Action Refinement from a Logical Point of View

Mila Majster-Cederbaum, Naijun Zhan, and Harald Fecher

Lehrstuhl für Praktische Informatik II,
Fakultät für Mathematik und Informatik,
Mannheim Universität,
D7,27, 68163, Mannheim, Deutschland
{mcb,zhan,hfecher}@pi2.informatik.uni-mannheim.de

Abstract. Action refinement provides a mechanism to design a complex reactive system hierarchically. This paper is devoted to action refinement from a logical point of view, and to combining the hierarchical implementation of a complex system with the hierarchical specification of the system in order to verify it in an easy way. To this end, we use a TCSP-like language with an action refinement operator as a modeling language, and an extension of the modal μ-calculus, called FLC (Fixpoint Logic with Chop) [18], as a specification language. Specifications in FLC can be refined via a mapping that takes as arguments an abstract specification ϕ for the process P, an action a of P and a specification ψ for the process Q that may refine a and produces a refined specification. We prove under some syntactical conditions: if $Q \models \psi$ then $P \models \phi$ iff $P[a \rightsquigarrow Q]$ satisfies the refined specification. Therefore our approach supports 'a priori' verification in system design and can be used to decrease substantially the complexity of verification.

Keywords: action refinement, modal logics, specification, verification, reactive systems

1 Introduction

Generally speaking, it is not easy, even impossible to capture a complex system at the beginning. *The hierarchical development methodology* is useful to understand a complex system. It has made a great success in sequential programming, and is known as *top-down system specification and analysis technique*. [3,9,20] introduced this method into process algebraic settings, called *action refinement*, that provides a mechanism to hierarchically develop concurrent systems. In this method, how to relate a hierarchical specification of a complex system with its hierarchical implementation (abstraction) in order to simplify verification is a challenging problem. In the literature, some first attempts to solve this problem are given, for example in [12,14,15].

The main results obtained in [12,14,15] are as follows: A model of a system is represented by processes or synchronization structures, and a specification of a system is given in terms of modal logic formulae. Then define action refinement both for the model and specification by refinement of a primitive of the abstract

L. Zuck et al. (Eds.): VMCAI 2003, LNCS 2575, pp. 253–267, 2003.

model. [12] coped with action refinement for models from a semantic point of view, whereas [14,15] dealt with it from a syntactic point of view, but they both considered *action refinement for the specification* as a form of *syntactical transformation*. In detail, let P stand for a high level system, ϕ for its specification, Q for the refinement of an abstract action a in P, $\rightsquigarrow$ for refinement relations both for the model and the specification. Then the main result of [12,14,15] can be re-illustrated as follows:

$$\begin{array}{lcl} P & \models & \phi \\ & \Updownarrow & \\ P[a \rightsquigarrow Q] & \models & \phi[a \rightsquigarrow Q] \end{array}$$

where $\phi[a \rightsquigarrow Q]$ stands for substituting $\langle a \rangle$ and $[a]$ in ϕ by some formulae of forms $\langle a_1 \rangle \langle a_2 \rangle \dots \langle a_n \rangle$ and $[a_1][a_2] \dots [a_n]$ respectively, where $a_1 a_2 \dots a_n$ is a run of Q. These results support 'a priori' verification in the following sense: Assume that $P \models \phi$ has been established and ϕ is refined to $\phi[a \rightsquigarrow Q]$ then we automatically obtain a process satisfying $\phi[a \rightsquigarrow Q]$. The analogous remark is true when we refine P to $P[a \rightsquigarrow Q]$. Then we obtain automatically a refined formula that is satisfied by the refined process.

In both approaches, the refinements of the specification are explicitly built on the structure of Q. This restricts the refinement step in two ways: firstly, there are properties of the refined system that cannot be deduced in the setting of [12, 14]. For example, let $P = a; b + a; c$, $\phi = \langle a \rangle$, $Q = a'; (c'; b'; d' + c'; b')$. It's obvious that $P \models \phi$ and $Q \models \langle a' \rangle [c'] \langle b' \rangle$. It is expected that $P[a \rightsquigarrow Q] \models \langle a' \rangle [c'] \langle b' \rangle$. But it cannot be derived using the approaches of [12,14]. Secondly, the refinement step is restricted to one choice of Q for refining an action a, which appears both in the refined process and the refined specification explicitly. In contrast to this we allow action a in P, where $P \models \phi$, to be refined by any process Q that satisfies a specification ψ and show that the refined system $P[a \rightsquigarrow Q]$ satisfies the refined specification under some conditions.

In this paper, we propose a general approach to construct a specification of a low-level complex reactive system based on a higher-level specification and the properties of the refinement of an abstract action. To this end, we also model processes by a TCSP-like language and use FLC as a specification language. The basic idea of our work is to define a refinement mapping Ω which maps a high-level specification ϕ and the properties ψ of the refinement of an abstract action a to a lower-level specification by substituting ψ for $\langle a \rangle$ and $[a]$ in ϕ. For example, in the above example, we can get $\Omega(\phi, \langle a' \rangle [c'] \langle b' \rangle, a) = \langle a' \rangle [c'] \langle b' \rangle$ which is exactly what we expect.

A safety property stipulates that some 'bad thing' does not happen during execution, whereas *a liveness property* stipulates that a 'good thing' eventually happens during execution. Therefore safety properties are completely different from liveness properties. [4] proved that every property can be represented as the conjunction of a safety property and a liveness property in linear models, a similar result for tree models was shown in [5]. On the other hand, the properties of a system also can be classified into *universal* and *existential*. So by our intu-

ition, a refinement mapping should be property-preserving, i.e. a safety property should be refined to a safety property and similarly for the other properties.

We can show the following theorem, called Refinement Theorem, that says: If $Q \models \psi$ and some other conditions hold then

$$\begin{array}{lcl} P & \models & \phi \\ & \Updownarrow & \\ P[a \rightsquigarrow Q] & \models & \Omega(\phi, \psi, a) \end{array}$$

To achieve the intended result, two tasks have to be done: the first one is to select a specification language in which we can implement our idea of refinement. The suitable candidate could be the μ-calculus since most modal and temporal logics that are used as specification languages for concurrent systems can be reduced to it. Unfortunately, the μ-calculus is not suitable for such a task. For example, suppose that $P \models \langle a \rangle \phi_1$, and there is no occurrence of $\langle a \rangle$ or $[a]$ in ϕ_1, and that $Q \models \psi$. After refining a by Q in P, a specification for the refined system that one expects should be naturally $\psi(\phi_1)$ which means that the behavior of the system can be divided into two successive segments such that the former satisfies ψ and the second one meets ϕ_1. But this is no longer a formula of the μ-calculus. Therefore, here we use an extension of the μ-calculus, FLC [18] as a specification language. FLC has a sequential composition operator, called '*chop*' operator (denoted by $\diamond$). Informally, $P \models \phi \diamond \psi$ means that there exists a behaviour of P which can be divided into two successive segments such that the first satisfies ϕ and the second meets ψ. Therefore, the idea of refinement can be implemented as syntactical substitution in this logic. In the above example, if $\langle a \rangle \diamond \phi_1$ is a formula FLC, after substituting ψ for $\langle a \rangle$, the refined formula $\psi \diamond \phi_1$ is still a formula of FLC. Furthermore, the refined formula expresses the expected meaning. In FLC, a safety property can be represented in the form $\nu X.([b]\diamond false) \wedge ([a]\diamond X)$ and a liveness property can be expressed in the form $\mu X.(\langle a \rangle \diamond true) \vee (\langle b \rangle \diamond X)$ (See [23]).

A second issue is the atomicity of action refinement for models. One of our aims in this work is to establish a correspondence between hierarchical implementations and hierarchical specifications, but if we allow that the refining process can be interleaved with others problems will arise. E.g. $(a \parallel_{\{\}} b)[a \rightsquigarrow a_1; a_2]$ means the parallel executions of a and b in which a is refined by $a_1; a_2$. It's obvious that $a \parallel_{\{\}} b$ satisfies $\langle a \rangle$, and $a_1; a_2$ satisfies $\langle a_1 \rangle \diamond (\langle a_2 \rangle \wedge [b] \diamond false)$ which means that $a_1; a_2$ firstly performs a_1, then follows a_2 but cannot perform b. We expect that $a \parallel_{\{\}} b$ meets $\langle a_1 \rangle \diamond (\langle a_2 \rangle \wedge [b] \diamond false)$ after refining a by $a_1; a_2$. This is not true in the case of non-atomic action refinement since b can be performed between the execution of a_1 and a_2, but it is valid if we assume that action refinement is atomic [6,8]. So, in the sequel, we discuss action refinement for models under the assumption of atomicity.

Due to the limitation of space, we will omit all proofs for the theorems and lemmas in this paper. The detailed proofs can be found in [16].

The remainder of this paper is organized as follows: A modeling language is defined in Section 2; Section 3 briefly reviews FLC. A refinement mapping for

specifications is given in Section 4. The correspondence between the hierarchical specification and the hierarchical implementation of a complex system is shown in Section 5. Finally, a brief conclusion is given in Section 6.

2 Modeling Language – A TCSP-Like Process Algebra

2.1 Syntax

As in [14], we use a TCSP-like process algebra in combination with an action refinement operator as a modeling language. We use Act (ranged over by $a, b, c, \dots$) to stand for an infinite set of (atomic) actions, $\surd$ for a special terminating action that only can be performed by the terminated process, A for a subset of Act. $Act \cup \{\surd\}$ is ranged over by $\gamma, \dots$. Let $\mathcal{X}$ be a set of process variables (ranged over by $x, y, z, ...$).

We consider to refine an action by a finite process. Furthermore, it is prohibited to refine an action by a terminated process, which is not only counter-intuitive but also technically difficult, as discussed, e.g. in [21]. Therefore, we define two classes of process expressions. The first are finite and are used as refining processes, the other represent processes that may be refined.

Definition 1. *Let $\mathcal{F}$ be the set of finite processes (ranged by $Q, Q', Q_1, ...$) generated by the following grammar:*

$$Q \ ::= \ a \ \mid \ (Q_1 + Q_2) \ \mid (Q; Q) \mid Q[a \rightsquigarrow Q'].$$

$\mathcal{P}$ be the set of all closed terms generated by the following grammar:

$$P ::= \delta \mid nil \mid a \mid x \mid P_1; P_2 \mid P_1 + P_2 \mid P_1 \parallel_A P_2 \mid rec\ x.P \mid P[a \rightsquigarrow Q]$$

where $Q \in \mathcal{F}$.

An occurrence of a process variable $x \in \mathcal{X}$ is called *bound* in a process expression P iff it does occur within a suberm of the form $rec\ x.P'$, otherwise called *free*. A process expression P is called *closed* iff all occurrences of all variables occuring in it are bound, otherwise it is called *open*.

Intuitively, $P[a \rightsquigarrow Q]$ means that the system replaces the execution of an action a by the execution of the subsystem Q every time when the subsystem P performs a. This operator provides a mechanism to hierarchically design reactive systems. The other expressions of $\mathcal{P}$ can be understood as usual.

Sometimes, we abuse $Act(P)$ to stand for the set of actions which occur in P. We use $F_{act}(P)$ to stand for the set of actions that is possibly performed by P immediately, i.e. $F_{act}(nil) = F_{act}(\delta) = F_{act}(x) = \emptyset$; $F_{act}(a) = \{a\}$; $F_{act}(P_1; P_2) =$ if $F_{act}(P_1) \neq \emptyset$ or $P_1 \equiv \delta$ then $F_{act}(P_1)$ else $F_{act}(P_2)$; $F_{act}(P_1 + P_2) = F_{act}(P_1 \parallel_A P_2) = F_{act}(P_1) \cup F_{act}(P_2)$; $F_{act}(rex\ x.P) = F_{act}(P)$; $F_{act}(P[a \rightsquigarrow Q]) =$ if $a \notin F_{act}(P)$ then $F_{act}(P)$ else $F_{act}(P) \cup F_{act}(Q) - \{a\}$.

Traces and runs of a process P are defined as in [11]. For example, the traces of the process $a; b + a; c$ are $\varepsilon, a, a; b$ and $a; c$, whereas its runs are $a; b$ and $a; c$, where ε stands for empty trace. We use $Tr(P)$ to denote the set of traces of P, and $Run(P)$ the set of its runs. The standard operators on traces and runs of processes will be used, e.g. $\widehat{}$ for catenation, $\restriction$ for restriction.

2.2 Operational Semantics

A transition system is a triple $\mathcal{T} = (S, A, \rightarrow)$ where S is a set of states or processes, A is a set of labels, and $\rightarrow \subseteq S \times A \times S$. Sometimes we use transition systems with initial states, $(S, A, \rightarrow, P_0)$, where $P_0 \in S$.

Here we define the operational semantics of the language by labeled transition systems where $\surd$-labeled transitions are deterministic and final. The meaning of the constructs can be defined in the standard way except for the refinement operator. In order to guarantee the atomicity of the refinement, the basic idea is to define a transition system for the process that maybe be refined, then replace all transitions labeled by the action to be refined by the transition system for the refinement.

Similar to [10], the above idea can be implemented by introducing an auxiliary operator $*$ to indicate that a process prefixed with it is the remainder of some process, which has the highest execution precedence and must be executed atomically. The operator is used to guarantee the atomicity of refinements of actions. The intermediate language, denoted by $\mathcal{P}^*$, ranged over by $s, \ldots$, is given by:

$$s \quad ::= nil \mid \delta \mid a \mid x \mid *s \mid s; s \mid P + P \mid s \parallel_A s \mid s[a \rightsquigarrow Q] \mid rec\ x.P$$

where $P \in \mathcal{P}, Q \in \mathcal{F}$.

Definition 2. *Let* $\surd$ *and* **ab** *be the minimal relations on* $\mathcal{P}^*$ *which satisfy the following rules respectively:*

$$\frac{}{\surd(nil)} \qquad \frac{\surd(s)}{\surd(*s)} \qquad \frac{\surd(s_1)\ \surd(s_2)}{\surd(s_1 \parallel_A s_2)}$$
$$\surd(rec\ x.s) \qquad \surd(s_1; s_2)$$
$$\surd(s[a \rightsquigarrow Q])$$

Definition of $\surd$

$$\frac{\surd(s)}{\mathbf{ab}(s)} \qquad \frac{}{\mathbf{ab}(\delta)} \qquad \frac{\mathbf{ab}(s_1)\ \ \mathbf{ab}(s_2)}{\mathbf{ab}(s_1; s_2)} \qquad \frac{\mathbf{ab}(s)}{\mathbf{ab}(s[a \rightsquigarrow Q])}$$
$$\mathbf{ab}(a) \qquad \mathbf{ab}(s_1 + s_2) \qquad \mathbf{ab}(rec\ x.s)$$
$$\mathbf{ab}(x) \qquad \mathbf{ab}(s_1 \parallel_A s_2)$$
where $Q \in \mathcal{F}$.

Definition of **ab**

Note that $\surd(s)$ means that s terminates after executing the terminated action $\surd$; $ab(s)$ means that s is in the $*$-free fragment of $\mathcal{P}^*$.

A process s is called *abstract* if $\mathbf{ab}(s)$, otherwise, called *concrete* . For technical reason, as in [10], we require the following well-formedness conditions on $\mathcal{P}^*, \mathcal{P}$ and $\mathcal{F}$: None of operands of $+$ meets the predicate $\surd$; Furthermore, recursion is allowed on guardedness in the presence of ; only. $*s$ behaves like s except that the execution of $*s$ can not be interleaved with others and it is to be executed before abstract processes.

An operational semantics of the process algebra is given by the following inference rules:

$$\text{Nil}\quad nil \xrightarrow{\surd} \delta \qquad \text{Act}\quad a \xrightarrow{a} nil$$

$$\text{Nd}\quad \frac{s_1 \xrightarrow{a} s_1'}{s_1+s_2 \xrightarrow{a} s_1' \text{ and } s_2+s_1 \xrightarrow{a} s_1'} \qquad \text{Seq-1}\quad \frac{s_1 \xrightarrow{a} s_1'}{s_1;s_2 \xrightarrow{a} s_1';s_2}$$

$$\text{Seq-2}\quad \frac{s_1 \xrightarrow{\surd} s_1' \;\; s_2 \xrightarrow{a} s_2'}{s_1;s_2 \xrightarrow{a} s_2'} \qquad \text{Ref-1}\quad \frac{s \xrightarrow{b} s'}{s[a \rightsquigarrow Q] \xrightarrow{b} s'[a \rightsquigarrow Q]}\;\; a \neq b$$

$$\text{Ref-2}\quad \frac{s \xrightarrow{a} s' \;\; Q \xrightarrow{a'} Q'}{s[a \rightsquigarrow Q] \xrightarrow{a'} (*Q');s'[a \rightsquigarrow Q]} \qquad \text{Rec}\quad \frac{s_1[rec\ x.s_1/x] \xrightarrow{a} s_1'}{rec\ x.s_1 \xrightarrow{a} s_1'}$$

$$\text{S-1}\quad \frac{s \xrightarrow{a} s'}{*s \xrightarrow{a} *s'} \qquad \text{Syn1}\quad \frac{s_1 \xrightarrow{\surd} s_1' \text{ and } s_2 \xrightarrow{\surd} s_2'}{s_1 \|_A s_2 \xrightarrow{\surd} s_1' \|_A s_2'}$$

$$\text{S-2}\quad \frac{s_1 \xrightarrow{a} s_1'}{s_1 \|_A s_2 \xrightarrow{a} s_1' \|_A s_2 \text{ and } s_2 \|_A s_1 \xrightarrow{a} s_2 \|_A s_1'}\; a \notin A \wedge \mathbf{ab}(s_2)$$

$$\text{Syn2}\quad \frac{s_1 \xrightarrow{a} s_1' \;\; s_2 \xrightarrow{a} s_2'}{s_1 \|_A s_2 \xrightarrow{a} s_1' \|_A s_2' \text{ and } s_2 \|_A s_1 \xrightarrow{a} s_2' \|_A s_1'}\; a \in A \wedge (\mathbf{ab}(s_1) \wedge \mathbf{ab}(s_2))$$

$$\text{Syn3}\quad \frac{s_1 \xrightarrow{a} s_1' \;\; s_2 \xrightarrow{a} s_2'}{s_1 \|_A s_2 \xrightarrow{a} s_1' \|_A s_2' \text{ and } s_2 \|_A s_1 \xrightarrow{a} s_2' \|_A s_1'}\; a \in A \wedge (\neg\mathbf{ab}(s_1) \wedge \neg\mathbf{ab}(s_2))$$

Definition 3. – *A binary symmetric relation R over the closed terms of $\mathcal{P}^*$ is a strong bisimulation if for all $(s_1, s_2) \in R$ and $s_1 \xrightarrow{\gamma} s_1'$, there exists s_2' such that $s_2 \xrightarrow{\gamma} s_2'$ and $(s_1', s_2') \in R$.*

– *s_1 and s_2 are strong bisimilar, denoted by $s_1 \cong s_2$, if and only if there exists a strong bisimulation R such that $(s_1, s_2) \in R$.*

According to the above semantics, it is easy to show that

Lemma 1. *For any closed term $s \in \mathcal{P}^*$, $s \cong *s$.*

Since a concrete process can not communicate with an abstract process, so $\cong$ is not preserved by $\|_A$ in $\mathcal{P}^*$. Even more $\cong$ is not a congruence relation over the language $\mathcal{P}$. For example, $a_1; a_2 \cong a[a \rightsquigarrow a_1; a_2]$, but $(a_1; a_2) \|_{\{\}} b \ncong a[a \rightsquigarrow a_1; a_2] \|_{\{\}} b$. However, once we strengthen Definition 3 by adding the following condition:

– If $\mathbf{ab}(s_1)$ then $\mathbf{ab}(s_2)$

then the resulting largest bisimulation, denoted by $\cong_{\mathbf{ab}}$, is a congruence relation over $\mathcal{P}^*$. Besides, obviously, $\cong_{\mathbf{ab}}$ is a proper subset of $\cong$. That is,

Lemma 2. $\cong_{\mathbf{ab}} \subset \cong$.

Theorem 1. $\cong_{\mathbf{ab}}$ *is a congruence over $\mathcal{P}^*$.*

3 Specification Language – FLC

3.1 Syntax and Semantics of FLC

Let $X, Y, Z, \ldots$ range over an infinite set Var of *variables*, $p, q, r, \ldots$ over an assumed finite set $Prop$ of *atomic propositions*, that contains $true$ and $false$.

The formulae of FLC are generated according to the following grammar:

$$\phi ::= p \mid term \mid X \mid [a] \mid \langle a \rangle \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \diamond \phi \mid \mu X.\phi \mid \nu X.\phi,$$

where $p \in Prop, X \in Var$ and $a \in Act$.

In the later, $\boxed{\text{a}}$ stands for $\langle a \rangle$ or $[a]$, *op* for $\wedge$ or $\vee$, and σ for μ or ν.

As in the modal μ-calculus, the two *fixpoint operators* μX and νX bind the respective variable X and we will apply the usual terminology of *free* and *bound* variables in a formula, *closed* and *open* formulae etc. $term, p, \langle a \rangle$ and $[a]$ are called *atomic formulae*. We say that X is guarded in ϕ if each occurrence of X is within some sub-formula $\boxed{\text{a}}\diamond\psi$. $\mu X.X$ is equivalent to $false$ and $\nu X.X$ is equivalent to $true$. It is easy to show that every closed formula is equivalent to a formula in which all variables are guarded.

The FLC is interpreted over a given labeled transition system $T = (S, Act, \rightarrow)$. Furthermore, an *interpretation* $\mathcal{I} \in (Prop \mapsto 2^S)$ is assumed, which assigns to each atomic proposition the set of states for which it is valid, and satisfies that $\mathcal{I}(false) = \emptyset$ and $\mathcal{I}(true) = S$. The meaning of variables is given by an *environment* $\varrho : Var \mapsto (2^S \mapsto 2^S)$ that maps variables to monotonic functions from sets to sets. $\varrho[X \mapsto f]$ agrees with ϱ except for associating f with X. The formulae of FLC are interpreted as *monotonic predicate transformers* that are simply mappings $f : 2^S \mapsto 2^S$ which are monotonic w.r.t. the inclusion ordering on 2^S. We use MPT_T to represent all these monotonic predicate transformers over S. MPT_T together with the inclusion ordering defined by $f \subseteq f'$ iff $f(A) \subseteq f'(A)$ for all $A \subseteq S$ forms a complete lattice. We denote the join and meet operators by $\sqcup$ and $\sqcap$. By Tarski-Knaster Theorem [24], the least and greatest fixed points of monotonic functions: $(2^S \mapsto 2^S) \mapsto (2^S \mapsto 2^S)$ exist. They are used to interpret the fixed point formulae.

The predicate transformer assigned to an formula ϕ, denoted by $\mathcal{C}_T^{\mathcal{I}}(\phi)(\varrho)$, is inductively constructed as follows:

$$\begin{aligned}
\mathcal{C}_T^{\mathcal{I}}(p)(\varrho)(E) &= \mathcal{I}(p) \\
\mathcal{C}_T^{\mathcal{I}}([a])(\varrho)(E) &= \{s \mid \forall s' : s \xrightarrow{a} s' \Rightarrow s' \in E\} \\
\mathcal{C}_T^{\mathcal{I}}(\langle a \rangle)(\varrho)(E) &= \{s \mid \exists s' : s \xrightarrow{a} s' \wedge s' \in E\} \\
\mathcal{C}_T^{\mathcal{I}}(\phi_1 \wedge \phi_2)(\varrho)(E) &= \mathcal{C}_T^{\mathcal{I}}(\phi_1)(\varrho)(E) \cap \mathcal{C}_T^{\mathcal{I}}(\phi_2)(\varrho)(E) \\
\mathcal{C}_T^{\mathcal{I}}(\phi_1 \vee \phi_2)(\varrho)(E) &= \mathcal{C}_T^{\mathcal{I}}(\phi_1)(\varrho)(E) \cup \mathcal{C}_T^{\mathcal{I}}(\phi_2)(\varrho)(E) \\
\mathcal{C}_T^{\mathcal{I}}(X)(\varrho) &= \varrho(X) \\
\mathcal{C}_T^{\mathcal{I}}(\mu X.\phi)(\varrho) &= \sqcap\{f \in MPF_T \mid \mathcal{C}_T^{\mathcal{I}}(\phi)(\varrho[X \rightsquigarrow f]) \subseteq f\} \\
\mathcal{C}_T^{\mathcal{I}}(\nu X.\phi)(\varrho) &= \sqcup\{f \in MPF_T \mid \mathcal{C}_T^{\mathcal{I}}(\phi)(\varrho[X \rightsquigarrow f]) \supseteq f\} \\
\mathcal{C}_T^{\mathcal{I}}(term)(\varrho)(E) &= E \\
\mathcal{C}_T^{\mathcal{I}}(\phi_1 \diamond \phi_2)(\varrho) &= \mathcal{C}_T^{\mathcal{I}}(\phi_1)(\varrho) \cdot \mathcal{C}_T^{\mathcal{I}}(\phi_2)(\varrho)
\end{aligned}$$

where $\cdot$ stands for the compositional operator over functions, $\setminus$ for the complementary operator over sets.

The set of processes *satisfying* a given closed formula ϕ is $\phi(S)$. A process P with associated rooted transition system $((S_P, Act(P), \rightarrow_P), P)$, where S_P

stands for the set of states in the transition system, is said to satisfy ϕ iff $P \in \mathcal{C}_T^{\mathcal{I}}(\phi)(\varrho)(S_P)$ for some interpretation $\mathcal{I}$ and environment ϱ, denoted by $P \models \phi$. $\phi \Leftrightarrow \psi$ denotes that for any process P with associated rooted transition system $T = ((S_P, Act(P), \rightarrow_P), P)$, $\mathcal{C}_T^{\mathcal{I}}(\phi)(\varrho)(E) = \mathcal{C}_T^{\mathcal{I}}(\psi)(\varrho)(E)$ for any $E \subset S_P$, interpretation $\mathcal{I}$, and environment ϱ. The other notations can be defined in a standard way.

[18] proved that FLC is strictly more expressive than the μ-calculus since context-free processes can be characterized in it; FLC is decidable for finite-state processes, undecidable for context-free processes, satisfiability and validity of it are undecidable; And FLC does not enjoy the finite-model property. [13] presented a model-checking algorithm of FLC for finite-state processes.

For the sake of proving technique, we introduce approximants of fixed point formulae. Let $\alpha, \beta, \lambda \in \mathcal{O}\mathrm{n}$, the ordinals, there λ is a limit ordinal. Then $\mu^0 X.\phi_1 = false$, $\mu^{\alpha+1} X.\phi_1 = \phi_1\{\mu^\alpha X.\phi_1/X\}$, $\mu^\lambda X.\phi_1 = \bigvee_{\alpha<\lambda} \mu^\alpha X.\phi_1$. $\nu^0 X.\phi_1 = true$, $\nu^{\alpha+1} X.\phi_1 = \phi_1\{\nu^\alpha X.\phi_1/X\}$, $\nu^\lambda X.\phi_1 = \bigwedge_{\alpha<\lambda} \nu^\alpha X.\phi_1$. Note that by Tarski and Knaster's Theorem, $\mu X.\phi \Leftrightarrow \bigvee_{\alpha\in\mathcal{O}\mathrm{n}} \mu^\alpha X.\phi$ and $\nu X.\phi \Leftrightarrow \bigwedge_{\alpha\in\mathcal{O}\mathrm{n}} \nu^\alpha X.\phi$. If only finite state processes are considered $\mathcal{O}\mathrm{n}$ can be replaced by ω, moreover, if P is a finite-state process then $P \models \sigma X.\phi$ iff $P \models \sigma^k X.\phi$ where $\sigma \in \{\nu, \mu\}$, and k is the number of the states or processes of the rooted transition system associating with P. As shown in [7], we can show that $\mathcal{O}\mathrm{n}$ can be replaced by ω_1, where ω_1 stands for the first uncountable limit ordinal.

Convention: In the sequel, we assume that the unary operators have the highest precedence, $\Diamond$ has a priority to other binary operators, $\vee$ and $\wedge$ have the same precedence too, but they have a priority over $\Rightarrow$ and $\Leftrightarrow$ in order to avoid the excessive use of brackets and improve the readability.

3.2 Normal Form

In this subsection, we define a special subset of FLC, called *normal form formulae (nff for short).* Intuitively, a normal form formula exactly corresponds to a formula of the μ-calculus if we omit *term* and the 'chop' operator occurring in it. [18] pointed out that the modal μ-calculus can be encoded into FLC straightforwardly by replacing $\langle a\rangle\Diamond\phi$ and $[a]\Diamond\phi$ for $\langle a\rangle\phi$ and $[a]\phi$ respectively. This implies that the expressiveness of the subset is at least as powerful as the modal μ-calculus.

Definition 4. *Given a formula $\phi \in FLC$, we define its* runs *as follows:*

$$Run(\phi) \;\widehat{=}\; \begin{cases} \{\varepsilon\} & \text{if } \phi = p, term, \text{ or } X \\ \{a\} & \text{if } \phi = \boxed{a} \\ Run(\phi_1) \cup Run(\phi_2) & \text{if } \phi = \phi_1 \, op \, \phi_2 \\ \{s\hat{}t \mid s \in Run(\phi_1) \wedge t \in Run(\phi_2)\} & \text{if } \phi = \phi_1 \Diamond \phi_2 \\ \bigcup_{\alpha<\mathcal{O}n} Run(\sigma^\alpha X.\phi_1) & \text{if } \phi = \sigma X.\phi_1 \end{cases}$$

where ε stands for the empty run that has the property $s\hat{}\varepsilon = s = \varepsilon\hat{}s$ for any run s. The set of traces of ϕ is defined as $Tr(\phi)\widehat{=}\{s \mid s \in Run(\phi) \vee \exists s'.s\hat{}s' \in Run(\phi)\}$.

Definition 5. *A formula which is of the following form is called* normal form*:*

$$\psi ::= p \mid X \mid term \mid \boxed{a} \mid \boxed{a}\Diamond\psi \mid \psi\, op\, \psi \mid \sigma X.\psi.$$

The set of nffs is denoted by $\mathcal{NF}$. For simplicity, we use cnff *for closed normal form formula.*

Definition 6. *Given a nff ϕ, we define its first non-trivial atomic sub-formula as follows:*

$$FSub(\phi) \;\widehat{=}\; \begin{cases} \{\} & \text{if } \phi = p, X, \text{ or } term \\ \{\langle a\rangle\} & \text{if } \phi = \langle a\rangle \text{ or } \langle a\rangle\Diamond\phi_1 \\ \{[a]\} & \text{if } \phi = [a] \text{ or } [a]\Diamond\phi_1 \\ FSub(\phi_1) \cup FSub(\phi_2) & \text{if } \phi = \phi_1\, op\, \phi_2 \\ FSub(\phi_1) & \text{if } \phi = \sigma X.\phi_1 \end{cases}$$

The set of first actions of ϕ is defined as: $F_{act}(\phi)\widehat{=}\{a \mid \boxed{a} \in FSub(\phi)\}$. The set of key actions of ϕ w.r.t. P, denoted by $K_{act}(\phi, P)$, is defined as: $K_{act}(\phi, P)\widehat{=}\{a \mid \exists s \in Tr(\phi).s \in Run(P) \wedge s\hat{}a \notin Run(P) \wedge s\hat{}a \in Tr(\phi)\}$.

Example 1. Given a formula $\phi\widehat{=}\langle a\rangle\Diamond\langle b\rangle \wedge [c]\Diamond\langle e\rangle\Diamond[f]$, $FSub(\phi) = \{\langle a\rangle, [c]\}$, the set of its first actions is $\{a, c\}$, and the set of its key actions w.r.t. $P\widehat{=}a; b{+}c; (e{+}d)$ is $\{f\}$.

Remark 1. The intention of key actions is to avoid the case that $P \models \phi$ and $Q \models \psi$, but $P; Q \not\models \phi\Diamond\psi$, because ϕ concerned some execution of Q. For instance, in the above example, let $Q\widehat{=}f; g$ and $\psi\widehat{=}\langle f\rangle\Diamond\langle g\rangle$. It's obvious that $P \models \phi$ and $Q \models \psi$, but $P; Q \not\models \phi\Diamond\psi$ since $K_{act}(\phi, P) \cap F_{act}(Q) \neq \emptyset$.

Definition 7. *A formula $\phi \in \mathcal{NF}$ is called* existential formula *if $\forall a \in Act.[a] \notin FSub(\phi)$. We use $\mathcal{ENF}$ to stand for the set of existential formulae. Dually, a formula $\phi \in \mathcal{NF}$ is called* universal formula *if $\forall a \in Act.\langle a\rangle \notin FSub(\phi)$. We use $\mathcal{UNF}$ to stand for the set of universal formulae. For technical reasons, we stipulate term $\notin \mathcal{UNF}$. A formula is called property formula if $\phi \Leftrightarrow \phi_1 \wedge \phi_2$, where $\phi_1 \in \mathcal{ENF}$ and $\phi_2 \in \mathcal{UNF}$. The set of property formulae is denoted by $\mathcal{PNF}$. A formula ϕ is called* pure path formula *if all variables occurring in it are guarded and no propositional letter occurs in it.*

For $\mathcal{NF}, \mathcal{ENF}, \mathcal{UNF}$, we have

Theorem 2. *$\mathcal{NF}, \mathcal{ENF}, \mathcal{UNF}$ are closed under all operators of the logic. I.e., there exists a $\psi \in \mathcal{NF}(\mathcal{ENF}, \mathcal{UNF})$ which is equivalent to $\phi\, op\, \varphi$ or $\sigma X.\phi$ for any closed ϕ and φ, if $\phi, \varphi \in \mathcal{NF}(\mathcal{ENF}, \mathcal{UNF})$ where $op \in \{\vee, \wedge, \Diamond\}$.*

FLC does not have finite model property [18], i.e., not all of satisfiable formulae of FLC have a finite model, but [13] showed that it has the tree model property, that is

Theorem 3. *If $s_1 \cong s_2$, then for any closed formula ϕ, $s_1 \models \phi$ iff $s_2 \models \phi$.*

4 Hierarchically Specifying Complex Reactive Systems

As the complexity of reactive system designs becomes overwhelming very quickly, methods which allow to develop designs in a hierarchical fashion must be supported by the design formalisms employed. Such methods allow to develop a design on different levels of abstraction thereby making the development procedure more transparent and thus tractable: Most likely, a developer first divides the intended (complex) design into various "sub-designs" to capture the abstract overall structure of the complete design. Subsequently, the sub-designs will be developed by enriching them step by step with details. This is the design technique usually encountered in practice, see e.g. in [22]. In the algebraic settings, action refinement as introduced in Section 2 supports the hierarchical design. In this section we investigate how to provide such technique in a logical framework.

To this end, we define a refining mapping which substitutes the properties of the refinement of an abstract action for the ones of the abstract action in a high-level specification and produces a lower-level specification. In a logical framework, actions are addressed as modalities and descriptions of systems are represented by formulae. In most modal logics, there are two kinds of modalities, i.e. $\langle a\rangle$ and $[a]$ which are used to express existential and universal properties respectively. By our intuition, a refinement mapping should be property-preserving, i.e. an existential property should be refined to an existential property and similarly for the other properties. Otherwise, the mapping is meaningless since it's impossible to establish a correspondence between action refinement for models and action refinement for specifications. For example, $P \widehat{=} a; b + a; c \models \langle a\rangle\Diamond\langle b\rangle$, $a_1; a_2 \models [a_1]\Diamond\langle a_2\rangle$, but $P[a \rightsquigarrow a_1; a_2] \not\models ([a_1]\Diamond\langle a_2\rangle)\Diamond\langle b\rangle$, since in the high-level specification, $\langle a\rangle\Diamond\langle b\rangle$ is an existential liveness property, however its refinement becomes a universal liveness property.

To ensure the mapping is property-preserving, we partition the property ψ of the refinement of a into two parts: existential property ψ_1 and universal property ψ_2 i.e. $\psi \in \mathcal{PNF}$. $[a]$ will be replaced by ψ_2, and $\langle a\rangle$ will be replaced by ψ_1. This is justified by the result shown in [5] that any property can be expressed as the intersection of a liveness property and a safety property in branching temporal logics. So, $\mathcal{PNF}$ is powerful enough to define the properties of reactive systems.

Therefore, we define the refinement mapping as follows:

Definition 8. *Suppose $\phi \in \mathcal{NF}$ is a high-level specification, a is an abstract action to be refined, $\psi \Leftrightarrow \psi_1 \wedge \psi_2 \in \mathcal{PNF}$ is the description of the refinement of a where $\psi_1 \in \mathcal{ENF}$ and $\psi_2 \in \mathcal{UNF}$. We define the refinement mapping, denoted by $\Omega(\phi, \psi, a)$, as $\phi\{\psi_1/\langle a\rangle, \psi_2/[a]\}$.*

According to the above definition, we have the following results.

Lemma 3. *Suppose X does not occur in ψ. Then*

$$\Omega(\phi_1\{\phi_2/X\}, \psi, a) \Leftrightarrow \Omega(\phi_1, \psi, a)\{\Omega(\phi_2, \psi, a,)/X\}.$$

Lemma 4. *If $\phi \Leftrightarrow \phi'$ then $\Omega(\phi, \psi, a) \Leftrightarrow \Omega(\phi', \psi, a)$.*

Theorem 4 (Applicability). *If $\phi \in \mathcal{NF}$ and $\psi \in \mathcal{PNF}$, then $\Omega(\phi, \psi, a) \in \mathcal{NF}$; If $\phi, \psi \in \mathcal{PNF}$, then $\Omega(\phi, \psi, a) \in \mathcal{PNF}$.*

We give the following example which is firstly given in [8] to demonstrate how to use our approach to hierarchically specify a complex systems.

Example 2. Suppose that a salesman has to go by car from his office in Paris to another office in London and work there for some time, and then has to go back to Paris repeatedly. He takes a hovercraft to cross the Channel.

At the beginning, we can specify the system as:

$$\phi \widehat{=} \nu X. \begin{pmatrix} \langle leave\_Paris \rangle \Diamond [fr\_through\_the\_Channel] \\ \Diamond \langle arrive\_in\_London \rangle \Diamond \langle work \rangle \Diamond \langle leave\_London \rangle \\ \Diamond [gb\_through\_the\_Channel] \Diamond \langle arrive\_in\_Paris \rangle \Diamond X \end{pmatrix}.$$

Then, we can refine $x\_through\_the\_Channel$ by a process with the property

$$\psi_x \widehat{=} \begin{pmatrix} [x\_load] \Diamond [x\_departure] \Diamond \langle cross\_the\_Channel \rangle \\ \Diamond \langle \overline{x}\_arrival \rangle \Diamond \langle \overline{x}\_unload \rangle \wedge true \end{pmatrix}.$$

Further, we can refine $x\_departure$ by a process with the property

$$\psi_2 \widehat{=} [finish\_loading] \Diamond \langle engine\_on \rangle \Diamond \langle bye - bye \rangle \wedge true$$

where $finish\_loading$ signals the end of loading, and $cross\_the\_Channel$ by a process with the property

$$\psi_3 \widehat{=} \langle sit\_down \rangle \Diamond (\begin{matrix} \langle newspaper \rangle \Diamond (\langle tea \rangle \vee \langle coffee \rangle) \\ \vee \ (\langle tea \rangle \vee \langle coffee \rangle) \Diamond \langle newspaper \rangle \end{matrix}) \Diamond \langle stand\_up \rangle \wedge true.$$

Hence, the final system should satisfy the specification given by

$$\Omega(\phi, \Omega(\Omega(\psi_1, \psi_2, x\_departure), \psi_3, cross\_the\_Channel), x\_through\_the\_Channel).$$

Where $x \in \{fr, gb\}$, and if $x = fr$ then $\overline{x} = gb$ else $\overline{x} = fr$.

5 Hierarchically Verifying Complex Reactive Systems

In this section we establish a correspondence presented by the Refinement Theorem below between action refinement for models and action refinement for specification. It states that if $Q \models \psi$ then under certain syntactical conditions $P \models \phi$ iff $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi, a)$. This result supports 'a priori' verification. In the development process we start with $P \models \phi$ and either refine P and obtain automatically a (relevant) formula that is satisfied by $P[a \rightsquigarrow Q]$. Or, we refine ϕ using $\Omega(\phi, \psi, a)$ and obtain automatically a refined process $P[a \rightsquigarrow Q]$ that satisfies the refined specification. Of course such refinement steps may be iterated.

It's possible that ψ only describes the partial execution of Q, so the semantic model of the refined specification cannot be simulated by the refined system.

Therefore, $P \models \phi$ and $Q \models \psi$, but $P[a \rightsquigarrow Q] \not\models \Omega(\phi, \psi, a)$. For example, it's obvious that $a; b + a; c \models \phi \widehat{=} \langle a \rangle \Diamond \langle b \rangle$ and $a_1; a_2 \models \psi \widehat{=} \langle a_1 \rangle$, but $(a; b + a; c)[a \rightsquigarrow a_1; a_2] \not\models \langle a_1 \rangle \Diamond \langle b \rangle$. In order to solve such a problem, we define a bridging formula $Br(\psi, Q)$ which depends on Q and ψ and will be appended to ψ such that $\psi \Diamond Br(\psi, Q)$ describes the full execution of Q, as follows:

Definition 9. *Given a sequence of actions s and a set of sequences of actions D, we define the distance from s to D, denoted by $d(s, D)$, as $d(s, D) = $ if $s \in D$ or $\exists s_1, s_2 . s = s_1 \frown s_2 \wedge s_1 \in D$ or $D = \emptyset$ then 0 else min $\{|s_2| \mid s_1 \frown s_2 \in D \wedge \forall 1 \leq i \leq |s_1| . s(i) = s_1(i) \wedge |s| > |s_1| \Rightarrow s_2(1) \neq s(|s_1| + 1)\}$, where $|s|$ stands for the length of s, $s(i)$ for the ith element of sequence s. Suppose B is another set of sequences of actions, we define $d(B, A) = max\ \{d(s, A) \mid s \in B\} \cup \{0\}$.*

Definition 10. *Given a process P and a formula ϕ, we define a bridging formula as follows: $Br(\phi, P) \widehat{=} term \vee \mu^{d(Run(\phi), Run(P)) + 1} X . (term \vee (\bigvee_{a \in Act(P)} \langle a \rangle) \Diamond X)$. For example, in the above example, $Br(\langle a_1 \rangle, a_1; a_2) = term \vee \langle a_1 \rangle \vee \langle a_2 \rangle$. It is easy to show that $(a; b + a; c)[a \rightsquigarrow a_1; a_2] \models (\langle a_1 \rangle \Diamond Br(\langle a_1 \rangle, a_1; a_2)) \Diamond \langle b \rangle$.*

Lemma 5. *$P \models \phi$ iff $P \models \phi \Diamond Br(\phi, P)$.*

In the following Refinement Theorem, $Act(P) \cap Act(Q) = \emptyset$ ensures that no deadlock will be introduced or removed by action refinement.

For the first part of the theorem, it is possible that ψ describes some properties concerning some partial or full executions of $P'[a \rightsquigarrow Q]$, $Q; P'[a \rightsquigarrow Q]$ or $Q; Q$ but it is not satisfied by them, where P' is P itself or one of its derivatives. For instance, let $P \widehat{=} a; b; c$, $\phi \widehat{=} \langle a \rangle \Diamond \langle b \rangle$, $Q \widehat{=} a_1$, and $\psi \widehat{=} \langle a_1 \rangle \Diamond [b] \Diamond \langle d \rangle$. It is obvious that $P \models \phi$ and $Q \models \psi$ but $P[a \rightsquigarrow Q] \not\models \Omega(\phi, \psi \Diamond Br(\psi, Q), a)$. Therefore, we stipulate that $K_{act}(\psi, Q) \cap (Act(P) \cup F_{act}(Q)) = \emptyset$ and $F_{act}(\psi) \cap Act(P) = \emptyset$ in order to avoid such case.

For the second part of the theorem, it must be ensured that P performs the action a according to ϕ iff $P[a \rightsquigarrow Q]$ performs some actions of Q according to ψ in $\Omega(\phi, \psi, a)$. Otherwise, the converse is not true. For instance, let $P \widehat{=} a; b; c$, $\phi \widehat{=} [a] \Diamond \langle c \rangle$, $Q \widehat{=} d; e$, $\psi \widehat{=} true$. It's obvious that $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi, a)$, but $P \not\models \phi$. So we require $Run(\psi) \subseteq Tr(Q) \restriction_{Act}$ and ψ is a pure path formula.

Theorem 5 (Refinement Theorem).
If $Q \models \psi$ and $Act(P) \cap Act(Q) = \emptyset$ then

- *if $K_{act}(\psi, Q) \cap (Act(P) \cup F_{act}(Q)) = \emptyset$ and $F_{act}(\psi) \cap Act(P) = \emptyset$, then $P \models \phi$ implies $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi \Diamond Br(\psi, Q), a)$;*
- *if $Run(\psi) \subseteq Tr(Q) \restriction_{Act}$, and ψ is a pure path formula then $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi, a)$ implies $P \models \phi$.*

Where $\psi \in \mathcal{PNF}, \phi \in \mathcal{NF}$.

Remark 2. Assume the number of actions and conbinators occuring in Q and P are n and m respectively, the number of atomic formulae and connectives occuring in ψ is l. Then the complexity of checking the syntactical constraints is in $\mathcal{O}(ln^2(m + n))$. On the other hand, FLC model checking is in EXPTIME

(See [13]). Therefore, using the above theorem, model-checking for a complex reactive system will be reduced to model-checking of its simple abstraction and the refinement of a primitive of the system, and this will indeed decrease the complexity of verification of the system.

We will continue Example 2 to show how to apply the Refinement Theorem to verify a complex system hierarchically.

Example 3. At the beginning, we can implement the system as

$$Sys \mathrel{\widehat{=}} fr\_Channel||_{\{fr\_through\_the\_Channel\}}salesman \\ ||_{\{gb\_through\_the\_Channel\}}gb\_Channel.$$

Where

$$x\_Channel \mathrel{\widehat{=}} rec\, y.x\_through\_the\_Channel; y,$$

and

$$salesman \mathrel{\widehat{=}} rec\, x.leave\_Paris; fr\_through\_the\_Channel; arrive\_in\_London; \\ work; leave\_London; gb\_through\_the\_Channel; arrive\_in\_Paris; x.$$

It's obviously, $Sys \models \phi$.

Then, $x\_through\_the\_Channel$ is implemented by

$$subsys_x \mathrel{\widehat{=}} x\_load||_{\{x\_load\}}Channel$$

where

$$Channel \mathrel{\widehat{=}} fr\_platform \,||_{\{fr\_arrival, fr\_departure\}}hovercraft \\ ||_{\{gb\_arrival, gb\_departure\}}gb\_platform, \text{ where}$$

$$hovercraft \mathrel{\widehat{=}} fr\_departure; cross\_the\_Channel; gb\_arrival + \\ gb\_departure; cross\_the\_Channel; fr\_arrival,$$

$$x\_platform \mathrel{\widehat{=}} x\_load; x\_departure + x\_arrival; x\_unload.$$

It's easy to show that $subsys_x \models \psi_x$.

Further, we can refine $x\_departure$ by $subsys_2$ and $cross\_the\_Channel$ by $subsys_3$, where,

$$subsys_2 \mathrel{\widehat{=}} finishing\_loading; engine\_on; bye - bye,$$

$$subsys_3 \mathrel{\widehat{=}} sit\_down; ((coffee + tea)||_{\{\}}newspaper); stand\_up.$$

Certainly, $subsys_2 \models \psi_2$ and $subsys_3 \models \psi_3$.

The final system is obtained as:

$$Sys[x\_through\_the\_Channel \rightsquigarrow subsys_x[\begin{matrix} x\_departure \rightsquigarrow subsys_2, \\ cross\_the\_Channel \rightsquigarrow subsys_3 \end{matrix}]].$$

Where $x \in \{fr, gb\}$. According to the Refinement Theorem, the final system satisfies the final specification.

6 Concluding Remarks

In this paper, we present an approach to refine an abstract specification by defining a refinement mapping from a high-level specification and the properties of the refined lower-level component to a lower-level specification. Furthermore,

we show $P \models \phi$ iff $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi, a)$ provided $Q \models \psi$ and some syntactical conditions hold.

Similar results are shown in [12,14,15], but in their approaches, a refined specification is obtained from the original specification and the refinement Q. Therefore, some interesting expected properties of the refined system cannot be derived using their approaches. Besides, we can show that their approaches can be seen as a special case of our method presented in this paper from a constructing specification point of view. [2] discussed composing, refining specifications of reactive systems as some sound rules of a logic. [1] considered the problem given a low-level specification and a higher-level specification, how to construct a mapping from the former to the latter in order to guarantee the former implements the latter. Our refinement mapping Ω maps the abstract specification to the detailed specification, i.e. we go the converse direction.

In our framework, composing specifications also can be dealt with, for example, supposing $P \models \phi$ and $Q \models \psi$, we can get a composite specification like $\phi \diamond Br(\phi, P) \diamond \psi$ for $P; Q$. We would like to leave more detailed discussion related to this topic for the full version of this paper.

In this paper, we used the standard interleaving setting, so we only consider the case of atomic action refinement. In fact, we believe our approach can be applied to the case of non-atomic action refinement, too, if a suitable specification language is available. For example, we can extend νTrPTL [19] with 'chop' and its duality as a specification language.

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991.
2. M. Abadi and G. Plotkin. A logical view of composition and refinement. *Theoretical Computer Science*, 114:3–30, 1993.
3. Aceto L. and Hennessy. M, Towards action refinement in process algebra. Proc. LICS'89, 138–145,1989.
4. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters,* 21:191–185, 1995.
5. A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. 18th ICALP,July 1991, LNCS 510, pp. 76–92.
6. G. Boudol. Atomic actions. *Bull. European Assoc. The. Com. Sci.* 38:136–144.
7. J.C. Bradfield. *Verifying Temporal Properties of Systems.* Birkhäuser Boston, Mass.
8. P. Degano and R. Gorrieri, Atomic Refinement in Process Description Languages. TR 17–91 HP Pisa Center, 1991.
9. R.J. van Glabbeek and F.W. Vaandrager. Petri nets models for algebraic theories of concurrency. Proc. PARLE conference, Eindhoven, The Netherlands 1987, Vol. II (Parallel Languages), LNCS 259, pp. 224–242.
10. R. Gorrieri and A. Rensink. Action refinement. *Handbook of Process Algebra,* Elsevier Science, 1047–1147. 2001.
11. C.A.R. Hoare. *Communicating Sequential Processes*, Prentice-Hall, 1985.
12. Michaela Huhn. Action refinement and properties inheritance in systems of sequential agents. CONCUR'96, LNCS 1119, pp. 263–277.

13. Martin Lange and Colin Stirling. Model checking fixed point logic with chop. FOSSACS 2002, LNCS 2303, pp. 250–263.
14. Mila Majster-Cederbaum and Frank Salger. Correctness by construction: towards verification in in hierarchical system development. SPIN 2000, LNCS 1885, pp. 163–180.
15. Mila Majster-Cederbaum and Frank Salger. A priori verification of reactive systems. In Eds T. Bolognesi, D. Latella: Formal Methods for Distributed System Development. pp.35–50. Kluwer Publishers, 2000.
16. Mila Majster-Cederbaum, Naijun Zhan and Harald Fecher. Action refinement from a logical point view. To appear as a research report.
17. R. Milner. Communication and Concurrency. Prentice Hall, 1989.
18. Markus Müller-Olm. A Modal Fixpoint Logic with Chop. STACS'99, LNCS 1563, pp. 510–520.
19. Peter Niebert. A ν-calculus with local views for systems of sequential agents. MFCS'95, LNCS 969, pp.563–573.
20. M. Nielsen, U. Engberg and K.S. Larsen. Fully abstract models for a process language with refinement. Proc. REX School on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, 1989, LNCS 354, pp. 523–548.
21. A. Rensink. *Models and Methods for Action Refinement.* PhD thesis, University of Twente, Enschede, Netherlands, Aug. 1993.
22. J. Sifakis. Research directions for concurrency. *ACM Computing Surveys,* 28(4es):55. 1996.
23. C. Stirling. Modal and temporal logics for processes, Banff Higher Order Workshop 1995, LNCS 1043, pp. 149–237.
24. A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific J. Math.,* 5:285–309, 1955.