A notion of refinement for automata

N. Sabadini, S. Vigna
Dipartimento di Scienze dell'Informazione,
Università di Milano, Via Comelico 39/41, I-20135 Milano MI, Italy
e_mail: sabadini@imiucca.csi.unimi.it, vigna@ghost.dsi.unimi.it

R.F.C. Walters
School of Mathematics and Statistics, University of Sydney,
N.S.W. 2006, Australia; e_mail: walters_b@maths.su.oz.au

1 Introduction

The notion of *refinement* of concurrent and distributed systems is a crucial one in any concurrency model. There are (at least) two distinct notions under the name of refinement, namely *refinement of specifications* and *refinement of machines* (automata). It is important to keep these two concepts distinct since their properties, and their mathematical formulation, are, or should be, quite different. A specification describes a whole class of machines and refinement here yields a smaller class of models, whereas a refinement of a machine yields a new, more complex, machine. It is not always clear which of these two notions is under consideration, because sometimes a mixture of the two approaches is appropriate and even because some formalisms allow different interpretations [AL87], [DGR], [BGV91].

This paper is concerned with the second notion, refinement of machines, where we refine both actions and the state space, in the context of the model of concurrency based on distributive automata, introduced by Sabadini and Walters in [SW93]. We leave to a later paper the discussion of refinement of specification in this context.

Action refinement [BGV91], [CvGG] is motivated by top-down design, and hence it is considered desirable that the refinement be performed in such a disciplined way that the behaviour and properties of a refined system are deducible from the unrefined one. For example, it is claimed that a semantic equivalence should be a congruence with respect to refinement.

However, there is another, more objective, notion of refinement, which arises from the need of a deeper analysis of a system, in which further information is introduced. Our definition of refinement captures precisely this idea. For example, on deeper analysis atomic actions may be seen to be non-atomic, actions which are conflicting for resources may be seen to be conflicting only part of the time, and actions which are parallel may be seen to be in fact not

parallel. Notice again a problem with names, since the word conflict which we use in relation to resources, is often used for the idea of a choice, in a specification, between different possible behaviours. Ideas similar to ours occur in [God93], [Zwi91].

With our definition, the refined system may have a much richer structure than the unrefined one. Thus we can study such issues as efficiency in time and resources. This implies that at each stage of refinement it is necessary to prove again, if possible, that desired properties of the system are preserved. This is not unreasonable, because the desired properties are properties of the *final* object of these refinement process, and at an earlier stage it may happen that it is not even possible to define them. This approach is advocated, for instance, by Chandy and Misra in [CM88]. Further, in contrast to the requirement that a semantic equivalence should be a congruence with respect to refinement, we expect that two systems which become equivalent on refinement should in fact be equivalent.

The notion of refinement is an integral part of any model of concurrency, one reason being that concurrency is concerned with the delay independent behaviour of systems. To introduce a delay into an atomic component (action) it is necessary to refine it. The resulting system should also be a refinement, in the sense that the set of its behaviours, restricted to the old states, should be exactly the same of those of the unrefined system, without delays. We prove in this paper that distributive automata are delay-insensitive in this precise sense. Note also that the encapsulation mechanism central to our model is just the replacement of a refined machine by an unrefined one, hence making a non atomic action atomic. It is also through the refinement notion that we are able to avoid non deterministic machines—the uncertainty of the occurence of an action is analysed in terms of refinements instead of non-determinism, as we will discuss later.

Formally, our definition is based on considering the automata as categories of transitions, and then a morphism is a functor between transition categories, and a refinement is an embedding of the category of one automaton in another one. The elementary categorical concepts used in this paper may be found in Walters [Wal].

This work has been supported by the Australian Research Council, Esprit BRA ASMICS, Italian MURST 40%, and the Italian CNR.

2 Distributive Automata

In [SW93] a model of distributed systems, *distributive automata*, was given in terms of a calculus for structured deterministic automata. Distributive automata are automata constructed from a given finite family of sets and function (data types and data operations) using the operations of a distributive category. Both the alphabet and state space of a distributive automaton are formed by the operations of sum and product from some given basic sets. The actions of a distributive automaton are formed from basic functions by composition, sum,

and product of functions, projections, injections, the diagonal and codiagonal, and the distributive isomorphism $X \times (Y + Z) \cong X \times Y + X \times Z$. Thus, the alphabet and state space have a rich structure reflecting parallel or conflicting, synchronous or asynchronous actions. There is one further operation. A distributive automaton whose alphabet is one letter and whose state space is of the form $X + U + Y$ may compute by iteration a (total) function from X to Y; such automata we call pseudofunctions. In the construction of distributive automata we may use the function computed by a pseudofunction. This operation allows hiding of state, encapsulation of iteration, and making atomic a non-atomic action. The notion of pseudofunction has a precursor in Elgot's iteration theories [Elg75] and Heller's work on recursion categories [Hel90]. A similar definition can also be found in [Knu73], and [Mil71].

Remarks on the model

One way to view our model is that our machines are finitary hierchical nets of deterministic automata, where, as we have mentioned above, non deterministic features are dealt with by requiring that constructions be delay insensitive.

The usual reasons for taking a non deterministic model for concurrency are that full knowledge of states is too complicated to deal with, and there is uncertainity in the timing of actions. Again, notice a problem with names: the fact that there is a lack of knowlege about which external actions occur is sometimes called non determinism, but it is a sort of "non determinism" already existing in finite state deterministic automata.

Moreover, once properly organized, states are not too complicated, and in any case there are far more possible behaviours of a system than states.

Another reason for the simplicity of our model is that, for example, there is no special mechanism for synchronizing processors. However there *is* a synchronization mechanism, which is purely mathematical in nature, namely the product of functions. The fundamental synchronization assumption is that atomic actions occur completely in parallel, or completely separated. The function $f \times g$ is the synchronized parallel product of f and g. For two actions to overlap non-trivially means that at least one is not atomic.

3 Refinement of automata

Definition 3.1 *Suppose M is a monoid and* $\mathbf{X}$ *an M-automaton; that is, a set X together with an action of M on X, $M \times X \to X : (m, x) \mapsto m \cdot x$; the action is required to satisfy the usual axioms $m_1 \cdot (m_2 \cdot x) = (m_1 m_2) \cdot x$ and $1 \cdot x = x$. Define the category Trans*$(\mathbf{X})$ *(the* transition category of X*) as follows:*

1. *objects are states (that is, elements) of X;*
2. *arrows from x to y are state transitions; that is, elements $m \in M$ such that $m \cdot x = y$;*
3. *composition is monoid multiplication.*

A morphism of automata, or, in short, a mapping *from* $\mathbf{X}$ *to* $\mathbf{Y}$*, is a functor from* $Trans(\mathbf{X})$ *to* $Trans(\mathbf{Y})$*, where* $\mathbf{Y}$ *is an* N*-automaton for a monoid* N*.*

An abstraction *from* $\mathbf{X}$ *to* $\mathbf{Y}$ *is a functor* $Trans(\mathbf{X})$ *to* $Trans(\mathbf{Y})$ *which is surjective on objects and arrows.*

A refinement *of* $\mathbf{X}$ *in* $\mathbf{Y}$ *is an inclusion, as a full subcategory, of* $Trans(\mathbf{X})$ *in* $Trans(\mathbf{Y})$*.*

In other words, in order to give a refinement of $\mathbf{X}$ one has to specify a bigger system $\mathbf{Y}$ which has a restriction to a system isomorphic to $\mathbf{X}$.

Notice that each arrow in $Trans(\mathbf{X})$ is determined by an element $m \in M$ *and* a domain and a codomain $x, y \in X$. Hence many distinct arrows will be labelled with the same element of the monoid.

In what follows, we will be concerned with free monoids on the structured alphabets we discussed. If $M = A^*$ and $N = B^*$, a functor F from $Trans(\mathbf{X})$ to $Trans(\mathbf{Y})$ is given by a function $F_{state} : X \to Y$ and a function $F_{action} : X \times A \to B^*$ satisfying the condition that if $\alpha \in A$ and $\alpha : x \to x'$ in $Trans(\mathbf{X})$, then $F_{action}(\alpha) : F_{state}(x) \to F_{state}(y)$ in $Trans(\mathbf{Y})$ (from now on, we will omit the subscripts when understood). For a refinement there is the further requirement that the function induced by F between $\mathbf{Hom}(x, x')$ and $\mathbf{Hom}(F(x), F(x'))$ is a bijection, and that the function between the state spaces in injective. (Morphisms of distributive automata should be defined by functions $X \to Y$ constructed using the operations of a distributive category, and by functions $A \times X \to B^*$ constructed using the operations of a countably extensive category with products [KWW], but this requirement is not necessary for the purposes of the present paper.)

Notice that the usual notion of substitution in language theory is a morphism which assigns to a letter a word or a language, but the latter ones are fixed once for all, and not dependent on state. Note also that not all full subcategories of $\mathbf{Y}$ induce a refinement. We can give also the following, weaker

Definition 3.2 *An* expansive mapping *is an inclusion* F *of* $Trans(\mathbf{X})$ *in* $Trans(\mathbf{Y})$ *such that whenever* $F(x \xrightarrow{\alpha} x') = F(x) \xrightarrow{s} F(x')$*, where* $\alpha \in A$ *and* $s \in B^*$*, then there are no* $x'' \in X$*,* $s' \in B^*$ *such that* s' *is a proper prefix of* s *and* $F(x) \xrightarrow{s'} F(x'')$*.*

When an atomic action is refined by an expansive mapping, the set of states spawned by the string it is mapped to lies entirely outside of the image of $\mathbf{X}$, except for the initial and final states (which are the image of the domain and of the codomain of the atomic action). We can indeed restate Definition 3.2 as follows:

1. $Y = X + U$ for some set U;
2. if $F(x, a) = b_1 \cdots b_n \in B^*$, then $b_1 \cdots b_k \cdot x \in U$ for $k = 1, 2, \ldots, n-1$.

Expansiveness and fullness are related by the following proposition:

Proposition 3.1 *Let* $\mathbf{X}$ *and* $\mathbf{Y}$ *be* A^* *and* B^* *automata, respectively. If a mapping* $F : \mathbf{X} \to \mathbf{Y}$ *is a refinement, then it is expansive.*

Proof. Suppose there are x'' and s' as in Definition 3.2. Then s factors as $s's''$, and $F(x'') \xrightarrow{s''} F(x')$. But because of faithfulness and fullness, there has to exist strings $t', t'' \in A^*$ such that $x \xrightarrow{t'} x''$ and $x'' \xrightarrow{t''} x'$. By composition, we get $t't'' = \alpha$. Thus, either $t' = \alpha$ and $t'' = \epsilon$, or $t' = \epsilon$ and $t'' = \alpha$. In both cases, s' is not a proper prefix of s.

This proposition cannot be reversed. Take $A = B = \{\alpha, \beta\}$, $X = \{*\}$ and $Y = \{0, 1\}$. Let the action of α be the identity on Y, and the action of β be $n \mapsto 1 - n$. The mapping sending the unique state of X into 0, α to α and β to $\beta\beta$ is expansive, but not full.

There is however a relevant case in which we can reverse Proposition 3.1:

Proposition 3.2 *Let* $\mathbf{X}$ *and* $\mathbf{Y}$ *be* A^* *and* B^* *automata, with* $A = B = \{\tau\}$*. If a mapping* $F : \mathbf{X} \to \mathbf{Y}$ *is expansive, then it is a refinement.*

Proof. If F is not a refinement, consider states $x, y \in X$ and an arrow $F(x) \xrightarrow{\tau^k} F(y)$ which is not image of an arrow from x to y. Assume without loss of generality that k is minimal. Let $F(x) \xrightarrow{\tau^n} F(z)$ be the image through F of $x \xrightarrow{\tau} z$. If $k > n$, then necessarily $F(z) \xrightarrow{\tau^{k-n}} F(y)$ is not in the image of $\mathbf{X}$, which contradicts the minimality of k. Then $n > k$. But this contradicts expansiveness.

Example 3.1 *When* $M = N = \{\tau\}^*$ *then refinement takes a particularly simple form. Such an automaton can be analyzed by considering the* orbits*, that is, sequences of states produced by the action of* τ *starting from a given initial state. A* refinement *of an automaton* $\mathbf{X}$ *is another automaton* $\mathbf{Y}$ *with state space of the form* $Y = X + U$ *such that the orbits of* $\mathbf{Y}$*, when restricted to* $\mathbf{X}$*, correspond exactly to orbits of* $\mathbf{X}$*.*

Remark 3.1 *It is clear that refinements form a category* **Refine***, and that abstractions form a category* **Abstract***. However, both refinement and abstraction can be looked at in the opposite direction, i.e., the domain of a refinement can be seen as a system in which space and time have been hidden, while the domain of an abstraction can be seen as a system with finer state space and actions (this is closely related to [Lyn87]). Formally, this correspond to the study of the categories* $\mathbf{Refine}^{op}$ *and* $\mathbf{Abstract}^{op}$*.*

The notion of transition category induces a notion of *behaviour* which is state dependent: for each pair of states x, y we can build the set of arrows between x and y, i.e., the *hom-set* between the objects x and y. Formally,

Definition 3.3 *The functor* $\mathbf{Behaviour} : \mathbf{Refine} \to \mathbf{Cat/Sets}$ *is defined by*

$$\mathbf{X} \mapsto \mathbf{Hom} : \mathit{Trans}(\mathbf{X})^{op} \times \mathit{Trans}(\mathbf{X}) \to \mathbf{Sets}$$

on objects, and by $F \mapsto F^{op} \times F$ *on morphisms.*

Note that $F^{op} \times F$ commutes with **Hom** up to isomorphisms exactly because F is a refinement. Note also that $F^{op} \times F$ is a morphism in **Cat/Sets**; this expresses the fact that the behaviour of $\mathbf{X}$ is a restriction of the behaviour of $\mathbf{Y}$ along the refinement.

4 Examples

4.1 Pseudofunctions

Consider a pseudofunction $\phi : X + U + Y \to X + U + Y$ which calculates the function $f : X \to Y$ by iteration, and suppose that $\phi(x) \in U + Y$ $(x \in X)$. Then from f we may obtain an automaton with a one-letter alphabet and state space $X + Y$ which calculates f in one step. Then the obvious inclusion $X + Y \to X + U + Y$ is a refinement. So, a pseudofunction is just the refinement of a function.

4.2 An IMP(G) Interpreter

The interpreter of **IMP**(G) programs described in [KW] is easily seen to be a refinement of each particular **IMP**(G) program. This was in fact the specification of the interpreter.

4.3 Mutual exclusion

Other theories of refinements often require that all the steps in the refinements of two conflicting action (systems) are conflicting. This seems to be reasonable when the word "conflict" means "irrevocable choice", but not when, as usual in applications, conflict comes from access to a common resource (in our setting, this means that two letters use the same part of the state space). Here, we can easily model the situation where the conflict may occur at only one step in the refinement.

4.4 Independent actions are not necessarily parallel

If we consider a fixed machine $\mathbf{Y}$, we may think of refinements of other machines $\mathbf{X}$ in this fixed machine $\mathbf{Y}$ as implementations of abstract machines $\mathbf{X}$ in a system $\mathbf{Y}$. The class of abstract machines has unbounded parallelism—there is no obstruction to considering arbitrary products of abstract machines. However in relation to $\mathbf{Y}$ it is possible to consider questions of resources. We can make the distinction between actions of $\mathbf{X}$ being "independent" and being "parallel". Actions are independent if they are parallel in the abstract machine $\mathbf{X}$. Actions are parallel if they are parallel in the system $\mathbf{Y}$.

The following example can be expressed by saying that independent actions in an abstract machine may not be parallel in the implementation.

Given two automata $\mathbf{X}$, $\mathbf{Y}$, both with alphabet $\{\tau\}$, suppose that there are refinements of $\mathbf{X}$ to $\mathbf{X}'$ and $\mathbf{Y}$ to $\mathbf{Y}'$, where $X' = X + U$ and $Y' = Y + U$, the

meaning being that the set U is the state space of some temporarily used (and reset after use) resource like a scratch pad, or printer. Then the synchronous parallel product $\mathbf{X} \times \mathbf{Y}$ of $\mathbf{X}$ and $\mathbf{Y}$ may be refined to an automaton in which there is only one resource U whose use is scheduled between $\mathbf{X}$ and $\mathbf{Y}$. The state space would be $Z = XY + UY + XY + XU$. The two letters α, β would act respectively on the first two and on the last two summands, applying $\langle \tau, \mathbf{1} \rangle$ and $\langle \mathbf{1}, \tau \rangle$, respectively (a trap state would complete the automaton in the obvious way). The injection of XY as first summand of Z would then define a refinement, which would schedule the parallel action $\langle \tau, \tau \rangle$ to $\alpha\beta$.

4.5 Shutdown

Consider a refined description of a system in which a new, destructive action can happen. This is a typical case of a sudden shutdown. We expect that the system can, at any time, be shut down, thus moving into a state which was impossible to reach before. In this case, the refinement space is formed by adding a single element, and a new letter to the alphabet; it sends to the new state any other state. The behaviour of the machine, if we ignore the shutdown state, is unmodified, which is exactly reflected in our definition of refinement.

4.6 Choice

Our refinement being a functor, it assigns to each action of the unrefined system a precise refinement. Hence it is not possible in our model to replace an action by two alternative actions even if two alternative actions may exist in the refined machine (such a thing would correspond to *two* different refinements). This accords with our view that machines, even asynchronous ones, are deterministic; the introduction of a choice in refinement is a non-determinism at the level of morphism. However, different choices can be identified by an abstraction morphism.

5 The Delay-Insensitivity of Distributive Automata

Let us consider a set S of basic functions. We will write $D(S)$ for the set of functions derived from S using the operations of a distributive category (for a detailed definition, see [SW93]). Consider also a set of functions T such that any function in S admits a refinement to a pseudofunction in $D(T)$.

The Delay-Insensitivity Theorem states that any S-automaton can be refined to a T-automaton. In other words, if each basic function hides the space and time of a complex computation which is expressed in terms of more elementary functions, we can refine an automaton built using the basic functions to another automaton built using the latter, more elementary functions.

The basis for this construction is given by Example 4.1, and by the properties of the **call**[] operator.

Theorem 5.1 *Let* $\mathbf{X}$ *be an automaton on the alphabet* A *with state space* X*, and actions taken from* $D(S)$*. Then there is an automaton* $\mathbf{Y}$ *on the alphabet* A *and actions taken from* $D(T)$ *such that* $\mathbf{Y}$ *is a refinement of* $\mathbf{X}$*.*

Proof. For each $a \in A$, the function $f_a : X \to X$ admits a refinement into an automaton $\phi_a : X + U_a + X \to X + U_a + X$. We now define

$$Y = I + X + \sum_{a \in A} U_a.$$

The action of $a \neq a'$ on $U_{a'}$ is the unique function $U_{a'} \to I$, while on X and on U_a is defined as ϕ_a. It is then clear that the inclusion $X \to Y$ induces a refinement.

6 Comparisons

As we remarked in the introduction, our notion of refinement differs markedly from notions currently being considered in Petri nets and process algebra; rather, it is in the spirit of [CM88, AL87, Zwi91]. The definition which is conceptually closest to our approach is the broader definition of Petri net morphism given in [MM90], where a single Petri net transition can be mapped to an entire computation, possibly composed by many parallel steps. However, due to the freedom with respect to the monoidal product, the mapping is not dependent on the global state of the net.

In contrast to the situation in action refinement ([CvGG],[DGR]), in our model it is not at all necessary that a refinement of two parallel processors be parallel (§4.4) (and hence we can discuss scheduling of resources), or a refinement of conflicting processors be conflicting in all steps (§4.3) (and hence we can discuss refinements which limit non-parallelism to exactly those points where common resources are needed). In contrast to Petri nets refinement ([BGV91]), we are unable to introduce a choice (§4.6) between actions to refine an action. This limitation simplifies considerably the theory without restricting its expressiveness.

References

[AL87] M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems*, number 430 in LNCS, pages 1–41, 1987.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[CvGG] I. Czaja, R. von Glabbeek, and U. Golz. Interleaving semantics and action refinement with atomic choice. Preprint.

[DGR] P. Degano, R. Gorrieri, and G. Rosolini. A categorical view of process refinement. In *Semantics: Foundations and Applications*, number 666 in LNCS.

[Elg75] C. Elgot. Monadic computation and iterative algebraic theories. *Studies in Logic and the Foundations of Mathematics*, 80:175–230, 1975.

[God93] P. Godefroid, D. Pirottin. Refining dependencies improves partial order verification methods. In C. Courcoubetis, editor, *CAV 93*, number 697 in LNCS, 1993.

[Hel90] A. Heller. An existence theorem for recursion categories. *Journal of Symbolic Logic*, 55(3):1252–1268, 1990.

[Zwi91] W. Janssen, M. Poel, J. Zwiers. Action systems and action refinement in the development of parallel systems. In J.C.M. Baeten, J.F. Groote, editors, *CONCUR 91*, number 527 in LNCS, 1991.

[Knu73] D.E. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1973.

[KW] W. Khalil and R.F.C. Walters. An imperative language based on distributive categories II. *RAIRO Informatique Théorique et Applications.* To appear.

[KWW] W. Khalil, E.G. Wagner, and R.F.C. Walters. Fixed-point semantics for programs in distributive categories. In preparation.

[Lyn87] N.A. Lynch. Multivalued possibility mappings. In *Stepwise Refinement of Distributed Systems*, number 430 in LNCS, pages 519–543, 1987.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. of the 2nd Joint Conference on Artificial Intelligence*, pages 481–489. BCS, 1971.

[MM90] J. Meseguer and U. Montanari. Petri nets are monoids. *Info. and Co.*, 88:105–155, 1990.

[SW93] N. Sabadini and R.F.C. Walters. On functions and processors: an automata theoretic approach to concurrency through distributive categories. Mathematics Report 93-7, Sydney University, 1993.

[BGV91] W. Vogler. Modular construction and partial order semantics of Petri nets. Number 625 in LNCS, 1992.

[Wal] R.F.C. Walters. *Categories and Computer Science.* Carslaw Publications (1991), Cambridge University Press (1992).