

$[in'] \mathbin{;} u$

$\sqsupseteq$ $u \sqsupseteq \sqcup I$ and definition of I

$[in'] \mathbin{;} \sqcup\{\alpha.[in' \mathbin{;} P'] \mathbin{;} [in \mathbin{;} P] \mid P \in \mathcal{L}\}$

$\sqsupseteq$ **skip** $\in \mathcal{L}$ and associativity

$([in'] \mathbin{;} \alpha.[in']) \mathbin{;} [in]$

$\sqsupseteq$ Lemma 1.*i*.

$[in]$.

Similarly for finalisation,

$u \,;\, [fin]$

$\sqsubseteq$ $u \sqsubseteq \sqcap F$ and definition of F

$\sqcap\{[P' \,;\, fin'] \,;\, \alpha.[P \,;\, fin] \mid P \in \mathcal{L}\} \,;\, [fin]$

$\sqsubseteq$ **skip** $\in \mathcal{L}$ and associativity

$[fin'] \,;\, (\alpha.[fin] \,;\, [fin])$

$\sqsubseteq$ Lemma 1.*ii*.

$[fin']$

which completes the proof. □

The upper bound $\sqcap F$ has been shown [3] to be an upwards simulation. A similar proof shows $\sqcup I \in usim(T, T')$ (in particular, $\sqcup I$ is well defined and sound).

Theorem 2 *If $T \sqsubseteq T'$ then $\sqcup I \in usim(T, T')$.*

Proof. By Theorem 1, $\sqcup I$ is well defined and $\sqcup I \sqsubseteq \sqcap F$. Therefore $\sqcup I$ satisfies equations (3) and (5). It remains to verify (4), which includes soundness. But

$\sqcup I \,;\, [P]$

$=$ definition of I

$\sqcup\{\alpha.[in' \,;\, Q'] \,;\, [in \,;\, Q] \mid Q \in \mathcal{L}\} \,;\, [P]$

$=$ pointwise definition on transformers: $(\sqcup t).q := \sqcup(t.q)$

$\sqcup\{\alpha.[in' \,;\, Q'] \,;\, [in \,;\, Q] \,;\, [P] \mid Q \in \mathcal{L}\}$

$\sqsubseteq$ Lemma 1.*i*.

$\sqcup\{[P'] \,;\, \alpha.[P'] \,;\, \alpha.[in' \,;\, Q'] \,;\, [in \,;\, Q] \,;\, [P] \mid Q \in \mathcal{L}\}$

$=$ Lemma 1.*iii*.

$\sqcup\{[P'] \,;\, \alpha.[in' \,;\, Q' \,;\, P'] \,;\, [in \,;\, Q \,;\, P] \mid Q \in \mathcal{L}\}$

$=$ definition of I

$\sqcup\{[P'] \,;\, I(Q \,;\, P) \mid Q \in \mathcal{L}\}$

$\sqsubseteq$ $[P']$ is monotonic and $\sqcup\{[P'].b \mid b \in B\} \sqsubseteq [P']. \sqcup B$

$[P'] \,;\, \sqcup\{I(Q \,;\, P) \mid Q \in \mathcal{L}\}$

$\sqsubseteq$ $\{P \,;\, Q \mid P \in \mathcal{L}\} \subseteq \mathcal{L}$

$[P'] \,;\, \sqcup\{I(Q) \mid Q \in \mathcal{L}\}$

$=$

$[P'] \,;\, \sqcup I$.

The result follows by taking a procedure $O \in \mathcal{O}$ instead of P. On the other hand, using the expression of α as $sp \circ rp$, one shows that $\alpha.t$ is universally disjunctive (in particular $\sqsubseteq, \sqsubseteq$- continuous) and then $\alpha.t \,;\, t$ is also continuous for continuous

t. Therefore $\sqcup I$ is continuous since for any bounded, $\sqsubseteq$-directed family $(p_i)_i$ of predicates one shows $a = \sqcup I.(\sqcup_i p_i) = \sqcup_i(\sqcup I.p_i) = b$ by simply proving $a \le q \equiv b \le q$ for any predicate q. □

Does a similar result hold for downward simulation? The problem requires the existence of a dual adjoint map β satisfying

$$\textbf{skip} \ \sqsubseteq \ \beta.x \fatsemi x \quad \text{and} \quad x \fatsemi \beta.x \ \sqsubseteq \ \textbf{skip}$$

and so is a little delicate; it would imply that the pair (α, β) forms a Galois connection between the spaces $(\mathcal{T}, \sqsubseteq)$ and $(\mathcal{T}, \sqsupseteq)$. Thus it suffices to decide universal $(\sqsubseteq, \sqsupseteq)$-junctivity of α. This is a current work.

3 Probabilistic data refinement

In this section we consider probabilistic computations by enlarging $\mathcal{L}$ to include the operator ${}_p\oplus$ for probabilistic choice. We study the extent to which the previous approach to data refinement applies, concluding with a negative result that is of interest also because it differs from the corresponding negative result in the standard case.

3.1 Spaces of simulations

An attempt to replay the previous approach to data refinement with probabilistic computations immediately confronts failure of identity (1): left distributivity fails [9] and is replaced by

$$\forall x, y, z \cdot x \fatsemi (y \sqcap z) \ \sqsubseteq \ x \fatsemi y \sqcap x \fatsemi z \,. \tag{6}$$

That is interesting because probabilistic choice ${}_p\oplus$ does not even appear there explicitly. Inequality may be revealed, however, by considering an x that makes a fair choice (*i.e.* with ${}_{\frac{1}{2}}\oplus$) between 0 or 1 to some variable, letting y be assignment of 0 to a second variable and letting z be assignment of 1 to that second variable. Straightforward reasoning then shows that the greatest expectation of the two variables having the same value is on the left 0 and on the right $\frac{1}{2}$. Of course refinement in that law is to be expected by monotonicity, so that argument shows strictness.

We infer that nondeterministic choice is restricted so that it is unable to guess in advance the outcome of a probabilistic choice. Otherwise the right-hand side, with its immediate resolution of the nondeterministic choice, would be no more constrained than the left-hand side and equality would hold. At run time the resolution of nondeterminism is allowed to depend on only the computation history, which includes its present knowledge. Therefore, the map α is no longer well defined. In fact, it is defined only for *conjunctive* computations (*i.e.* those computations x satisfying (1)). That means the previous results characterising data refinement and bounding upwards simulations no longer hold, and other techniques are needed to study probabilistic data refinement.

We begin by estabilishing structure on the sets of simulations.

Lemma 4 *The space $usim(T, T')$ of upwards simulations between datatypes T and T' is stable under probabilistic choice ${}_p\oplus$. The space $dsim(T, T')$ of downwards simulations is stable under nondeterministic choice $\sqcap$.*

Proof. For the first claim, if $u, v : usim(T, T')$ then properties (3), (4) and (5) must be established for the probabilistic choice $u\ {}_p\oplus\ v$. We are allowed to use

$$x \mathbin{;} (y\ {}_p\oplus\ z) \ \sqsupseteq\ x \mathbin{;} y\ {}_p\oplus\ x \mathbin{;} z \tag{7}$$
$$(x\ {}_p\oplus\ y) \mathbin{;} z \ =\ x \mathbin{;} z\ {}_p\oplus\ y \mathbin{;} z \tag{8}$$

as well of course as other laws of probabilistic programming [9].

For $u, v \in usim(T, T')$, we reason as follows. For initialisation,

in

$=$ law $P \ = \ P\ {}_p\oplus\ P$

$in\ {}_p\oplus\ in$

$\sqsubseteq$ Law (3) since $u, v : usim(T, T')$

$in' \mathbin{;} u\ {}_p\oplus\ in' \mathbin{;} v$

$\sqsubseteq$ Law (7)

$in' \mathbin{;} (u\ {}_p\oplus\ v)\,.$

For invocation,

$(u\ {}_p\oplus\ v) \mathbin{;} O$

$=$ Law (8)

$u \mathbin{;} O\ {}_p\oplus\ v \mathbin{;} O$

$\sqsubseteq$ Law (4) since $u, v : usim(T, T')$

$O' \mathbin{;} u\ {}_p\oplus\ O' \mathbin{;} v$

$\sqsubseteq$ Law (7)

$O' \mathbin{;} (u\ {}_p\oplus\ v)\,.$

For finalisation,

$(u\ {}_p\oplus\ v) \mathbin{;} fin$

$=$ Law (8)

$u \mathbin{;} fin\ {}_p\oplus\ v \mathbin{;} fin$

$\sqsubseteq$ Law (5) since $u, v : usim(T, T')$

$fin' \mathbin{;} u\ {}_p\oplus\ fin' \mathbin{;} v$

$\sqsubseteq$ Law (7)

$fin' \mathbin{;} (u\ {}_p\oplus\ v)\,.$

The second claim, for downwards simulation, is similar using instead left subdistributivity. □

The methodology of incremental refinement relies on transitivity of simulations. Lifting sequential composition $\mathbin{;}$ to sets of computations, we have the following result.

Lemma 5 *For datatypes T, T' and T'',*

$$usim(T', T'') \mathbin{⨾} usim(T, T') \;\subseteq\; usim(T, T'')$$

and similarly for dsim.

Proof. The proof is based on calculations that are by now familiar. For u : $usim(T, T')$ and v : $usim(T', T'')$, use of associativity and monotonicity of ⨾ establishes the three required conditions:

initialisation: $in \;\sqsubseteq\; in' \mathbin{⨾} u \;\sqsubseteq\; in'' \mathbin{⨾} v \mathbin{⨾} u$

invocation: $v \mathbin{⨾} u \mathbin{⨾} O \;\sqsubseteq\; v \mathbin{⨾} O' \mathbin{⨾} u \;\sqsubseteq\; O'' \mathbin{⨾} v \mathbin{⨾} u$

finalisation: $v \mathbin{⨾} u \mathbin{⨾} fin \;\sqsubseteq\; v \mathbin{⨾} fin' \;\sqsubseteq\; fin''$.

Dual reasoning establishes the analagous result for *dsim*. □

3.2 Incompleteness

Just as neither simulation is alone complete for data refinement in the standard relational model, so in the distributional model of probabilistic programs. However we have a stronger result, applying even if the datatypes are deterministic.

Theorem 3 *Neither upwards nor downwards simulation is alone complete for data refinement in the distributional model of probabilistic programs, even when restricted to deterministic types.*

Proof. Consider the 'fork and spade' example of Figure 2. Indeed (writing δ_x for the point mass as x) the computation

$$d = \{(0, \delta_a), (1, \tfrac{1}{2}\delta_b + \tfrac{1}{2}\delta_c), (2, \delta_d), (3, \delta_e)\}$$

is a sound downward simulation [9] (the simulation does not modify the global state), so $T \sqsubseteq T'$. But there is no upwards simulation from T to T' although T and T' are both deterministic.

In fact, if u is one such upwards simulation, necessarily $u.b = \delta_1$ but $u \mathbin{⨾} O.b = \{\frac{1}{2}\delta_3 + \frac{1}{2}\delta_4\}$ and $O' \mathbin{⨾} u.b = \{\delta_b\} \not\subseteq \{\frac{1}{2}\delta_b + \frac{1}{2}\delta_c\}$. □

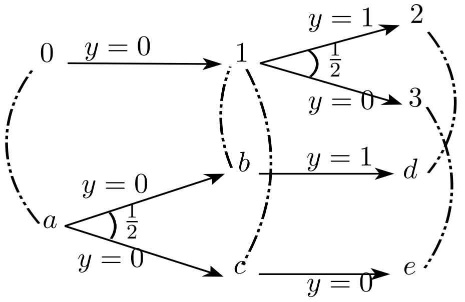

Fig. 2. The *fork and spade* example for the proof of Theorem 3, in which the usual nondeterministic choice from state a is replaced by a fair choice.

4 Discussion

The difficulty of data refinement in the probabilistic setting lies with the resolution of non-deterministic choice. That poses problems for both soundness and completeness.

For soundness of upwards simulation we need to establish that if (4) holds for two procedures it also holds for their nondeterministic choice. But

$$\begin{array}{ll} u \mathbin{;} (A \sqcap B) \sqsubseteq (A' \sqcap B') \mathbin{;} u & \\ = & \text{law of probabilistic programs} \\ u \mathbin{;} (A \sqcap B) \sqsubseteq (A' \mathbin{;} u) \sqcap (B' \mathbin{;} u) & \end{array}$$

at which point we are stuck because Law (6) does not support the distribution we seek on the left. More must be assumed about the interrelationship between u and the programs A and B. Recalling the example at the start of Section 3.1, presumably the meaning of the data invocation ought not to depend on where in the program text the datatype is defined. So either we assume that the simulation contains no probabilistic choice (not very helpful), or further assumptions must be imposed on the datatype's semantics, providing tighter control of interference between distinct variables that is possible with the usual current models [8].

For completeness, recall that in the standard case, if T is deterministic (*i.e.* its procedures are all deterministic) then data refinement of T by T' yields construction of a simulation (constructed, in fact, using the inductive method encapsulated in Theorem 1). That suggested [5] interpolation of a deterministic datatype U bisimilar to T but in which nondeterminism of T is removed entirely. For that reason datatype U is called the power-set type. In that case there is an upwards simulation (in fact bisimulation) from T to U. But since U is deterministic and U is data refined by T', there is a downwards simulation from U to T'. And so in sequence the two simulations are complete [5,14].

That result depended on the completeness of downward simulation for deterministic datatypes. In the presence of probability, 'deterministic' again means maximality in the refinement ordering. But now the 'fork and spade' example has shown that each simulation is alone incomplete, *even for deterministic probabilistic programs.*

In proposing a complete technique for data refinement in the probabilistic setting, much remains to be done. What might play the role of the power-set construction in the probabilistic case? Theorem 3 shows that, unlike the standard case, it does not suffice to consider deterministic datatypes. Moreover the result is crucially dependent on the semantic model of computations and even on the manner in which the semantics of a module is cast. The former is inherited from the standard case but the latter is new.

Constructions that combine probability and nondeterminism include the following. Varacca and Winskel [16] show that there is no idempotent and distributive operator over a power set construction; in their proof non-deterministic choice is represented by 'straight' set union rather than convex closure of the union. The constructions of Tix *et al.* [15] and of Mislove *et al.* [10,11] provide domains combining probability with nondeterminism. Finally, to impose a systematic relationship on the semantic models, the span construction of Clare Jones *et al.* [2] has been considered for probabilistic models.

This is an area of continuing work.

References

[1] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall International, 1974.

[2] P. H. B. Gardiner, C. E. Martin and O. de Moor. An algebraic construction of predicate transformers. In *Mathematics of Program Construction*, *LNCS*, **669**:100-121, Springer-Verlag, 1993.

[3] P. H. B. Gardiner and Carroll Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, **5**(4):367–382, 1993.

[4] J. He and K. Seidel and A.K. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, **28**(2):171–192, 1997.

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

[6] C. A. R. Hoare, I. J. Hayes, Jifeng He, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sørensen, J. M. Spivey and B. A. Sufrin. The laws of programming. *Communications of the ACM*, **30**(8):672–686, 1987.

[7] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming.* Prentice Hall, 1998.

[8] A. K. McIver and C. C. Morgan. A probabilistic approach to information hiding. In A. K. McIver and C. C. Morgan, editors, *Programming Methodology*, Monographs in Computer Science, Springer Verlag, 2003.

[9] A. K. McIver and C. C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer Verlag, 2005.

[10] M. Mislove, J. Ouaknine and J. Worrell. Axioms for probability and nondeterminism. *ENTCS*, **96**:7–28, 2004.

[11] M. W. Mislove. On combining probability and nondeterminism. *ENTCS*, **162**:261–265, 2006.

[12] C. C. Morgan, A. K. McIver and K. Seidel. Probabilistic Predicate Transformers. *ACM TOPLAS*, **18**(3):325–353, 1996.

[13] T. M. Rabehaja and J. W. Sanders. Refinement algebra with explicit probabilism. In *The 3rd IEEE Theoretical Aspects of Software Engineering Conference*, Tianjin, China, 2009.

[14] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1998.

[15] R. Tix, K. Keimel and G. Plotkin. Semantic domains for combining probability and non-determinism. *ENTCS*, **222**(1571-0661):3–99, Elsevier, 2009.

[16] D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, **16**(1):87–113, 2006.