ARTICLE IN PRESS

Information and Computation xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

On hierarchically developing reactive systems

Naijun Zhan [a,*,1], Mila Majster-Cederbaum [b]

[a] Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, South Fourth Street, No. 4, Zhong Guan Cun, 100190 Beijing, PR China
[b] Lehrstuhl für Praktische Informatik II, Fakultät für Mathematik und Informatik, Universität Mannheim, D7,27, 68163 Mannheim, Deutschland, Germany

ARTICLE INFO

Article history:
Received 1 June 2006
Revised 12 August 2009
Available online xxx

Keywords:
Action refinement
Modal logics
Specification
Verification
Reactive systems

ABSTRACT

The hierarchical development method is one of the most practical and effective methods for designing large reactive systems by allowing a design at different levels of abstraction. Combining hierarchical specification with hierarchical implementation plays a key role in decreasing the complexity of the verification of these systems. But, up to now, little work has been done related to the topic. In this paper, we investigate this issue.

1. Introduction

Generally speaking, it is not easy to capture the requirements of a complex system at the beginning. *The hierarchical development methodology* allows one to sketch an overall structure of a complex system at the beginning, and then implement each abstract primitive in details step by step. This approach had a great success in sequential programming, and is known as *top-down system specification and analysis technique*. This method is introduced into process algebraic settings in [4,52,44], as *action refinement*. How to relate a hierarchical specification of a complex system to a hierarchical implementation in order to simplify the verification of the system is a challenging problem. In the literature, some first attempts to solve this problem are given, for example, in [27,38,39].

The main results obtained in [27,38,39] 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 action refinement is defined both for the model and specification according to the refinement of a primitive of the abstract model. In [27], Huhn coped with action refinement for models from a semantic point of view, whereas Majster-Cederbaum and Salger in [38,39] dealt with it from a syntactic standpoint, but they both considered *action refinement for the specification* as a form of *syntactical transformation*. Formally speaking, 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, where Q is a finite process, in which there is no recursion. Then the main result of [27,38,39] can be reformulated as follows:

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

[*] Corresponding author.
Email addresses: znj@ios.ac.cn or nj_zhan@iist.unu.edu (N. Zhan), mcb@pi2.informatik.uni-mannheim.de (M. Majster-Cederbaum).
[1] The first author is supported in part by the projects NKBRPC-2002cb312200, NSFC-60493200, NSFC-60721061, NSFC-60573007, NSFC-90718041, NSFC-60736017 and NSFC-60970031.

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.04.002

under some conditions, where $\phi[a \rightsquigarrow Q]$ stands for substituting $\langle a\rangle$ and $[a]$ in ϕ by some formulae of the forms $\langle a_1\rangle\langle a_2\rangle\ldots\langle a_n\rangle$ and $[a_1][a_2]\ldots[a_n]$, respectively, where $\langle a\rangle$ stands for the modality "for some a-successor" and $[a]$ for the modality "for all a-successors", and $a_1a_2\ldots a_n$ 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 $P[a \rightsquigarrow Q]$, that is, we obtain automatically a refined formula that is satisfied by the refined process.

In both approaches, the refinements of the specification and the model are explicitly built on the structure of Q. This restricts the refinement step in three ways:

(i) There are some desired properties of the refined system that cannot be deduced in the setting of [27,38,39]. For example, let $P = a; b + a; c$, $\phi = \langle a\rangle$, $Q = a'; (c'; b'; d' + c'; b')$. It is 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 [27,38,39].
(ii) 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.
(iii) An abstract action can only be refined by a finite process. However, in many applications, an action has to be refined by a process with recursion.

In contrast to the above methods, in this paper we propose a general approach to construct a low-level specification by refining the higher-level specification by providing a formula that a process that is to be substituted for the abstract action a should satisfy. In addition, the refinement may be a process with recursion. The basic idea is to define a refinement mapping Ω which maps the high-level specification ϕ and the property ψ for the refinement of an abstract action a to a lower-level specification by substituting ψ for $\langle a\rangle$ and $[a]$ in ϕ. By choosing ψ appropriately, we can get the desired result. 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* expresses that a "good thing" eventually happens during execution. It was proved in [7] 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 [13]. Meanwhile, the properties of a system can also be classified into *universal* and *existential* and so on. Intuitively, a sound refinement mapping should preserve the type of the property to be refined. Otherwise, the mapping is meaningless since it is impossible to relate hierarchical specifications of a complex system to its hierarchical implementations. For example, $a; b + a; c \models \langle a\rangle; \langle b\rangle$, $a_1; a_2 \models [a_1]; \langle a_2\rangle$, but $(a; b + a; c)[a \rightsquigarrow a_1; a_2] \not\models ([a_1]; \langle a_2\rangle); \langle b\rangle$, since in the high-level specification, $\langle a\rangle; \langle b\rangle$ is an existential property, however its refinement becomes a universal property.

In order to define a refinement mapping, the property ψ for the refinement will be represented by $\psi_1 \wedge \psi_2$, where ψ_1 is a *existential formula*, while ψ_2 is a *universal one*. ψ_1 will be used to substitute for $\langle a\rangle$ and ψ_2 for $[a]$ in ϕ. Such a representation is justified by the result proved in [13].

We prove the following Refinement Theorem:

Theorem (Refinement Theorem). *If some syntactical conditions hold and* $P \models \phi$ *and* $Q \models \psi_1 \wedge \psi_2$, *then* $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi_1, \psi_2, a)$.

The above theorem supports "a priori" verification in the following sense: in the development process we start with $P \models \phi$ and $Q \models \psi_1 \wedge \psi_2$, and either refine P and obtain automatically a (relevant) formula that is satisfied by $P[a \rightsquigarrow Q]$; or, we refine ϕ using $\Omega(\phi, \psi_1, \psi_2, a)$ and obtain automatically a refined process $P[a \rightsquigarrow Q]$ that satisfies the refined specification. Of course such refinement steps may be iterated.

To achieve the intended result, two issues have to be addressed: the first one is to choose an appropriate specification language in which we can express the idea of refinement. A suitable candidate could be the μ-calculus since most modal and temporal logics that are used as specification languages for concurrent systems can be defined in it. Unfortunately, the μ-calculus is not suitable for such a task. For example, suppose that $P \models \langle a\rangle\phi$, and there is no occurrence of $\langle a\rangle$ or $[a]$ in ϕ, and that $Q \models \psi_1 \wedge \psi_2$. After refining a by Q in P, a specification for the refined system that one expects should be naturally $\psi_1; \phi$ which means that the behavior of the system can be divided into two successive segments such that the first segment satisfies ψ_1 and the second one meets ϕ. But $\psi_1; \phi$ is no longer a formula of the μ-calculus. Therefore, here we use FLC [42] as a specification language.

FLC is due to Müller-Olm [42], and is an extension of the μ-calculus by introducing the chop operator ";". FLC is strictly more expressive than the μ-calculus because the former can define non-regular properties [42], whereas, the latter can only express regular properties [21,28]. The model-checking of FLC over finite-state processes was investigated in [31,32].

Informally, $P \models \phi; \psi$ means that the behavior of P can be split 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 $\langle a\rangle; \phi$ is a formula of FLC, after substituting ψ_1 for $\langle a\rangle$, the refined formula $\psi_1; \phi$ is still a formula of FLC. What is more, the refined formula exactly expresses the expected meaning.

The second issue is the atomicity of action refinement for models. The major goal in this work is to establish a correspondence between hierarchical implementations and hierarchical specifications. If we allow that the refinement can be interleaved with the environment some problems will arise. For example, $(a \parallel_{\{\}} b)[a \rightsquigarrow a_1; a_2]$ means the parallel executions of a and b in which a is refined by $a_1; a_2$. It is obvious that $a \parallel_{\{\}} b$ satisfies $\langle a \rangle$, and $a_1; a_2$ satisfies $\langle a_1 \rangle; (\langle a_2 \rangle \wedge [b]; \mathit{ff})$ 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 $\langle a_1 \rangle; (\langle a_2 \rangle \wedge [b]; \mathit{ff})$ after refining a by $a_1; a_2$. However, this becomes false 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 [14,19]. So, in the sequel, we discuss action refinement for models under the assumption of atomicity.

Atomic refinement is quite useful in practice, although the philosophy that atomicity should be given up at certain abstraction level is more popular in the process algebra community. In contrast, in database theory, the user may view his (her) transactions as being executed atomically, although each transaction comprises sequences of actions and may run in parallel with other transactions [20]; in Web-services, it is assumed that the execution of each service is atomic, that is, a service should be either completed or not executed at all. Such atomicity gives rise to lots of troubles in compensation handling [16]; in concurrent programming, e.g., in the Parallel Java Language most refinements are implemented as atomic ones [54].

Preliminary results of this paper were first reported in [40], and further revised in [56] as there exist some technical mistakes in [40]. But in [40,56], we only considered to refine an abstract action by a finite process, without details and proofs. In this paper, we essentially extend our previous results by allowing to refine an abstract action by a process with recursion.

The remainder of this paper is organized as follows: a modeling language is defined in Section 2. Section 3 briefly reviews FLC. A connection between the chop “;" of FLC and the sequential composition “;" of process algebra is investigated in Section 4 which plays a key role in the proof for the Refinement Theorem. A refinement mapping for specifications is defined in Section 5. Section 6 is devoted to establishing the correspondence between the hierarchical specification and the hierarchical implementation of a complex system. In Section 7, we compare our approach with the ones proposed in [27,38,39]. Finally, a brief conclusion is given in Section 8.

2. Modeling language – a TCSP-like process algebra

2.1. Syntax

In this paper, we use a TCSP-like process algebra in combination with an action refinement operator as modeling language. Let Act be an infinite set of (atomic) actions, ranged over by $a, b, c, \ldots$, and A be a subset of Act. Let $\mathcal{X}$ be a set of process variables, ranged over by $x, y, z, \ldots$ The language of processes, denoted by $\mathcal{P}$ and ranged over by $P, Q, \ldots$, is generated by the following grammar:

Definition 1

$$P ::= \delta \mid \mathit{nil} \mid a \mid x \mid P; Q \mid P + Q \mid P \parallel_A Q \mid \mathit{rec}\, x.\, P \mid P[a \rightsquigarrow Q]$$

where $a \in \mathit{Act}$, $A \subseteq \mathit{Act}$, $x \in \mathcal{X}$, and $P, Q \in \mathcal{P}$.

Informally, the constructs of $\mathcal{P}$ can be understood as follows:

- δ is a deadlocked process that cannot proceed.
- nil is a process that does nothing but terminate. The difference between δ and nil is that the former stays in the idle state for ever, while the latter terminates immediately.
- a can execute action a and then evolves to nil.
- x is used to evaluate process terms of the form $\mathit{rec}\, x.\, P$ (see the next section). In isolation, x behaves like δ.
- $P + Q$ denotes the system that can non-deterministically execute the sub-systems P or Q. The non-determinism is resolved by the environment, after that the not selected sub-system is discarded.
- $P; Q$ stands for the system that executes the sub-system P and, upon successful termination of P, proceeds with the execution of Q.
- $P \parallel_A Q$ means that the sub-systems P and Q can be performed concurrently. Each action in A has to be executed synchronously.
- $\mathit{rec}\, x.\, P$ denotes the system that executes the sub-system P recursively, i.e., $\mathit{rec}\, x.\, P = P[\mathit{rec}\, x.\, P/x]$.
- $P[a \rightsquigarrow Q]$ means that the system replaces the execution of the 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.

In the next subsection, we shall formally interpret $\mathcal{P}$.

An occurrence of a process variable $x \in \mathcal{X}$ is called *bound* in a process term P iff it does occur within a sub-term of the form $\mathit{rec}\, x.\, P'$, otherwise called *free*. A process expression P is called *closed* iff all occurrences of each variable occurring in it are bound, otherwise called *open*. We will use $\mathit{fn}(P)$ to stand for the variables that have some free occurrence in P, $\mathit{bn}(P)$ for the variables that have some bound occurrence in P. We say a process P is *terminated*, if it does nothing but terminate (see Definition 2). A variable $x \in \mathcal{X}$ is called *guarded* within a term P iff every occurrence of x is within a sub-term Q where Q

lies in a subexpression $Q^*; Q$ such that Q^* is not terminated. A term P is called *guarded* iff all variables occurring in it are guarded. In the following, we abuse $Act(P)$ to stand for the set of actions which occur in P.

In what follows, we will use $\mathcal{F}$ to stand for the set of closed terms of $\mathcal{P}$ in which neither δ, nor $rec\, x$ occurs, that is, each process in $\mathcal{F}$ does not evolve to deadlock and has finite behavior.

As in [22], we require the following well-formedness conditions on $\mathcal{P}$:

(i) Two operands of $+$ both are either terminated, or non-terminated. That is, termination is deterministic. It is illegal if one operand of $+$ is terminated and the other is non-terminated.
(ii) All process terms are guarded.
(iii) The refinement of an action cannot be a terminated process. As discussed, e.g., in [46], refining an action by a terminated process is not only counter-intuitive but also technically difficult.

2.2. Operational semantics

Here we define an operational semantics for $\mathcal{P}$ in terms of transition systems. The meaning of the constructs of the language can be interpreted in the standard way except for the refinement operator. In order to guarantee the atomicity of a refinement Q, the basic idea is to define a transition system T_1 for the process that may be refined and a transition system T_2 for the refining process Q, and then replace each edge of T_1 labeled by the action to be refined by T_2.

Similar to [22], 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 precedence and must be performed completely. The state language, denoted by $\mathcal{P}^*$, ranged over by $s, \ldots$, is given by:

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

where $a \in Act$, $x \in \mathcal{X}$, $P, Q \in \mathcal{P}$.

It is clear that $\mathcal{P}$ is a proper subset of $\mathcal{P}^*$, i.e., $\mathcal{P} \subset \mathcal{P}^*$.

Note that in the above definition we only admit non-deterministic choice among $P, Q \in \mathcal{P}$. Similarly, it is required that at least one of the two operands of the parallel operator is in $\mathcal{P}$. Moreover, it is counterintuitive that s_2 will be executed earlier than s_1 in the term $s_1; s_2$ if it has a higher priority by having a prefix with more $*$s. Thus, we need the following well-formedness condition:

(iv) In the expression $s_1; s_2$, the number of * in s_2's front is no greater than that of s_1's.

For example, $**a$, $*b; a$, $*(a+b)$ are legal, but not $a; *b$ nor $*a+b$.

Definition 2. Let $\mathcal{T}$ be the least subset of $\mathcal{P}^*$ such that

- $nil \in \mathcal{T}$;
- If $s \in \mathcal{T}$ then $*s \in \mathcal{T}$, $rec\,x.\,s \in \mathcal{T}$, and $s[a \rightsquigarrow Q] \in \mathcal{T}$;
- If $s_1 \in \mathcal{T}$ and $s_2 \in \mathcal{T}$ then $s_1 \parallel_A s_2 \in \mathcal{T}$, $s_1 + s_2 \in \mathcal{T}$ and $s_1; s_2 \in \mathcal{T}$,

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

We say a state s is terminated if $s \in \mathcal{T}$.

Definition 3. Let $\mathcal{A}$ be the least subset of $\mathcal{P}^*$ such that

- If $s \in \mathcal{T}$ then $s \in \mathcal{A}$;
- If $s \in \mathcal{P}$ then $s \in \mathcal{A}$;
- If $s_1 \in \mathcal{A}$ and $s_2 \in \mathcal{A}$ then $s_1 \parallel_A s_2 \in \mathcal{A}$ and $s_1; s_2 \in \mathcal{A}$;
- If $s \in \mathcal{A}$ then $s[a \rightsquigarrow Q] \in \mathcal{A}$,

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

A state s is called *abstract* if $s \in \mathcal{A}$, otherwise, called *concrete*.

We use $\mathcal{T}(s)$ as a shorthand for $s \in \mathcal{T}$ and $\mathcal{A}(s)$ for $s \in \mathcal{A}$ in what follows.

An operational semantics of $\mathcal{P}^*$ is given by the following transition rules:

$$\text{Act} \quad a \xrightarrow{a} nil \qquad\qquad \text{Nd} \quad \frac{P \xrightarrow{a} s}{P+Q \xrightarrow{a} s, \quad Q+P \xrightarrow{a} s}$$

$$\text{Seq-1}\quad \frac{s_1 \xrightarrow{a} s_1'}{s_1; s_2 \xrightarrow{a} s_1'; s_2} \qquad \text{Seq-2}\quad \frac{\mathcal{T}(s_1) \text{ and } s_2 \xrightarrow{a} s_2'}{s_1; s_2 \xrightarrow{a} s_2'}$$

$$\text{Ref-1}\quad \frac{s \xrightarrow{b} s'}{s[a \rightsquigarrow Q] \xrightarrow{b} s'[a \rightsquigarrow Q]}\ a \neq b \qquad \text{Ref-2}\quad \frac{s \xrightarrow{a} s',\quad Q \xrightarrow{a'} s_1}{s[a \rightsquigarrow Q] \xrightarrow{a'} (*s_1); s'[a \rightsquigarrow Q]}$$

$$\text{Rec}\quad \frac{s[rec\,x.\,s/x] \xrightarrow{a} s'}{rec\,x.\,s \xrightarrow{a} s'} \qquad \text{Star}\quad \frac{s \xrightarrow{a} s'}{*s \xrightarrow{a} *s'}$$

$$\text{Asyn-1}\quad \frac{s_1 \xrightarrow{a} s_1'}{s_1 \parallel_A s_2 \xrightarrow{a} s_1' \parallel_A s_2}\quad a \notin A \wedge \mathcal{A}(s_2)$$

$$\text{Asyn-2}\quad \frac{s_2 \xrightarrow{a} s_2'}{s_1 \parallel_A s_2 \xrightarrow{a} s_1 \parallel_A s_2'}\quad a \notin A \wedge \mathcal{A}(s_1)$$

$$\text{Syn}\quad \frac{s_1 \xrightarrow{a} s_1',\ \ s_2 \xrightarrow{a} s_2'}{s_1 \parallel_A s_2 \xrightarrow{a} s_1' \parallel_A s_2'}\quad a \in A \wedge \mathcal{A}(s_1) \wedge \mathcal{A}(s_2)$$

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

We would like to comment on some special rules as follows: The rule Ref-2 states that the residual s_1 of Q is non-interruptible. The rule Star says that $*s$ behaves like s, but the reached state is still concrete (if not properly terminated). The rules Asyn-1 and Asyn-2 give priority to the concrete component. At any time, if a concrete process is in parallel with an abstract process, the latter has to remain idle till the former finishes the execution. Observe that there is no way to reach a state where both components are concrete, starting from an initial abstract state (in fact, such a state would not be well formed). Moreover, if both components are abstract, the rule allows either of them to proceed first. The rule Syn states that only two abstract processes can communicate with each other. The communication between a concrete process and another process may destroy the atomicity of the refinement. In fact, it is impossible to reach a state where a concrete process synchronizes with another process from an initial abstract state. The rules Asyn-1, Asyn-2 and Syn imply that a "concrete" s never executes an action in the synchronization set. In particular, one cannot use an action in a refinement that may synchronize with the environment. The other rules can be conceived as usual. The above rules guarantee that the execution of the refinement Q is not only non-interruptible, but also either executed completely, or not at all.

Note that for any process term $P \in \mathcal{P}^*$, if P satisfies the well-formedness conditions, then all its derivatives produced according to the above rules meet the well-formedness conditions as well.

We use the following example to demonstrate the semantics of $\mathcal{P}^*$:

$$\begin{array}{llll}
b[b \rightsquigarrow d; a; c] \parallel_{\{a\}} a; nil & \xrightarrow{d} & *(a; c); (nil \parallel_{\{a\}} a; nil) & \text{(by Act, Ref-2)} \\
 & \xrightarrow{a} & *c; (nil \parallel_{\{a\}} a; nil) & \text{(by Seq-1)} \\
 & \xrightarrow{c} & (nil \parallel_{\{a\}} a; nil) & \text{(by Seq-2)} \\
 & \equiv & \delta &
\end{array}$$

In addition, we need the following well-formedness condition:

(v) Each state expression s is *image-finite*, i.e., $|\{s' \mid s \xrightarrow{a} s'\}| < \infty$ for any $a \in Act$.

In the following, we investigate the notion of strong bisimulation on $\mathcal{P}^*$.

Definition 4

- A binary symmetric relation R over the closed terms of $\mathcal{P}^*$ is a strong bisimulation if for any $(s_1, s_2) \in R$
  - $\mathcal{T}(s_1)$ iff $\mathcal{T}(s_2)$; and
  - for any $a \in Act$, $s_1 \xrightarrow{a} s_1'$, there exists s_2' s.t. $s_2 \xrightarrow{a} s_2'$ and $(s_1', s_2') \in R$.
- Two closed terms s_1 and s_2 are strongly bisimilar, denoted by $s_1 \sim s_2$, if and only if there exists a strong bisimulation R such that $(s_1, s_2) \in R$.
- Let $E, F \in \mathcal{P}^*$ and $fn(E) \cup fn(F) \subseteq \{x_1, \ldots, x_n\}$. Then $E \sim F$ iff for any closed terms $s_1, \ldots, s_n$, $E\{s_1/x_1, \ldots, s_n/x_n\} \sim F\{s_1/x_1, \ldots, s_n/x_n\}$, if $E\{s_1/x_1, \ldots, s_n/x_n\}$ and $F\{s_1/x_1, \ldots, s_n/x_n\}$ are well formed.

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

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

Because a concrete process has a priority in parallel with an abstract process, $\sim$ is not preserved by $\|_{\mathcal{A}}$. For example, $a_1; a_2 \sim a[a \rightsquigarrow a_1; a_2]$, but $(a_1; a_2) \|_{\{\}} b \not\sim a[a \rightsquigarrow a_1; a_2] \|_{\{\}} b$. However, once we strengthen Definition 4 by adding the following condition:

- $\mathcal{A}(s_1)$ iff $\mathcal{A}(s_2)$,

then the resulting strong bisimulation is called *strong bisimulation* w.r.t. $\mathcal{A}$, and the resulting largest bisimulation is called *strong bisimilarity* w.r.t. $\mathcal{A}$, denoted by $\sim_{\mathcal{A}}$. Obviously, $\sim_{\mathcal{A}}$ is an equivalence relation over $\mathcal{P}^*$.

The following theorem indicates that $\sim_{\mathcal{A}}$ is a congruence too and a proper subset of $\sim$.

Theorem 1. *$\sim_{\mathcal{A}}$ is a congruence over $\mathcal{P}^*$ and $\sim_{\mathcal{A}} \subset \sim$.*

Proof. The proof for the second part is obvious from the definition, so here we only show the first part.

In order to prove that $\sim_{\mathcal{A}}$ is a congruence, we only need to show that if $s_1 \sim_{\mathcal{A}} s_2$, then $C[s_1] \sim_{\mathcal{A}} C[s_2]$ for any context $C[\cdot]$ such that $C[s_1]$ and $C[s_2]$ are well formed. We prove this by case analysis on $C[\cdot]$.

- $C[\cdot] \equiv x, \delta, nil, a$
  It is obvious.
- $C[\cdot] \equiv *\cdot$
  Since $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_{\mathcal{A}}$. Let $R' \widehat{=} \{(*s, *s') \mid (s, s') \in R_{\mathcal{A}}\}$. It is easy to show that R' is a strong bisimulation w.r.t. $\mathcal{A}$.
- $C[\cdot] \equiv \cdot; s$ (the symmetric case is left to readers)
  Since $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_{\mathcal{A}}$. Let $R'_{\mathcal{A}} \widehat{=} R_{\mathcal{A}}; (s, s) \cup Id_s$, where $R; (s, s') \widehat{=} \{(s_1; s, s_2; s') \mid (s_1, s_2) \in R\}$ and Id_s stands for the identity relation on the set of the states of the transition system representing s.
  The symmetry of $R'_{\mathcal{A}}$ can be derived from that of $R_{\mathcal{A}}$ and Id_s. Therefore, in order to prove $R'_{\mathcal{A}}$ is a strong bisimulation w.r.t. $\mathcal{A}$, we only need to prove $R'_{\mathcal{A}}$ is preserved by transitions subject to $\mathcal{A}$. Let $(s'_1, s'_2) \in R'_{\mathcal{A}}$, then (i) $(s'_1, s'_2) \equiv (s_1; s, s_2; s)$ for some $(s_1, s_2) \in R_{\mathcal{A}}$ or (ii) $(s'_1, s'_2) \equiv (s', s')$ for some $(s', s') \in Id_s$. For (i), if $s'_1 \xrightarrow{a} s''_1$, only if $s_1 \xrightarrow{a} s_{11}$ and $s''_1 \equiv s_{11}; s$ by Seq-1 or $\mathcal{T}(s_1)$, $s \xrightarrow{a} s'$, and $s''_1 \equiv s'$ by Seq-2. For the former, because $(s_1, s_2) \in R_{\mathcal{A}}$, there exists s_{21} such that $s_2 \xrightarrow{a} s_{21}$, $\mathcal{A}(s_{11})$ iff $\mathcal{A}(s_{21})$ and $(s_{11}, s_{21}) \in R_{\mathcal{A}}$. By Seq-1, $s_2; s \xrightarrow{a} s_{21}; s$. By the definition of $R'_{\mathcal{A}}$, $(s_{11}; s, s_{21}; s) \in R'_{\mathcal{A}}$ and $\mathcal{A}(s_{11}; s)$ iff $\mathcal{A}(s_{21}; s)$. For the latter, we have $\mathcal{T}(s_2)$ since $\mathcal{T}(s_1)$ and $(s_1, s_2) \in R_{\mathcal{A}}$. By Seq-2, $s_2; s \xrightarrow{a} s'$. It is obvious that $(s', s') \in R'_{\mathcal{A}}$. For (ii), it is easy to show $R'_{\mathcal{A}}$ is closed under transitions subject to $\mathcal{A}$ as well.
- $C[\cdot] \equiv . + P$ (symmetrically, $P + \cdot$ is left to readers), where $P \in \mathcal{P}$.
  If $s_1, s_2 \notin \mathcal{P}$, then $s_1 + P$ and $s_2 + P$ are not well defined. Hence the claim is true. Otherwise, because $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_{\mathcal{A}}$. Let $R'_{\mathcal{A}} \widehat{=} R_{\mathcal{A}} - \{(s_1, s_2)\} \cup \{(s_1 + P, s_2 + P)\} \cup Id_P$. It is easy to show that $R'_{\mathcal{A}}$ is a strong bisimulation w.r.t. $\mathcal{A}$.
- $C[\cdot] \equiv \cdot[a \rightsquigarrow Q]$.
  Since $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_{\mathcal{A}}$. Let $R_1 \widehat{=} \{(s_1[a \rightsquigarrow Q], s_2[a \rightsquigarrow Q]) \mid (s_1, s_2) \in R_{\mathcal{A}}\}$, $R'_{\mathcal{A}} \widehat{=} R_1 \cup \bigcup_{(s_1, s_2) \in R_1} Id_{*Q}; (s_1, s_2) \cup Id_{*Q}$.
  It is easy to show that $R'_{\mathcal{A}}$ is symmetric. So we only need to prove $R'_{\mathcal{A}}$ is closed under transitions subject to $\mathcal{A}$ in order to show that $R'_{\mathcal{A}}$ is a strong bisimulation w.r.t. $\mathcal{A}$. Assume $(s'_1, s'_2) \in R'_{\mathcal{A}}$. Then (i) $\exists (s_1, s_2) \in R_{\mathcal{A}}. s'_1 \equiv s_1[a \rightsquigarrow Q] \wedge s'_2 \equiv s_2[a \rightsquigarrow Q]$; or (ii) $\exists (*Q', *Q') \in Id_{*Q}, \exists (s_1, s_2) \in R_{\mathcal{A}}. s'_1 \equiv *Q'; s_1 \wedge s'_2 \equiv *Q'; s_2$; or (iii) $\exists (*Q', *Q') \in Id_{*Q}. s'_1 \equiv *Q' \wedge s'_2 \equiv *Q'$.
  For (i) $s'_1 \xrightarrow{b} s''_1$ only if (a) $s_1 \xrightarrow{b} s_{11}$ where $b \neq a$, and $s''_1 \equiv s_{11}[a \rightsquigarrow Q]$ by Ref-1; or (b) $s_1 \xrightarrow{a} s_{11}$, $Q \xrightarrow{b} Q'$ and $s''_1 \equiv *Q'; s_{11}[a \rightsquigarrow Q]$ by Ref-2. For (a), since $(s_1, s_2) \in R_{\mathcal{A}}$, there exists s_{21} such that $s_2 \xrightarrow{b} s_{21}$ and $(s_{11}, s_{21}) \in R_{\mathcal{A}}$. By Ref-1, $s'_2[a \rightsquigarrow Q] \xrightarrow{b} s_{21}[a \rightsquigarrow Q]$. From the definition of $R'_{\mathcal{A}}$, $(s_{11}[a \rightsquigarrow Q], s_{21}[a \rightsquigarrow Q]) \in R'_{\mathcal{A}}$; For (b), as $(s_1, s_2) \in R_{\mathcal{A}}$, there exists s_{21} such that $s_2 \xrightarrow{a} s_{21}$ and $(s_{11}, s_{21}) \in R_{\mathcal{A}}$. By Ref-2, $s'_2[a \rightsquigarrow Q] \xrightarrow{b} *Q'; s_{21}[a \rightsquigarrow Q]$. From the definition of $R'_{\mathcal{A}}$, $(*Q'; s_{11}[a \rightsquigarrow Q], *Q'; s_{21}[a \rightsquigarrow Q]) \in R'_{\mathcal{A}}$. Therefore, we have that if $s'_1 \xrightarrow{b} s''_1$, then there exists s''_2 such that $s'_2 \xrightarrow{b} s''_2$, $(s''_1, s''_2) \in R'_{\mathcal{A}}$ and $\mathcal{A}(s''_1)$ iff $\mathcal{A}(s''_2)$. For (ii) and (iii), it can be proved similarly.
- $C[\cdot] \equiv s[a \rightsquigarrow \cdot]$.
  So, there exists $Q_1, Q_2 \in \mathcal{P}$ s.t. $s_1 \equiv Q_1 \wedge s_2 \equiv Q_2$. Because $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_{\mathcal{A}}$. Let $*R_{\mathcal{A}} \widehat{=} \{(*s_1, *s_2) \mid (s_1, s_2) \in R_{\mathcal{A}}\}$. It is easy to prove that $*R_{\mathcal{A}}$ is also a strong bisimulation w.r.t. $\mathcal{A}$. Let $R_1 \widehat{=} \{(s'[a \rightsquigarrow Q_1], s'[a \rightsquigarrow Q_2]) \mid (s', s') \in Id_s\}$, $R'_{\mathcal{A}} \widehat{=} R_1 \cup \bigcup_{(s_1, s_2) \in R_1} *R_{\mathcal{A}}; (s_1, s_2) \cup *R_{\mathcal{A}}$. Similarly to the above case, we can show $R'_{\mathcal{A}}$ is a strong bisimulation w.r.t. $\mathcal{A}$.
- $C[\cdot] \equiv s \|_{\mathcal{A}} \cdot$ (symmetrically, $\cdot \|_{\mathcal{A}} s$ is left to readers).
  For $s_1 \sim_{\mathcal{A}} s_2$, there exists a strong bisimulation $R_{\mathcal{A}}$ w.r.t. $\mathcal{A}$ s.t. $(s_1, s_2) \in R_{\mathcal{A}}$. Let $R'_{\mathcal{A}} \widehat{=} Id_s \cup R_{\mathcal{A}} \cup \{(s' \|_{A} s_1, s' \|_{A} s_2) \mid (s', s') \in Id_s \wedge (s_1, s_2) \in R_{\mathcal{A}} \wedge (\mathcal{A}(s') \vee (\mathcal{A}(s_1) \wedge \mathcal{A}(s_2))\}$. The symmetry of $R'_{\mathcal{A}}$ is easy to get by that of $R_{\mathcal{A}}$ and Id_s.
  Suppose $(s'_1, s'_2) \in R'_{\mathcal{A}}$. Then (i) $\exists (s', s') \in Id_s. s'_1 \equiv s' \wedge s'_2 \equiv s'$; or (ii) $\exists (s_{11}, s_{21}) \in R_{\mathcal{A}}. s'_1 \equiv s_{11} \wedge s'_2 \equiv s_{21}$; or (iii) $\exists (s', s') \in Id_s \exists (s_{11}, s_{21}) \in R_{\mathcal{A}}. s'_1 \equiv s' \|_{A} s_{11} \wedge s'_2 \equiv s' \|_{A} s_{21} \wedge (\mathcal{A}(s') \vee (\mathcal{A}(s_{11}) \wedge \mathcal{A}(s_{12})))$. For (i) and (ii), it is easy

to show that $R'_{\mathcal{A}}$ is closed under transitions subject to $\mathcal{A}$. As for (iii), $s_1 \xrightarrow{a} s''_1$ only if (a) $s' \xrightarrow{a} s'' \wedge a \notin A \wedge ab(s_{11})$ by ASyn-1; or (b) $s_{11} \xrightarrow{a} s'_{11} \wedge a \notin A \wedge ab(s')$ by ASyn-1 again; or (c)$s' \xrightarrow{a} s'' \wedge s_{11} \xrightarrow{a} s'_{11} \wedge a \in A \wedge \mathcal{A}(s') \wedge \mathcal{A}(s_{11})$ according to Syn. For (a), applying ASyn-1, we have $s' \parallel_A s_{21} \xrightarrow{a} s'' \parallel s_{21}$. From the definition of $R'_{\mathcal{A}}$, $(s'' \parallel_A s_{11}, s'' \parallel_A s_{21}) \in R'_{\mathcal{A}}$. Furthermore, it is clear that $\mathcal{A}(s'' \parallel_A s_{11})$ iff $\mathcal{A}(s'' \parallel_A s_{21})$. Similarly, we can prove that in the cases (b) and (c), $R'_{\mathcal{A}}$ is closed under transitions subject to $\mathcal{A}$ as well.
Thus $s \parallel_A s_1 \sim_{\mathcal{A}} s \parallel_A s_2$ since if $\mathcal{A}(s) \vee (\mathcal{A}(s_1) \wedge \mathcal{A}(s_2))$ then $(s \parallel_A s_1, s \parallel_A s_2) \in R'_{\mathcal{A}}$; otherwise $s \parallel_A s_1$ and $s \parallel_A s_2$ both are not well formed.

- $C[\cdot] \equiv rec\,x.\,\cdot$.
Let $R'_{\mathcal{A}} \widehat{=} \{(G\{rec\,x.\,s_1/y\}, G\{rec\,x.\,s_2/y\}) \mid G$ contains at most the variable $y\}$. Using the standard way, induction on the structure of G (see [41]), we can prove $R'_{\mathcal{A}}$ is a strong bisimulation w.r.t. $\mathcal{A}$. Taking $G \equiv y$, it follows that $rec\,x.\,s_1 \sim_{\mathcal{A}} rec\,x.\,s_2$. □

Convention: From now on, we do not distinguish concrete processes and abstract processes, and uniformly use $P, Q, \ldots$ to stand for them.

Definition 5. A process P is said to be *normed* if for any derivative P' of P, P' can reach a terminated state in finitely many steps.

Note that a normed process could be evolved to a deadlock or a livelock (divergence).

3. Specification language – a fixpoint logic with chop (FLC)

FLC is an extension of the modal μ-calculus by introducing the chop operator ";", which can express non-regular properties [42]. It is therefore strictly more powerful than the μ-calculus, since it was proved in [21] that only regular properties can be defined in the μ-calculus, while in [28] the converse was proved. Informally, $P \models \phi_1; \phi_2$ means that any execution of P can be divided into two successive segments such that the first satisfies ϕ_1 and the second meets ϕ_2. In addition, τ [2] was introduced into FLC too, interpreted as the identity function, which plays the role of neutral element of the chop operator ";".
For our purpose we present FLC slightly differently from its original version [42]. The differences lie in the following two points:

(i) In our presentation, FLC only contains three special propositional constants, i.e., tt, ff and $\surd$. The first two are as usual and the last one is used to characterize terminated processes. FLC contains a set of propositional letters in [42].
(ii) In our setting, $[a]$ is satisfied only by non-terminated processes in contrast to that any process satisfies it in [42]. The aim is to distinguish between terminated processes and deadlocked processes in FLC, e.g., $\surd$ characterizes all terminated processes and does not hold for any deadlocked process, while $\bigwedge_{a \in Act}[a]; ff$ characterizes all deadlocked processes, but does not hold for any terminated process.

3.1. Syntax and semantics

Let $X, Y, Z, \ldots$ range over an infinite set *Var* of *variables*, tt and ff be two propositional constants as usual, and $\surd$ be another one that is used to indicate if a process is terminated.
Formulae of FLC are generated according to the following grammar:

$$\phi ::= tt \mid ff \mid \surd \mid \tau \mid X \mid [a] \mid \langle a \rangle \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1; \phi_2 \mid \mu X.\phi \mid \nu X.\phi$$

where $X \in Var$ and $a \in Act$.
In the sequel, we use @ to stand for $\langle a \rangle$ or $[a]$, p for tt, ff or $\surd$, and σ for ν or μ, $Act(\phi)$ for all actions that occur in ϕ.
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 occurrences* of variables in a formula as well as *closed* and *open* formulae etc. We will use $fn(\phi)$ to stand for the set of the variables which have free occurrence in ϕ and $bn(\phi)$ for the set of the variables that have bound occurrence in ϕ.

Definition 6. In the following, we define what it means for a formula to be a *guard*:

(1) @ and p are guards;
(2) if ϕ and ψ are guards, so are $\phi \wedge \psi$ and $\phi \vee \psi$;
(3) if ϕ is a guard, so are $\phi; \psi$ and $\sigma X.\phi$, where ψ is any formula of FLC .

[2] In [42], τ is written as *term*.

X is said to be *guarded* in ϕ if each occurrence of X is within a subformula ψ that is a guard. If all variables in $fn(\phi) \cup bn(\phi)$ are guarded, then ϕ is called *guarded*.

In what follows we denote by $\mathcal{L}_{\text{FLC}}$ the set of formulae of FLC that are closed and guarded.

FLC is interpreted over labeled transition systems $T = (\mathcal{S}, A, \rightarrow)$, where $\mathcal{S} \subseteq \mathcal{P}^*$, $A \subseteq Act$, and $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$. A formula is interpreted as a *monotonic predicate transformer*, which is simply a mapping $f : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$ that is monotonic w.r.t. the inclusion ordering on $2^{\mathcal{S}}$. We use MPT_T to represent all these monotonic predicate transformers over $\mathcal{S}$. MPT_T together with the inclusion ordering defined by

$$f \subseteq f' \text{ iff } f(\mathcal{A}) \subseteq f'(\mathcal{A}) \text{ for all } \mathcal{A} \subseteq \mathcal{S}$$

forms a complete lattice. We denote the join and meet operators by $\sqcup$ and $\sqcap$. By Tarski-Knaster Theorem [50], the least and greatest fixed points of monotonic functions: $(2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}) \rightarrow (2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}})$ exist. They are used to interpret fixed point formulae of FLC.

tt and *ff* are interpreted in the standard way, i.e., by $\mathcal{S}$ and $\emptyset$, respectively. The meaning of $\surd$ is to map any subset of $\mathcal{S}$ to the set that consists of all terminated processes in $\mathcal{S}$. So, a process P meets $\surd$ iff $\mathcal{T}(P)$. τ is interpreted as the identity mapping. Because *nil* and δ have different behavior in the presence of ;, they should be distinguished in FLC . To this end, $[a]$ is interpreted as a function that maps a set of processes $\mathcal{A}$ to the set in which each process is not terminated and any a-successor of the process must be in $\mathcal{A}$. This is different from its original interpretation in [42]. Hence, according to our interpretation, $P \models [a]$ only if $\neg\mathcal{T}(P)$. On the contrary, in [42] it is always valid that $P \models [a]$ for any $P \in \mathcal{P}^*$. Thus, it is easy to show that $nil \not\models \bigwedge_{a\in Act}[a]; ff$, while $\bigwedge_{a\in Act}[a]; ff$ is the characteristic formula of δ. The chop operator ; will be interpreted as the function composition operator. The meaning of variables is given by an *environment* $\rho : var \rightarrow (2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}})$ that assigns variables to monotonic functions of sets to sets. $\rho[X \rightsquigarrow f]$ agrees with ρ except for assigning f to X.

Definition 7. Formally, the meaning of a formula ϕ under a given labeled transition system $T = (\mathcal{S}, A, \rightarrow)$ and valuation ρ, denoted by $\mathcal{C}_T^\rho(\phi)$, is inductively defined as follows:

$$\begin{aligned}
\mathcal{C}_T^\rho(tt)(\mathcal{A}) &= \mathcal{S}\\
\mathcal{C}_T^\rho(ff)(\mathcal{A}) &= \emptyset\\
\mathcal{C}_T^\rho(\surd)(\mathcal{A}) &= \{P \mid P \in \mathcal{S} \wedge \mathcal{T}(P)\}\\
\mathcal{C}_T^\rho(\tau)(\mathcal{A}) &= \mathcal{A}\\
\mathcal{C}_T^\rho(X) &= \rho(X)\\
\mathcal{C}_T^\rho([a])(\mathcal{A}) &= \{P \mid \neg\mathcal{T}(P) \wedge \forall P' : P \xrightarrow{a} P' \Rightarrow P' \in \mathcal{A}\}\\
\mathcal{C}_T^\rho(\langle a\rangle)(\mathcal{A}) &= \{P \mid \exists P' : P \xrightarrow{a} P' \wedge P' \in \mathcal{A}\}\\
\mathcal{C}_T^\rho(\phi_1 \wedge \phi_2)(\mathcal{A}) &= \mathcal{C}_T^\rho(\phi_1)(\mathcal{A}) \cap \mathcal{C}_T^\rho(\phi_2)(\mathcal{A})\\
\mathcal{C}_T^\rho(\phi_1 \vee \phi_2)(\mathcal{A}) &= \mathcal{C}_T^\rho(\phi_1)(\mathcal{A}) \cup \mathcal{C}_T^\rho(\phi_2)(\mathcal{A})\\
\mathcal{C}_T^\rho(\phi_1; \phi_2) &= \mathcal{C}_T^\rho(\phi_1) \cdot \mathcal{C}_T^\rho(\phi_2)\\
\mathcal{C}_T^\rho(\mu X.\phi) &= \sqcap\{f \in \text{MPT}_\text{T} \mid \mathcal{C}_T^{\rho[X\rightsquigarrow f]}(\phi) \subseteq f\}\\
\mathcal{C}_T^\rho(\nu X.\phi) &= \sqcup\{f \in \text{MPT}_\text{T} \mid \mathcal{C}_T^{\rho[X\rightsquigarrow f]}(\phi) \supseteq f\}
\end{aligned}$$

where $\mathcal{A} \subseteq \mathcal{S}$, and $\cdot$ stands for the compositional operator over functions.

The set of processes *satisfying* a given formula ϕ under the given environment ρ is $\mathcal{C}_T^\rho(\phi)(\mathcal{S})$. A process P is said to satisfy ϕ iff $P \in \mathcal{C}_T^\rho(\phi)(\mathcal{S})$ for some environment ρ, denoted by $P \models \phi$. We also abuse $\phi(\mathcal{A})$ to stand for $\mathcal{C}_T^\rho(\phi)(\mathcal{A})$ if T and ρ are clear from the context. $\phi \Rightarrow \psi$ means that $\mathcal{C}_T^\rho(\phi)(\mathcal{A}) \subseteq \mathcal{C}_T^\rho(\psi)(\mathcal{A})$ for any T and $\mathcal{A} \subset \mathcal{S}_T$ and any ρ. $\phi \Leftrightarrow \psi$ means $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$. Note that $\phi \Leftrightarrow \psi$ is stronger than $P \models \phi$ iff $P \models \psi$.

The following lemma says that if X does not have any free occurrence in ϕ, then the meaning of ϕ is independent of X.

Lemma 2. *If $X \notin fn(\phi)$, then $\mathcal{C}_T^\rho(\phi)(\mathcal{A}) = \mathcal{C}_T^{\rho[X\rightsquigarrow f]}(\phi)(\mathcal{A})$ for any environment ρ, $\mathcal{A} \subseteq \mathcal{S}$ and $f \in \text{MPT}_\text{T}$.*

The lemma below claims that the interpretation of a formula ϕ is monotonic on environments, i.e.,

Lemma 3. *If $\rho \subseteq \rho'$, then $\mathcal{C}_T^\rho(\phi)(\mathcal{A}) \subseteq \mathcal{C}_T^{\rho'}(\phi)(\mathcal{A})$ for any ϕ of FLC and $\mathcal{A} \subseteq S$, where $\rho \subseteq \rho'$ means for any X and $\mathcal{A} \subseteq \mathcal{S}$, $\rho(X)(\mathcal{A}) \subseteq \rho'(X)(\mathcal{A})$.*

In the sequel, in order to avoid the excessive use of parentheses, we assume ; has higher priority than $\vee$ and $\wedge$, $\vee$ and $\wedge$ have higher priority than νX and μX, while νX and μX have higher priority than $\Rightarrow$ and $\Leftrightarrow$.

3.2. Approximant of FLC

In this subsection, we introduce an approximant of FLC, denoted by AFLC, whose formulae are constructed by

$$\phi ::= p \mid \tau \mid X \mid @ \mid \bigwedge_{i\in I} \phi_i \mid \bigvee_{i\in I} \phi_i \mid \phi;\phi \mid \mu^\alpha X.\phi \mid \nu^\alpha X.\phi \mid \sigma X.\phi$$

where $i, I, \alpha \in \mathcal{O}n$, the ordinals.

For convenience, in what follows we use $\phi\{\psi/\chi\}$ to stand for substituting ψ for each occurrence of χ in ϕ, where $\chi \in \{X, \surd, \langle a\rangle, [a]\}$.

Given a labeled transition system T and an environment ρ, those formulae of AFLC that are also of FLC can be interpreted as in the previous subsection, the additional parts are interpreted as follows:

$$\begin{aligned}
\mathcal{C}_T^\rho(\bigwedge_{i\in I}\phi_i)(\mathcal{A}) &= \bigcap_{i\in I}\mathcal{C}_T^\rho(\phi_i)(\mathcal{A}),\\
\mathcal{C}_T^\rho(\bigvee_{i\in I}\phi_i)(\mathcal{A}) &= \bigcup_{i\in I}\mathcal{C}_T^\rho(\phi_i)(\mathcal{A}),\\
\mathcal{C}_T^\rho(\mu^0X.\phi_1) &= \mathcal{C}_T^\rho(ff),\\
\mathcal{C}_T^\rho(\mu^{\alpha+1}X.\phi_1) &= \mathcal{C}_T^\rho(\phi_1\{\mu^\alpha X.\phi_1/X\}),\\
\mathcal{C}_T^\rho(\mu^\lambda X.\phi_1) &= \mathcal{C}_T^\rho(\bigvee_{\alpha\in\lambda}\mu^\alpha X.\phi_1),\\
\mathcal{C}_T^\rho(\nu^0X.\phi_1) &= \mathcal{C}_T^\rho(tt),\\
\mathcal{C}_T^\rho(\nu^{\alpha+1}X.\phi_1) &= \mathcal{C}_T^\rho(\phi_1\{\nu^\alpha X.\phi_1/X\}),\\
\mathcal{C}_T^\rho(\nu^\lambda X.\phi_1) &= \mathcal{C}_T^\rho(\bigwedge_{\alpha\in\lambda}\nu^\alpha X.\phi_1),
\end{aligned}$$

where $\mathcal{A} \subseteq \mathcal{S}, \alpha, \lambda \in \mathcal{O}n$ is a limit ordinal.

By Tarski and Knaster's Theorem, it is clear that

Proposition 1. *For any $\mu X.\phi$, there exists an ordinal α such that $\mu X.\phi \Leftrightarrow \mu^\alpha X.\phi$. Analogously, for any $\nu X.\phi$. there exists an ordinal α such that $\nu X.\phi \Leftrightarrow \nu^\alpha X.\phi$.*

It is therefore easy to see that FLC is a fragment of AFLC.

As pointed out in [31], if only finite state processes are concerned, α can be replaced by ω. Moreover, if P is a finite-state process then $P \models \sigma X.\phi \Leftrightarrow \sigma^k X.\phi$, where k is the number of the states or processes reachable from P. Similarly, we can show that α in Proposition 1 can be replaced by 2^{ω_1}, where ω_1 stands for the first uncountable limit ordinal, because the cardinality of

$$\{f : 2^S \mapsto 2^S \mid f \text{ is a monotonic w.r.t. the inclusion ordering on } 2^S\}$$

is at most 2^{ω_1}.

Some notions of FLC like *guardedness*, *free* and *bound occurrence* of variable can be easily extended to AFLC.

We will use $\mathcal{L}_{\text{AFLC}}$ to stand for the set of all formulae of AFLC that are closed and guarded.

From now on, all formulae are referred to be in AFLC if not otherwise stated.

3.3. Some theorems concerning FLC *and AFLC*

Some properties for FLC have been shown in [31,42], e.g., FLC is strictly more expressive than the μ-calculus; the model-checking problem of FLC is decidable for finite-state processes, undecidable for context-free processes; the satisfiability and validity of FLC are undecidable; FLC does not enjoy the finite-model property. However, FLC has the tree model property,

Theorem 2 (Lange and Stirling, [31]).[3] *Given $P, Q \in \mathcal{P}^*$, $P \sim Q$ iff for any closed formula ϕ of FLC (AFLC), $P \models \phi$ iff $Q \models \phi$.*

[3] Although we modified FLC here, the proof for the tree model property in [31] still works in our case.

The following lemma follows directly from the definitions of semantics of FLC and AFLC.

Lemma 4

N	$\tau;\phi \Leftrightarrow \phi;\tau \Leftrightarrow \phi$	P1	$p;\phi \Leftrightarrow p$
P2	$\langle a\rangle;ff \Leftrightarrow ff$	T1	$\surd \vee [a];tt \Leftrightarrow tt$
T2	$\surd \wedge [a];tt \Leftrightarrow ff$	C	$(\phi;\psi);\varphi \Leftrightarrow \phi;(\psi;\varphi)$
IC	$(\bigwedge_{i\in I}\phi_i);\varphi \Leftrightarrow \bigwedge_{i\in I}(\phi_i;\varphi)$	DC	$(\bigvee_{i\in I}\phi_i);\varphi \Leftrightarrow \bigvee_{i\in I}(\phi_i;\varphi)$

A formula ϕ is called a *propositional normal form* (PNF for short) if it does not contain any subformula of the form $p;\psi$, or $\tau;\psi$, or $\psi;\tau$.

Lemma 5. *For any given formula ϕ, there is another formula ϕ' which is PNF such that $\phi \Leftrightarrow \phi'$.*

Thus, from now on, we assume that all formulae are PNF if not otherwise stated.
From Definition 7, the following proposition is obvious.

Proposition 2. *If $\phi \Rightarrow \psi$ then $\phi;\varphi \Rightarrow \psi;\varphi$ and $\varphi;\phi \Rightarrow \varphi;\psi$.*

Definition 8. Given a formula ϕ, we define its beginning atomic sub-formulae, denoted $\mathrm{BSub}(\phi)$, as:

$$\mathrm{BSub}(\phi) \hat{=} \begin{cases} \{\phi\} & \text{if } \phi = p, X, @, \tau \\ \bigcup_{\mathrm{i}\in\mathrm{I}} \mathrm{BSub}(\phi_{\mathrm{i}}) & \text{if } \phi = \bigwedge_{i\in I}\phi_i \text{ or } \phi = \bigvee_{i\in I}\phi_i \\ \mathrm{BSub}(\phi_1) & \text{if } \phi = \phi_1;\phi_2 \text{ and } \tau \notin \mathrm{BSub}(\phi_1) \\ (\mathrm{BSub}(\phi_1) - \{\tau\}) \cup \mathrm{BSub}(\phi_2) & \text{if } \phi = \phi_1;\phi_2 \text{ and } \tau \in \mathrm{BSub}(\phi_1) \\ \mathrm{BSub}(\phi_1) & \text{if } \phi = \sigma^\alpha X.\phi_1 \end{cases}$$

Symmetrically, we define its ending atomic sub-formulae, denoted $\mathrm{ESub}(\phi)$, as:

$$\mathrm{ESub}(\phi) \hat{=} \begin{cases} \{\phi\} & \text{if } \phi = p, X, @, \tau \\ \bigcup_{\mathrm{i}\in\mathrm{I}} \mathrm{ESub}(\phi_{\mathrm{i}}) & \text{if } \phi = \bigwedge_{i\in I}\phi_i \text{ or } \phi = \bigvee_{i\in I}\phi_i \\ \mathrm{ESub}(\phi_2) & \text{if } \phi = \phi_1;\phi_2 \text{ and } \tau \notin \mathrm{ESub}(\phi_2) \\ (\mathrm{ESub}(\phi_2) - \{\tau\}) \cup \mathrm{ESub}(\phi_1) & \text{if } \phi = \phi_1;\phi_2 \text{ and } \tau \in \mathrm{ESub}(\phi_2) \\ \mathrm{ESub}(\phi_1) & \text{if } \phi = \sigma^\alpha X.\phi_1 \end{cases}$$

If $BSub(\phi)$ is singleton, then we say that ϕ has only unique beginning atomic sub-formula, denoted $USF(\phi)$.

Example 1. It is easy to see that $\mathrm{ESub}(\langle a\rangle;tt) = \{tt\}$, while $\mathrm{ESub}(\langle a\rangle) = \{\langle a\rangle\}$. Furthermore,

$$\mathrm{BSub}((\langle a'\rangle \vee \tau);(\langle a\rangle;\langle b\rangle \wedge [c];\langle e\rangle;[f]);(\tau \vee [b'])) = \{\langle a\rangle', \langle a\rangle, [c]\}, \text{ whereas}$$
$$\mathrm{ESub}((\langle a'\rangle \vee \tau);(\langle a\rangle;\langle b\rangle \wedge [c];\langle e\rangle;[f]);(\tau \vee [b'])) = \{\langle b\rangle, [f], [b']\}.$$

Lemma 6. *For any $\phi \in \mathcal{L}_{\mathrm{FLC}}(\mathcal{L}_{\mathrm{AFLC}})$, there exists $\psi \in \mathcal{L}_{\mathrm{FLC}}(\mathcal{L}_{\mathrm{AFLC}})$ of the form $\bigvee_{i=1}^{n}(\bigwedge_{j=1}^{n_i}\psi_{i,j})$ such that $\phi \Leftrightarrow \psi$, $BSub(\phi) = BSub(\psi)$ and $USF(\psi_{i,j})$ for all $1 \le i \le n$ and $1 \le j \le n_i$.*

Proof. Using Lemma 4, by induction on ϕ. □

Definition 9. A formula ϕ is said to be *existential* if for any $a \in Act$, $[a] \notin BSub(\phi)$. We use $\mathcal{EF}$ to stand for the set of existential formulae. Dually, a formula ϕ is said to be *universal* if for any $a \in Act$, $\langle a\rangle \notin BSub(\phi)$. We use $\mathcal{UF}$ to stand for the set of universal formulae. A formula is called a *property formula* if $\phi \Leftrightarrow \phi_1 \wedge \phi_2$, where $\phi_1 \in \mathcal{EF}$ and $\phi_2 \in \mathcal{UF}$. The set of property formulae is denoted by $\mathcal{PF}$.

According to Definition 9, the following propositions are obvious.

Proposition 3. *$\mathcal{EF}, \mathcal{UF}$ are closed under all operators of the logic. That is, $\bigwedge_{i\in I}\phi_i \in \mathcal{EF}(\mathcal{UF})$ and $\bigvee_{i\in I}\phi_i \in \mathcal{EF}(\mathcal{UF})$ iff $\phi_i \in \mathcal{EF}(\mathcal{UF})$ for any $i \in I$; $\phi_1;\phi_2 \in \mathcal{EF}(\mathcal{UF})$ if $\phi_1,\phi_2 \in \mathcal{EF}(\mathcal{UF})$; $\sigma^\alpha X.\phi, \sigma X.\phi \in \mathcal{EF}(\mathcal{UF})$ iff $\phi \in \mathcal{EF}(\mathcal{UF})$.*

Proposition 4

1. *If $\phi \in \mathcal{EF}$ and $\tau \notin BSub(\phi)$, then for any formula φ of AFLC, $\phi; \varphi \in \mathcal{EF}$.*
2. *If $\phi \in \mathcal{UF}$ and $\tau \notin BSub(\phi)$, then for any formula φ of AFLC, $\phi; \varphi \in \mathcal{UF}$.*

Proposition 5

1. *For any closed formula $\phi \in \mathcal{UF}$, if $P_1 \models \phi$, $P_2 \models \phi$ and $P_1 + P_2$ is well formed, then $P_1 + P_2 \models \phi$.*
2. *For any closed formula $\phi \in \mathcal{EF}$, if $P_1 \models \phi$ and $P_1 + P_2$ is well formed, then $P_1 + P_2 \models \phi$.*

Suppose that $c, a \in Act$ are two different actions. Then $c; P \models [a]; \phi$ for arbitrary P and ϕ. The following lemma generalizes this observation.

Lemma 7. *Let ϕ be a closed formula in $\mathcal{UF}$. If $c \notin Act(\phi)$, $\phi \not\Leftrightarrow ff$ and $\tau, \surd \notin BSub(\phi)$, then $c; P \models \phi; \varphi$.*

Proof. By induction on ϕ.

- $\phi = ff, \langle a \rangle, \tau$
  Impossible according to the condition and the definition of $\mathcal{UF}$.
- $\phi = tt$
  It is trivial by Definition 7.
- $\phi = [a]$
  As $c \notin Act(\phi)$, it follows that $a \neq c$. According to Definition 7, it is easy to see that $c; P \models [a]; \varphi$.
- $\phi = \bigwedge_{i \in I} \phi_i$
  Because $\phi \in \mathcal{UF}$, $\phi \not\Leftrightarrow ff$ and $\tau, \surd \notin BSub(\phi)$, we obtain that $\phi_i \in \mathcal{UF}$, $\phi_i \not\Leftrightarrow ff$ and $\tau, \surd \notin BSub(\phi_i)$ for each $i \in I$. Moreover, it can be shown that $c \notin Act(\phi_i)$ as $c \notin Act(\phi)$ for all $i \in I$. Thus, by the induction hypothesis, it derives that $c; P \models \phi_i; \varphi$, i.e., $c; P \models (\bigwedge_{i \in I} \phi_i); \varphi$ by IC.
- $\phi = \bigvee_{i \in I} \phi_i$
  Similarly to the above case.
- $\phi = \phi_1; \phi_2$
  By Lemma 6, without loss of generality,
  $$\phi_1 \Leftrightarrow \bigvee_{i=1}^{k} \left(\tau \wedge \bigwedge_{j=1}^{n_i} \psi_{i,j}\right) \vee \bigvee_{i=k+1}^{m} \left(\bigwedge_{j=1}^{n_i} \psi_{i,j}\right)$$
  where $USF(\psi_{i,j})$ and $\tau \notin BSub(\psi_{i,j})$ for all $1 \leq i \leq m$ and $1 \leq j \leq n_i$. Now, we consider the following two cases:
  1. $m > k$ and there exists $h > k$ such that $\bigwedge_{j=1}^{n_h} \psi_{h,j} \not\Leftrightarrow ff$. Thus, $\phi_1; \phi_2 \Leftrightarrow \bigvee_{i=1}^{k}(\tau \wedge \bigwedge_{j=1}^{n_i} \psi_{i,j}); \phi_2 \vee \bigvee_{i=k+1}^{m}(\bigwedge_{j=1}^{n_i} \psi_{i,j}); \phi_2$ by DC. It is clear that $(\bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2 \in \mathcal{UF}$, $c \notin Act((\bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2)$ and $\tau, \surd \notin BSub((\bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2)$, hence we get $c; P \models (\bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2; \varphi$ by the induction hypothesis. Therefore, it follows that $c; P \models (\phi_1; \phi_2); \varphi$ by DC and Definition 7.
  2. Otherwise, there exists $1 \leq h \leq k$ such that $\bigwedge_{j=1}^{n_h} \psi_{h,j} \not\Leftrightarrow ff$ and $\phi_2 \not\Leftrightarrow ff$ since $\phi_1; \phi_2 \not\Leftrightarrow ff$. Besides, we know that $\phi_2 \in \mathcal{UF}$ and $\tau, \surd \notin BSub(\phi_2)$ by the definition and the assumption. As $c \notin Act(\phi)$, we have $c \notin Act(\phi_2)$. Therefore, by the induction hypothesis, we obtain $c; P \models (\bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2; \varphi$ and $c; P \models \phi_2; \varphi$. Using N and IC, we have $c; P \models (\tau \wedge \bigwedge_{j=1}^{n_h} \psi_{h,j}); \phi_2; \varphi$. According to Definition 7, we get $c; P \models (\phi_1; \phi_2); \varphi$.
- $\phi = \nu^{\alpha} X.\phi'$
  This case proceeds by induction on α.
  $\underline{\alpha = 0}$ It is trivial.
  $\underline{\alpha = \lambda}$, where λ is a limit ordinal
  Since $\nu^{\lambda} X.\phi' \in \mathcal{UF}$, $\phi \not\Leftrightarrow ff$, $c \notin Act(\nu^{\lambda} X.\phi')$ and $\tau, \surd \notin BSub(\nu^{\lambda} X.\phi')$, we obtain that $\nu^{\beta} X.\phi'$ meets the listed requirements as well for each $\beta < \lambda$. By the local induction hypothesis, we have for any $\beta < \lambda$, $c; P \models (\nu^{\beta} X.\phi'); \varphi$. By IC, it follows that $c; P \models (\nu^{\lambda} X.\phi'); \varphi$.
  $\underline{\alpha = \beta + 1}$
  Thus, $\nu^{\beta+1} X.\phi' \Leftrightarrow \phi'\{\nu^{\beta} X.\phi'\}$. It is easy to see that $\phi' \in \mathcal{UF}$, $c \notin Act(\phi'\{\nu^{\beta} X.\phi'\})$, $\phi'\{\nu^{\beta} X.\phi'\} \not\Leftrightarrow ff$, $\surd \notin BSub(\phi'\{\nu^{\beta} X.\phi'\})$ according to the assumption. We will prove this sub-case by the case analysis on the structure of ϕ'.
  - $\phi' = tt, \tau, [a]$
    It is obvious.
  - $\phi' = X$
    It is straightforward from the local induction hypothesis.

- $\phi' = \bigwedge_{i\in I}\phi'_i$ or $\bigvee_{i\in I}\phi'_i$
  This sub-sub-case can be proved same as the cases when $\phi = \bigwedge_{i\in I}\phi_i$ and $\phi = \bigvee_{i\in I}\phi_i$, respectively.
- $\phi' = \phi'_1; \phi'_2$
  This sub-sub-case can be proved same as the case when $\phi = \phi_1; \phi_2$.
- $\phi' = \sigma^\alpha X.\phi''$
  This sub-sub-case can be proved same as the cases when $\phi = \nu^\alpha X.\phi'$ and $\phi = \mu^\alpha X.\phi'$.

- $\phi = \mu^\alpha X.\phi'$
  Similarly to the above case.
- $\phi = \sigma X.\phi'$
  This case can be reduced to the above two cases by Proposition 1. □

By Lemma 7, the following corollary is immediate simply by instantiating φ with τ.

Corollary 1. *Let ϕ be closed in $\mathcal{UF}$. If $c \notin Act(\phi)$, $\phi \not\Leftrightarrow ff$ and $\surd, \tau \notin BSub(\phi)$, then $c; P \models \phi$.*

In order to ease proofs by induction on formulae, we need to define a well-founded order on the formulae of AFLC, denoted by $<$. To this end, we first define a partial order, denoted by $\prec$, on AFLC $\times$ AFLC as: $(\phi_1, \phi_2) \prec (\psi_1, \psi_2)$ iff $\phi_1; \phi_2 \Leftrightarrow \psi_1; \psi_2$ and ϕ_1 is a proper subformula of ψ_1. In other words, we assume the left association of ; has a higher precedence. For example, $(\langle a\rangle, \langle b\rangle; \langle c\rangle) \prec (\langle a\rangle; \langle b\rangle, \langle c\rangle)$. Then, we say $\phi < \psi$ iff either ϕ is a proper subformula of ψ, or $\phi \prec \psi$. It is easy to see that $<$ is well founded.

4. A connection between the chop and the sequential composition

In this section, we study the relation between the chop ";" of FLC and the sequential composition ";" of process algebra that plays a key role in the proof for our main theorem presented in Section 6.

In general, although $P \models \phi$ and $Q \models \psi$, one cannot deduce that $P; Q \models \phi; \psi$ because possibly ϕ only describes incomplete executions of P. For example, let $P = a; b, Q = c; d$. It is obvious that $P \models \langle a\rangle$ and $Q \models \langle c\rangle$, but $P; Q \not\models \langle a\rangle; \langle c\rangle$. Therefore, we require that ϕ must specify the full executions of P. This is similar to the premise of the rule Seq-2 that in the process $P; Q$, only after the first segment P finishes executing, then Q can start to run.

Note that here a full execution of a process P means one of its runs, not a trace of the process. For example, $\underbrace{aaaa\cdots}_{\text{infinitely many}}$ is a full execution of the process $rec\,x.\,a; x$, but not a^n, for any $n \in \mathbb{N}$. Hence $\nu X.\langle a\rangle; X$ specifies the full executions of $rec\,x.\,a; x$, but $\mu X.[a]; X$ does not, because $\nu X.\langle a\rangle; X$ expresses that there is at least an infinite a-run, while $\mu X.[a]; X$ says that all a-runs are finite, thus $rec\,x.\,a; x \models (\nu X.\langle a\rangle; X); \surd$, and $rec\,x.\,a; x \not\models (\mu X.[a]; X); \surd$.

Another issue is that by the definition of the semantics of $\mathcal{P}^*$, $nil; P \sim P$. Therefore, the properties concerning intermediate terminations should be omitted in the resulting formula. Otherwise, the resulting property does not hold in the combined system. For example, let $P = a; nil$ and $Q = b; \delta$, $\phi = \langle a\rangle; \surd$, and $\psi = \langle b\rangle$. It is obvious that $P \models \phi; \surd$ and $Q \models \psi$ but $P; Q \not\models \phi; \psi$. This is because nil is a neutral element of the sequential composition in process algebra, but $\surd$ is not a neutral element of the corresponding chop ";" in the logic. To solve this problem, we will replace every occurrence of $\surd$ in ϕ with τ in the resulting formulae, i.e., $\phi\{\tau/\surd\}; \psi$. Because τ is a neutral element of the chop, this is in accordance with that nil being a neutral element of the sequential composition [5].

Additionally, according to P1, when calculating the meaning of formula ϕ, any sub-formula φ appearing in the context of $\surd; \varphi$ will be discarded, but the sub-formula will be picked up during interpreting $\phi\{\tau/\surd\}; \psi$. This will give rise to troubles. For example, $nil \models \surd; [a]; \langle b\rangle$ and $a; c \models \langle a\rangle; \langle c\rangle$, but $nil; (a; c) \not\models (\tau; [a]; \langle b\rangle); (\langle a\rangle; \langle c\rangle)$. So, we require that ϕ is *propositional normal form*. In fact, such a requirement is reasonable by Lemma 5.

In a word, we have the following connection between the chop of FLC and the sequential composition of process algebra:

Theorem 3. *Assume $\phi, \psi \in \mathcal{L}_{\text{AFLC}}$, which are PNF. If $P \models \phi; \surd$ and $Q \models \psi$, then $P; Q \models \phi\{\tau/\surd\}; \psi$.*

Proof. The proof proceeds by induction on the structure of ϕ w.r.t. $<$.

Base cases:

- $\phi = \tau$
  Since $P \models \tau; \surd$, it follows that $P \models \surd$ by N. Hence, $\mathcal{T}(P)$. Thus, $P; Q \models \phi\{\tau/\surd\}; \psi$ by Seq-2, N and the assumption $Q \models \psi$.
- $\phi = tt$ or ff
  It is easy.
- $\phi = \surd$
  From $P \models \phi; \surd$, we have $P \models \surd$ by P1. Thus, $\mathcal{T}(P)$. Hence, $P; Q \models \phi\{\tau/\surd\}; \psi$ by Seq-2, N and the assumption $Q \models \psi$.

- $\phi = \langle a \rangle$
  Since $P \models \langle a \rangle; \surd$, it follows that there exists P' such that $P \xrightarrow{a} P'$ and $\mathcal{T}(P')$. Thus, $P; Q \xrightarrow{a} Q$ by Seq-2. Furthermore, because $Q \models \psi$, we obtain that $P; Q \models \phi\{\tau/\surd\}; \psi$.
- $\phi = [a]$
  It is easy to show that $\neg\mathcal{T}(P)$ and $\forall P'.P \xrightarrow{a} P'$ implies $\mathcal{T}(P')$ because $P \models [a]; \surd$. On the other hand, by Seq-2, it can be shown that $\forall R.P; Q \xrightarrow{a} R$ only if $\exists P'.P \xrightarrow{a} P' \wedge R \sim Q$. By Theorem 2, $P; Q \models [a]; \psi$.

Induction hypothesis: For any closed PNF formulae φ and χ, and $P_1, P_1', P_2 \in \mathcal{P}^*$, if φ and χ are PNF, $P_1 \models \varphi; \surd$ and $P_2 \models \chi$, then for any PNF formula γ, if $\gamma < \varphi$ and $P_1' \models \gamma; \surd$, then $P_1'; P_2 \models \gamma\{\tau/\surd\}; \chi$. **(IH)**

Induction steps:

- $\phi = \bigwedge_{i \in I} \phi_i$
  Since $P \models (\bigwedge_{i \in I} \phi_i;)\surd$, it follows that $P \models \phi_i; \surd$ for any $i \in I$ by IC. Besides, it is clear that ϕ_i is PNF since ϕ is PNF. Whence, we have that $P; Q \models \phi_i\{\tau/\surd\}; \psi$ by (IH) for each $i \in I$. Thus, we have $P; Q \models (\bigwedge \phi_i\{\tau/\surd\}); \psi$ from IC.
- $\phi = \bigvee_{i \in I} \phi_i$
  Similar to the above case.
- $\phi = \phi_1; \phi_2$ (W.l.o.g., assume $\phi_2 \not\Leftrightarrow \tau$ by N)
  This case is committed by induction on ϕ_1 w.r.t. $<$ as follows:
  - $\phi_1 = tt$ or ff
    It is easy to show by P1.
  - $\phi_1 = \surd$
    This violates the assumption that $\surd$ does not occur in ϕ, and needs therefore not be considered.
  - $\phi_1 = \tau$
    For $P \models (\phi_1; \phi_2); \surd$, it follows that $P \models \phi_2; \surd$ by N. Since ϕ_2 is a proper sub-formula of ϕ, we get $P; Q \models \phi_2\{\tau/\surd\}; \psi$ by (IH). Therefore, $P; Q \models (\phi_1; \phi_2)\{\tau/\surd\}; \psi$ by N.
  - $\phi_1 = \langle a \rangle$
    As $P \models \langle a \rangle; \phi_2; \surd$, there exists P' such that $P \xrightarrow{a} P'$ and $P' \models \phi_2; \surd$. Because $\phi_2 < \phi$, it follows that $P'; Q \models \phi_2\{\tau/\surd\}; \psi$ by (IH). Thus, by Seq-1, $P; Q \models (\langle a \rangle; \phi_2)\{\tau/\surd\}; \psi$.
  - $\phi_1 = [a]$
    It is easy to see that $\neg\mathcal{T}(P)$ and for all $P', P \xrightarrow{a} P'$ implies $P' \models \phi_2; \surd$, as $P \models [a]; \phi_2; \surd$. Moreover, since $\phi_2 < \phi$, we get $P'; Q \models \phi_2\{\tau/\surd\}; \psi$ by (IH). On the other hand, by Seq-1, for any R, $P; Q \xrightarrow{a} R$ only if there exists P' such that $P \xrightarrow{a} P'$ and $R \sim P'; Q$. Hence, $P; Q \models ([a]; \phi_2)\{\tau/\surd\}; \psi$ by Theorem 2.
  - $\phi_1 = \bigwedge_{i \in I} \phi_i'$
    Since $P \models (\bigwedge_{i \in I} \phi_i'); \phi_2; \surd$, it follows $P \models \bigwedge_{i \in I}((\phi_i'; \phi_2); \surd)$ from IC. Therefore $P \models (\phi_i'; \phi_2); \surd$ for each $i \in I$. Obviously, $\phi_i'; \phi_2 < \bigwedge_{i \in I} \phi_i'; \phi_2$. It follows that $P; Q \models (\phi_i'; \phi_2)\{\tau/\surd\}; \psi$ for any $i \in I$ from (IH). Thus, $P; Q \models ((\bigwedge_{i \in I} \phi_i'); \phi_2)\{\tau/\surd\}; \psi$ by IC.
  - $\phi_1 = \bigvee_{i \in I} \phi'$
    Similar to the above case.
  - $\phi_1 = \phi'; \phi''$
    By C, $(\phi'; \phi''); \phi_2 \Leftrightarrow \phi'; (\phi''; \phi_2)$. Thus, it follows that $P \models \phi'; (\phi''; \phi_2); \surd$ because $P \models (\phi'; \phi''); \phi_2; \surd$. On the other hand, it is easy to see that $\phi'; (\phi''; \phi_2) < (\phi'; \phi''); \phi_2)$ by the definition of $<$. So, we obtain that $P; Q \models (\phi'; (\phi''; \phi_2))\{\tau/\surd\}; \psi$ by (IH) and therefore $P; Q \models ((\phi'; \phi''); \phi_2)\{\tau/\surd\}; \psi$ by applying C.
  - $\phi_1 = \nu^\alpha X.\phi'$
    We will use the following claim to justify this case, i.e.,

Claim 1. *If $P' \models (\nu^\alpha X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2; \surd$ and $Q' \models \chi$, then $P'; Q' \models [(\nu^\alpha X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2]\{\tau/\surd\}; \chi$, where φ is PNF and of the form $Y, p, \tau, @, \phi_1' \vee \phi_2', \phi_1' \wedge \phi_2', \phi_1'; \phi_2', \sigma Y.\phi_1'$ or $\sigma^{\lambda'} Y.\phi_1'$ where λ' is a limit ordinal.*

Proof for Claim 1. By induction on $\alpha + (\beta_1 + 1) + \cdots + (\beta_n + 1)$.

1. $\alpha = 0$
   It is trivial.
2. $\alpha = \lambda$, where λ is a limit ordinal
   Thus, $P' \models (\nu^\lambda X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2; \surd$ iff $\forall \beta < \lambda.P' \models (\nu^\beta X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2; \surd$. By the local induction hypothesis, it immediately follows
   $$P'; Q' \models ((\nu^\lambda X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2)\{\tau/\surd\}; \chi.$$
3. $\alpha = \beta + 1$

By the semantics of AFLC,

$$\begin{aligned}&\nu^{\beta+1}\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi\\ \Leftrightarrow\ &\varphi\{\varphi'/X\}\{\varphi_1'/X_1\}\cdots\{\varphi_n'/X_n\},\text{ where,}\end{aligned}$$

$$\begin{aligned}\varphi' &= \nu^{\beta}X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi,\\ \varphi_1' &= \sigma_1^{\beta_1}X_1.\sigma_2^{\beta_2+1}X_2.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi\{\varphi'/X\},\\ &\vdots\\ \varphi_i' &= \sigma_i^{\beta_i}X_i.\sigma_{i+1}^{\beta_{i+1}+1}X_{i+1}.\cdots.\sigma_n^{\beta_n+1}.\varphi\{\varphi'/X\}\{\varphi_1'/X_1\}\cdots\{\varphi_{i-1}'/X_{i-1}\},\\ &\vdots\\ \varphi_n' &= \sigma_n^{\beta_n}X_n.\varphi\{\varphi'/X\}\{\varphi_1'/X_1\}\cdots\{\varphi_{n-1}'/X_{n-1}\}.\end{aligned}$$

For brevity, we use $\{\overrightarrow{\varphi^*}\}$ to denote the vector $\{\varphi'/X\}\{\varphi_1'/X_1\}\cdots\{\varphi_n'/X_n\}$.

Now we show this subcase by the case analysis on the structure of φ.

(a) $\varphi = p$
It is straightforward.

(b) $\varphi = \tau$
Using (IH), it is easy to show.

(c) $\varphi = X$ or X_i where $i = 1, \ldots, n$
It is trivial by the local induction hypothesis.

(d) $\varphi = @$
Similar to the subcases when $\phi_1 = @$.

(e) $\varphi = \phi_1' \wedge \phi_2'$
Thus, $P' \models (\phi_i'\{\overrightarrow{\varphi^*}\}; \phi_2); \surd$ for $i = 1, 2$ as $P' \models (\varphi'; \phi_2); \surd$. Applying (IH), we get $P'; Q' \models ((\phi_i'\{\overrightarrow{\varphi^*}\}); \phi_2))\{\tau/\surd\}; \chi$ for $i = 1, 2$. Hence,

$$P'; Q' \models ((\nu^{\beta+1}.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi); \phi_2)\{\tau/\surd\}; \chi.$$

(f) $\varphi = \phi_1' \vee \phi_2'$
Similar to the above subcase.

(g) $\varphi = \phi_1'; \phi_2'$
Obviously, $\phi_1'\{\overrightarrow{\varphi^*}\}; (\phi_2'\{\overrightarrow{\varphi^*}\}; \phi_2) < (\phi_1'\{\overrightarrow{\varphi^*}\}; \phi_2'\{\overrightarrow{\varphi^*}\}); \phi_2)$ according to the definition of $<$. Therefore,

$$P'; Q' \models (\phi_1'\{\overrightarrow{\varphi^*}\}; (\phi_2'\{\overrightarrow{\varphi^*}\}; \phi_2))\{\tau/\surd\}; \chi$$

by applying (IH). By C, it follows that

$$P'; Q' \models (\nu^{\alpha}X.\sigma_1^{\beta_1+1}X_1.\cdots.\sigma_n^{\beta_n+1}X_n.\varphi; \phi_2)\{\tau/\surd\}; \chi.$$

(h) $\phi' = \sigma Y.\phi''$ or $\sigma^{\lambda'}Y.\phi'$ where λ' is a limit ordinal
Because of Proposition 1, these subcases can be readily reduced to the case (2) in the proof for Claim 1.

End of the Proof for Claim 1

– $\phi_1 = \mu^{\alpha}X.\phi'$
Similar to the above subcase.

– $\phi_1 = \sigma X.\phi'$
Applying Proposition 1, this case can be reduced to the previous two subcases.

• $\phi = \sigma^{\alpha}X.\phi_1$ or $\sigma X.\phi_1$
Similar to the subcase when $\phi_1 = \nu^{\alpha}X.\phi'$ in the proof for the case $\phi = \phi_1; \phi_2$. □

Remark 1. In Theorem 3, if P is not a normed process, and $P \models \phi; \surd$, where ϕ is PNF, then we can prove that $\phi\{\tau/\surd\}; \psi \Leftrightarrow \phi\{\tau/\surd\}$. This is in accordance with that $P; Q \sim P$ at the model level. For example, $P\widehat{=}rec\,x.\,a; x$, $Q\widehat{=}c; d$, $\phi\widehat{=}\nu X.\langle a\rangle; X$ and $\psi\widehat{=}\langle c\rangle; \langle d\rangle$. Obviously, $P \models \phi; \surd$ and $Q \models \psi$, thus $P; Q \models \phi\{\tau/\surd\}; \psi$. On the other hand, it is easy to see that $P; Q \sim P$ and $\phi\{\tau/\surd\}; \psi \Leftrightarrow \phi\{\tau/\surd\}$.

Remark 2. The above remark implies that the converse of Theorem 3 is not valid in general, that is, it is possible that $P; Q \models \phi\{\tau/\surd\}; \psi$ and $P \models \phi; \surd$, where ϕ is PNF, but $Q \not\models \psi$. For example, in the above example, let $\psi'\widehat{=}\langle d\rangle; \langle c\rangle$. Since

$P; Q \sim P$ and $\phi\{\tau/\surd\}; \psi' \Leftrightarrow \phi\{\tau/\surd\}$, it is easy to see that $P; Q \models \phi\{\tau/\surd\}; \psi'$ from $P \models \phi; \surd$. However, obviously $Q \not\models \psi'$.

5. Towards hierarchical specifications

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. In the algebraic settings, action refinement as introduced in Section 2 supports the hierarchical design.

However, how to introduce such an idea of refinement into specification logics of process algebras is still an uncultivated field although some first attempts have been done, e.g., [27,38,39]. To this end, a refinement mapping is defined by substituting the properties regarding the refinement of an abstract action a for the modalities $\langle a\rangle$ and $[a]$ in a high-level specification, producing 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 intuition, a well-defined refinement mapping should preserve the types of properties to be refined, i.e., an existential property should be refined to an existential property and similarly for the other properties. Otherwise, the mapping is meaningless in the sense that it is impossible to establish a correspondence between action refinement for models and action refinement for specifications, which plays an important role in decreasing the complexity of the verification of large systems. For example, $P \widehat{=} a; b + a; c \models \langle a\rangle; \langle b\rangle, a_1; a_2 \models [a_1]; \langle a_2\rangle$, but $P[a \rightsquigarrow a_1; a_2] \not\models ([a_1]; \langle a_2\rangle); \langle b\rangle$, since in the high-level specification, $\langle a\rangle; \langle b\rangle$ is an existential property, however its refinement becomes a universal property.

To ensure that the mapping preserves the types of properties to be refined, we partition the property ψ concerning a refinement into two parts: existential property ψ_1 and universal property ψ_2, i.e., $\psi \in \mathcal{PF}$. $[a]$ will be replaced by ψ_2, and $\langle a\rangle$ will be replaced by ψ_1. This is justified by the result shown in [13] that any property can be expressed as the intersection of a liveness property and a safety property in branching temporal logics and the fact that a liveness property can be represented by an existential formula and a safety property by a universal formula, respectively (see [49]). So, $\mathcal{PF}$ is powerful enough to define the properties of reactive systems.

Therefore, we define the refinement mapping as follows.

Definition 10. Suppose ϕ is a high-level specification, a is an abstract action to be refined, and $\psi_1 \in \mathcal{EF}$ and $\psi_2 \in \mathcal{UF}$ and $\psi_1 \wedge \psi_2 \in \mathcal{PF}$ is the description of the refinement of a. We define the refinement mapping, denoted by $\Omega(\phi, \psi_1, \psi_2, a)$, as follows:

$$\Omega(\phi, \psi_1, \psi_2, a) \widehat{=} \phi\{\psi_1\{\tau/\surd\}/\langle a\rangle, \psi_2\{\tau/\surd\}/[a]\}.$$

Sometimes, for brevity, we directly write $\Omega(\phi, \psi, a)$ for $\Omega(\phi, \psi_1, \psi_2, a)$ if $\psi = \psi_1 \wedge \psi_2 \in \mathcal{PF}$, where $\psi_1 \in \mathcal{EF}$ and $\psi_2 \in \mathcal{UF}$, is clear from the context.

According to the above definition, it is easy to get the following results.

Lemma 8. *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 9

1. $\Omega(\phi, \psi, a) \Leftrightarrow \phi$, *if* $\phi = p, \tau$ *or* $\textcircled{b}$, *where* $a \neq b$;
2. $\Omega(\langle a\rangle, \psi, a) \Leftrightarrow \psi_1\{\tau/\surd\}$;
3. $\Omega([a], \psi, a) \Leftrightarrow \psi_2\{\tau/\surd\}$;
4. $\Omega(\bigwedge_{i\in I} \phi_i, \psi, a) \Leftrightarrow \bigwedge_{i\in I} \Omega(\phi_i, \psi, a)$
5. $\Omega(\bigvee_{i\in I} \phi_i, \psi, a) \Leftrightarrow \bigvee_{i\in I} \Omega(\phi_i, \psi, a)$
6. $\Omega(\phi_1; \phi_2, \psi, a) \Leftrightarrow \Omega(\phi_1, \psi, a); \Omega(\phi_2, \psi, a)$
7. $\Omega(\sigma^\alpha X.\phi, \psi, a) \Leftrightarrow \sigma^\alpha X.\Omega(\phi, \psi, a)$, *where X is not free in* ψ
8. $\Omega(\sigma X.\phi, \psi, a) \Leftrightarrow \sigma X.\Omega(\phi, \psi, a)$, *where X is not free in* ψ.

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

Here, we study the example of a salesman that is firstly presented in [19] to demonstrate how to employ our approach to hierarchically specify a complex system.

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.

So, the top-most specification of the system may be represented as follows:

$$\phi \;\widehat{=}\; \nu X. \left(\begin{array}{c} \langle \text{leave\_Paris}\rangle;\, [\text{fr\_thr\_Channel}];\, \langle \text{arrive\_in\_London}\rangle;\, \langle \text{work}\rangle; \\ \langle \text{leave\_London}\rangle;\, [\text{gb\_thr\_Channel}];\, \langle \text{arrive\_in\_Paris}\rangle;\, X \end{array} \right)$$

where the actions “work" and “x_thr_Channel" will be refined subsequently.

The job of the salesman in London is either to contact some of his customers by phone, or to meet some of them in his office to discuss something, or decide to finish that day's work. After finishing one task, the salesman may repeat the above procedure. Therefore, “work" may be refined by a process that meets the following property, namely

$$\psi_1 \,\widehat{=}\, \mu X.((\langle \text{contact\_Customers}\rangle \vee \langle \text{meet\_Customers}\rangle);\, X \vee \langle \text{finish\_Work}\rangle).$$

Meanwhile, we can describe “x_thr_Channel" in more detail. There are two platforms lying on the two sides of the Channel, respectively that take charge of the hovercraft. At the beginning, one of them loads the salesman's car, then arranges the hovercraft to depart. Then the hovercraft crosses through the Channel. After the hovercraft arrives at the opposite side, the other platform unloads the car. Hence, “x_thr_Channel" may be enriched as follows:

$$\psi_x \,\widehat{=}\, [\text{x\_load}];\, [\text{x\_departure}];\, \langle \text{cross\_Channel}\rangle;\, \langle \overline{\text{x}}\text{\_arrival}\rangle;\, \langle \overline{\text{x}}\text{\_unload}\rangle.$$

Furthermore, we can refine “x_departure" by a process with the property

$$\psi_2 \,\widehat{=}\, [\text{finish\_loading}];\, \langle \text{engine\_on}\rangle;\, \langle \text{bye\_bye}\rangle$$

where finish_loading signals the end of loading, and cross_Channel by a process with the property

$$\begin{aligned} \psi_3 \;\widehat{=}\;& \langle \text{sit\_down}\rangle; \\ & (\mu \text{X}.((\langle \text{newspaper}\rangle \vee \langle \text{tea}\rangle \vee \langle \text{coffee}\rangle);\, \text{X} \vee \langle \text{keep\_idle}\rangle));\, \langle \text{stand\_up}\rangle. \end{aligned}$$

So, the specification for the final system can be represented by

$$\begin{aligned} &\Omega(\Omega(\phi,\, \Omega(\Omega(\psi_x,\, \psi_2,\, \text{x\_departure}),\, \psi_3,\, \text{cross\_Channel}), \\ &\qquad\qquad\qquad\qquad \text{x\_thr\_Channel}),\, \psi_1,\, \text{work}), \end{aligned}$$

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

Note that in the above example, if the parameter ψ_1 or ψ_2 of Ω is ignored, then it is implicitly set to tt.

6. Relating hierarchical specifications of a complex system to its hierarchical implementations

In this section we will establish a correspondence presented by the Refinement Theorem below between hierarchical specifications of a complex system and its hierarchical implementations. It states that if $Q \models \psi; \surd$, $P \models \phi$ and some syntactical conditions hold, then $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi, a)$. 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.

In order to ensure that the Refinement Theorem is valid, the following syntactical conditions are necessary:

Above all, it is required that $(Act(P) \cup Act(\phi)) \cap (Act(Q) \cup Act(\psi)) = \emptyset$, because of the following considerations:

(i) No deadlock will be introduced or destroyed because of action refinement.
(ii) It is impossible that $P[a \rightsquigarrow Q]$ will fail to satisfy $\Omega(\phi, \psi, a)$ because ϕ involves Q. For instance, let $P \widehat{=} a;\, b$, $\phi \widehat{=} ([a];\, \langle b\rangle) \wedge ([c];\, \langle d\rangle)$, $Q \widehat{=} c;\, e$ and $\psi \widehat{=} [c];\, \langle e\rangle$. It is obvious that $P \models \phi$ and $Q \models \psi; \surd$, but $P[a \rightsquigarrow Q] \not\models \Omega(\phi, \psi, a)$.

(iii) Symmetrically, it is impossible that $P[a \rightsquigarrow Q]$ will fail to satisfy $\Omega(\phi, \psi, a)$ because ψ involves P. For example, let $P \widehat{=} a; b + b; a$, $\phi \widehat{=} [a]; \langle b \rangle$, $Q \widehat{=} c; e$ and $\psi \widehat{=} [c]; \langle e \rangle \wedge [b]; \langle d \rangle$. It is obvious that $P \models \phi$ and $Q \models \psi; \surd$, but $P[a \rightsquigarrow Q] \not\models \Omega(\phi, \psi, a)$.

It is clear that the disjoint action condition can guarantee the above three requirements.

Furthermore, it is possible that ψ only describes some incomplete executions of Q, so the refined specification may not be satisfied by the refined system. For example, it is obvious that $a; b + a; c \models \langle a \rangle; \langle b \rangle$ and $a_1; a_2 \models \langle a_1 \rangle$, but $(a; b + a; c)[a \rightsquigarrow a_1; a_2] \not\models \langle a_1 \rangle; \langle b \rangle$. In order to solve such a problem, we require that ψ describes complete executions of Q, i.e., $Q \models \psi; \surd$.

Finally, as argued in Theorem 3, it is required that ϕ and ψ are PNF.

Now, we can exactly state the Refinement Theorem as follows:

Theorem 5 (Refinement Theorem). *Let $\phi \in \mathcal{L}_{\text{AFLC}}$, $\psi_1 \in \mathcal{EF}$ and $\psi_2 \in \mathcal{UF}$, where ϕ, ψ_1 and ψ_2 are PNF. If $(Act(P) \cup Act(\phi)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$, $Q \models (\psi_1 \wedge \psi_2); \surd$, and $P \models \phi$ then $P[a \rightsquigarrow Q] \models \Omega(\phi, \psi_1, \psi_2, a)$.*

In order to demonstrate how to apply the Refinement Theorem to verify a complex system hierarchically, we will continue Example 2.

Example 3. According to the specification explained in Example 2, at the top level, we can implement the system as:

$$\text{Sys} \widehat{=} \text{fr\_Channel} \parallel_{\{\text{fr\_thr\_Channel}\}} \text{Salesman} \parallel_{\{\text{gb\_thr\_Channel}\}} \text{gb\_Channel}.$$

where x_Channel $\widehat{=}$ $rec\, y$. x_thr_Channel; y, and

$$\begin{aligned} \text{Salesman} \widehat{=} rec\, x.\ & \text{leave\_Paris; fr\_thr\_Channel; arrive\_in\_London;} \\ & \text{work; leave\_London; gb\_thr\_Channel; arrive\_in\_Paris; } x. \end{aligned}$$

It is easy to check that Sys $\models \phi$.

Then, we can refine “work" by Subsys_1 which is defined by

$$\text{Subsys}_1 \widehat{=} rec\, x. ((\text{contact\_Customers} + \text{meet\_Customers}); x + \text{finish\_Work}).$$

Using model-checking or other methods, it can be verified that $\text{Subsys}_1 \models \psi_1; \surd$.

Then, “x_thr_Channel" may be implemented by $\text{Subsys}_x \widehat{=} \text{x\_load} ||_{\{\text{x\_load}\}} \text{Channel}$,

$$\begin{aligned} \text{where Channel} \quad \widehat{=} \text{fr\_Platform} \quad & \parallel_{\{\text{fr\_arrival,fr\_departure}\}} \text{Hovercraft} \\ & \parallel_{\{\text{gb\_arrival,gb\_departure}\}} \text{gb\_Platform}, \end{aligned}$$

$$\begin{aligned} \text{where} \quad \text{Hovercraft} \quad & \widehat{=} \quad \text{fr\_departure;cross\_Channel; gb\_arrival +} \\ & \quad\quad \text{gb\_departure;cross\_Channel; fr\_arrival,} \\ \text{x\_Platform} \quad & \widehat{=} \quad \text{x\_load;x\_departure + x\_arrival;x\_unload.} \end{aligned}$$

It is not hard to show that $\text{Subsys}_\text{x} \models \psi_x; \surd$.

Furthermore, we may refine “x_departure" by Subsys_2 and “cross_Channel" by Subsys_3, where,

$$\begin{aligned} \text{Subsys}_2 & \widehat{=} \text{finish\_loading; engine\_on; bye\_bye,} \\ \text{Subsys}_3 & \widehat{=} \text{sit\_down;} \\ & \quad rec\, x. ([(\text{coffee} + \text{tea}) ||_{\{\}} \text{newspaper}]; x + \text{keep\_idle}); \text{stand\_up}. \end{aligned}$$

Certainly, it can be proved that $\text{Subsys}_2 \models \psi_2; \surd$ and $\text{Subsys}_3 \models \psi_3; \surd$.

Thus, the final system may be obtained as

$$\mathrm{Sys}\left[\begin{array}{l}\mathrm{work} \rightsquigarrow \mathrm{Subsys}_1,\\ \mathrm{x\_thr\_Channel} \rightsquigarrow \mathrm{Subsys}_{\mathrm{x}}\left[\begin{array}{l}\mathrm{x\_departure} \rightsquigarrow \mathrm{Subsys}_2,\\ \mathrm{cross\_Channel} \rightsquigarrow \mathrm{Subsys}_3\end{array}\right]\end{array}\right],$$

where $\mathrm{x} \in \{\mathrm{fr}, \mathrm{gb}\}$.

According to the Refinement Theorem, the final system satisfies the final specification.

In the following, we will give the proof for the Refinement Theorem.

Proof. The proof proceeds by induction on the structure of ϕ w.r.t. $<$.

Basic cases:

- $\phi = p, \tau, \textcircled{b}$, where $b \neq a$
  It is straightforward by Definition 10.
- $\phi = [a]$
  Thus, from Lemma 9, $\Omega([a], \psi_1, \psi_2, a) \Leftrightarrow \psi_2\{\tau/\surd\}$. On the other hand, similar to the proof given by Aceto and Hennessy in [5], we can prove

  $$P \sim \sum_{i=1}^{m} a;\, P_i + \sum_{i=1}^{n} *a;\, P_i' + \sum_{i=1}^{\ell} b_i;\, Q_i + \sum_{i=1}^{h} *c_i;\, Q_i'$$

  where $b_i \neq a$ for $1 \leq i \leq \ell$ and $c_i \neq a$ for $1 \leq i \leq h$. So, applying Theorem 1, it follows that

  $$P[a \rightsquigarrow Q] \sim \left(\sum_{i=1}^{m} a;\, P_i + \sum_{i=1}^{n} *a;\, P_i' + \sum_{i=1}^{\ell} b_i;\, Q_i + \sum_{i=1}^{h} *c_i;\, Q_i'\right)[a \rightsquigarrow Q].$$

  Furthermore, after multiple applications of Theorem 1, we get

  $$\begin{aligned}P[a \rightsquigarrow Q] \sim & \sum_{i=1}^{m} *Q;\, P_i[a \rightsquigarrow Q] + \sum_{i=1}^{n} *Q;\, P_i'[a \rightsquigarrow Q] + \\ & \sum_{i=1}^{\ell} b_i;\, Q_i[a \rightsquigarrow Q] + \sum_{i=1}^{h} *c_i;\, Q_i'[a \rightsquigarrow Q]. \end{aligned} \tag{1}$$

  As $(Act(\phi) \cup Act(P)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$, it follows that $b_i \notin Act(\psi_2) \cup Act(Q)$ for $1 \leq i \leq \ell$ and $c_i \notin Act(\psi_2) \cup Act(Q)$ for $1 \leq i \leq h$. Moreover, by the well-formedness condition (iii) listed in Section 2.1, it is obvious that $\neg\mathcal{T}(Q)$. Since $Q \models \psi_2; \surd$ and $\psi_2 \in \mathcal{UF}$, it follows that $\psi_2\{\tau/\surd\} \in \mathcal{UF}$, $\psi_2\{\tau/\surd\} \not\Leftrightarrow f\!f$ and $\surd \notin BSub(\psi_2\{\tau/\surd\})$. By Lemma 6, without loss of generality,

  $$\psi_2 \Leftrightarrow \bigvee_{i=1}^{k}(\tau \wedge \bigwedge_{j=1}^{n_i} \psi_{i,j}) \vee \bigvee_{i=k+1}^{m'} (\bigwedge_{j=1}^{n_i} \psi_{i,j}) \tag{2}$$

  where $USF(\psi_{i,j})$ and $\tau \notin BSub(\psi_{i,j})$ for all $1 \leq i \leq m'$ and $1 \leq j \leq n_i$. So, there exists $k < i' \leq m'$ such that $Q \models (\bigwedge_{j=1}^{n_{i'}} \psi_{i',j}); \surd$. Thus, by Corollary 1, it can be shown that for any $1 \leq i \leq \ell$, $b_i;\, Q_i[a \rightsquigarrow Q] \models (\bigwedge_{j=1}^{n_{i'}} \psi_{i',j})\{\tau/\surd\}$ and for each $1 \leq i \leq h$, $*c_i;\, Q_i'[a \rightsquigarrow Q] \models (\bigwedge_{j=1}^{n_{i'}} \psi_{i',j})\{\tau/\surd\}$. It follows that for any $1 \leq i \leq \ell$, $b_i;\, Q_i[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}$ and for each $1 \leq i \leq h$, $c_i;\, Q_i'[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}$ from (2).

  As $P_i[a \rightsquigarrow Q] \models \tau$ by Definition 7 and $*Q \models \psi; \surd$, from Lemma 1 and Theorem 2, applying Theorem 3, we get $*Q;\, P_i[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}; \tau$ for any $1 \leq i \leq m$. According to N, $*Q;\, P_i[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}$ for any $1 \leq i \leq m$. Similarly, it can be shown that $*Q;\, P_i'[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}$ for any $1 \leq i \leq n$. Therefore, we get $P[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}$ from Proposition 5 and Theorem 2.
- $\phi = \langle a \rangle$
  Similar to the above case.

Induction hypothesis: Let $\varphi \in \mathcal{L}_{\text{AFLC}}$, $\chi_1 \in \mathcal{EF}$ and $\chi_2 \in \mathcal{UF}$ be closed PNF formulae, and $P_1, P_2 \in \mathcal{P}^*$. If $(Act(P_1) \cup Act(\varphi)) \cap (Act(\chi_1 \wedge \chi_2) \cup Act(P_2)) = \emptyset$, $P_1 \models \varphi$ and $P_2 \models (\chi_1 \wedge \chi_2); \surd$, then if $\varphi' < \varphi$, $(Act(P_1) \cup Act(\varphi')) \cap (Act(\chi_1 \wedge \chi_2) \cup Act(P_2)) = \emptyset$, and $P_1 \models \varphi'$, then $P_1[a \rightsquigarrow P_2] \models \Omega(\varphi', \chi_1, \chi_2, a)$. **(IH)**

Induction steps:

- $\phi = \bigwedge_{i \in I} \phi_i$
  Since $P \models \bigwedge_{i\in I} \phi_i$, we get $P \models \phi_i$ for each $i \in I$. It is easy to see that for each $i \in I$, $(Act(P) \cup Act(\phi_i)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$, and therefore, from (IH), we get $P[a \rightsquigarrow Q] \models \Omega(\phi_i, \psi_1, \psi_2, a)$. This entails $P[a \rightsquigarrow Q] \models \Omega(\bigwedge_{i\in I} \phi_i, \psi_1, \psi_2, a)$ by Lemma 9.
- Case $\phi = \bigvee_{i\in I} \phi_i$
  Similar to the above case.
- $\phi = \varphi_1; \varphi_2$
  This case can be proved by induction on φ_1 w.r.t. $<$.
  - $\varphi_1 = tt, ff, \surd$
    It is trivial.
  - $\varphi_1 = \tau$
    Since $P \models \varphi_1; \varphi_2$ and $\varphi_1; \varphi_2 \Leftrightarrow \varphi_2$ by N, it concludes that $P \models \varphi_2$. Furthermore, it is obvious that $\varphi_2 < \phi$ and $(Act(P) \cup Act(\varphi_2)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$. It follows $P[a \rightsquigarrow Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a)$ by (IH).
  - $\varphi_1 = \langle b\rangle$ where $a \neq b$
    Since $P \models \langle b\rangle; \varphi_2$, there exists P' such that $P \xrightarrow{b} P'$ and $P' \models \varphi_2$. It is clear that $\varphi_2 < \phi$ and $(Act(P') \cup Act(\varphi_2)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$, so it follows that $P'[a \rightsquigarrow Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a)$ by (IH). This entails $P'[a \rightsquigarrow Q] \models \Omega(\langle b\rangle; \varphi_2, \psi_1, \psi_2, a)$ from Ref-1 and Lemma 9.
  - $\varphi_1 = [b]$ where $a \neq b$
    Since $P \models [b]; \varphi_2$, $\neg\mathcal{T}(P)$ and for any P', $P \xrightarrow{b} P'$ implies $P' \models \varphi_2$. Moreover, it is obvious that $\varphi_2 < \phi$ and $(Act(P') \cup Act(\varphi_2)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$. Thus, we get $P'[a \rightsquigarrow Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a)$ according to (IH). On the other hand, by Ref-1, for any R, $P[a \rightsquigarrow Q] \xrightarrow{b} R$ only if there exists P' such that $P \xrightarrow{b} P'$ and $R \sim P'[a \rightsquigarrow Q]$, because of the assumption that $(Act(\phi) \cup Act(P)) \cap (Act(\psi) \cup Act(Q)) = \emptyset$. Hence, $P[a \rightsquigarrow Q] \models \Omega(\varphi_1; \varphi_2, \psi_1, \psi_2, a)$ by Lemma 9 and Theorem 2.
  - $\varphi_1 = [a]$
    Thus, $\Omega([a]; \varphi_2, \psi_1, \psi_2, a) \Leftrightarrow \psi_2\{\tau/\surd\}; \Omega(\varphi_2, \psi_1, \psi_2, a)$ from Lemma 9. From (1) and $(Act(\phi) \cup Act(P)) \cap (Act(\psi_1 \wedge \psi_2) \cup Act(Q)) = \emptyset$, it follows that $b_i, c_j \notin Act(\psi_1 \wedge \psi_2) \cup Act(Q)$ for $1 \le i \le \ell$ and $1 \le j \le h$. Moreover, we get $\psi_2\{\tau/\surd\} \in \mathcal{UF}$, $\psi_2\{\tau/\surd\} \not\Leftrightarrow ff$ and $\surd \notin BSub(\psi_2\{\tau/\surd\})$, as $Q \not\sim nil$, $Q \models \psi_2$ and $\psi_2 \in \mathcal{UF}$. On the other hand,

$$\psi_2\{\tau/\surd\} \Leftrightarrow \bigvee_{i=1}^{k} (\tau \wedge \bigwedge_{j=1}^{n_i} \psi_{i,j}) \vee \bigvee_{i=k+1}^{m} (\bigwedge_{j=1}^{n_i} \psi_{i,j}) \tag{3}$$

by Lemma 6, where $\psi_{i,j} \in \mathcal{UF}$, $USF(\psi_{i,j})$ and $\tau \notin BSub(\psi_{i,j})$ for $1 \le i \le m$ and $1 \le j \le n_i$. Since $\neg\mathcal{T}(Q)$ and $Q \models \psi_2; \surd$, it follows that there exists $k < h \le m$ such that $\bigwedge_{j=1}^{n_i} \psi_{h,j} \not\Leftrightarrow ff$. Thus, by Lemma 7, it can be shown that for any $1 \le k \le \ell$,

$$b_k; Q_k[a \rightsquigarrow Q] \models (\bigwedge_{j=1}^{n_i} \psi_{i,j}); \Omega(\varphi_2, \psi_1, \psi_2, a)$$

and

$$*c_k; Q_k'[a \rightsquigarrow Q] \models (\bigwedge_{j=1}^{n_i} \psi_{i,j}); \Omega(\varphi_2, \psi_1, \psi_2, a)$$

for each $1 \le k \le h$. Furthermore, applying DC, we get for any $1 \le k \le \ell$,

$$b_k; Q_k[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}; \Omega(\varphi_2, \psi_1, \psi_2, a)$$

and

$$*c_k; Q_k'[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}; \Omega(\varphi_2, \psi_1, \psi_2, a)$$

for each $1 \le k \le h$.

For $1 \le k \le m$, $P_k[a \rightsquigarrow Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a)$ by (IH). Besides, as $Q \models (\psi_1 \wedge \psi_2); \surd$, according to Theorem 3, Theorem 2 and (IH), $*Q; P'_k[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}; \Omega(\varphi_2, \psi_1, \psi_2, a)$ for any $1 \le k \le n$. Therefore, we get

$$P[a \rightsquigarrow Q] \models \psi_2\{\tau/\surd\}; \Omega(\varphi_2, \psi_1, \psi_2, a)$$

from (1), Proposition 5 and Theorem 2.

– $\varphi_1 = \langle a \rangle$
Similar to the above case.

– $\varphi_1 = \bigwedge_{i\in I} \varphi_{1i}$
Since $P \models (\bigwedge_{i\in I} \varphi_{1i}); \varphi_2$, we have $P \models \bigwedge_{i\in I}(\varphi_{1i}; \varphi_2)$ by IC. Moreover, it is clear that $\varphi_{1i}; \varphi_2 < \varphi_1; \varphi_2$ for each $i \in I$. Therefore, $P[a \rightsquigarrow Q] \models \Omega(\varphi_{1i}; \varphi_2, \psi, a)$ using (IH) for each $i \in I$. Hence,

$$P[a \rightsquigarrow Q] \models \Omega((\bigwedge_{i\in I} \varphi_{1i}); \varphi_2, \psi_1, \psi_2, a)$$

by Lemma 9.

– $\varphi_1 = \bigvee_{i\in I} \varphi_{1i}$
Similar to the above case.

– $\varphi_1 = \varphi_{11}; \varphi_{12}$
Since $(\varphi_{11}, \varphi_{12}; \varphi_2) < (\varphi_{11}; \varphi_{12}, \varphi_2)$, it follows

$$P[a \rightsquigarrow Q] \models \Omega(\varphi_{11}; (\varphi_{12}; \varphi_2), \psi_1, \psi_2, a)$$

by applying (IH). Thus, by C and Lemma 9,

$$P[a \rightsquigarrow Q] \models \Omega((\varphi_{11}; \varphi_{12}); \varphi_2, \psi_1, \psi_2, a).$$

– $\varphi_1 = \sigma^\alpha X.\varphi_{11}$ or $\sigma X.\varphi_{11}$
Similar to the subcase when $\psi_1 = \nu^\alpha X.\phi'$ in the proof for Theorem 3.

• $\phi = \sigma^\alpha X.\phi_1$ or $\sigma X.\phi_1$
Similar to the subcase when $\psi_1 = \nu^\alpha X.\phi'$ in the proof for Theorem 3. $\square$

7. Comparing with syntactic action refinement (SAR)

In [27,38,39], the authors proposed an approach, called SAR (syntactic action refinement), to construct a lower-level specification $\phi[a \rightsquigarrow Q]$ from a high-level specification ϕ and the refinement Q of an abstract action a. Here we show that SAR can be viewed as a special case of our approach, as far as the constructed specification is concerned.

Recall that in [27,38,39] the authors only considered to refine an abstract action by a finite process, which contains no recursion, cannot be a terminated process nor a deadlock process (see the definition in Section 2.1) either.

Definition 11 (SAR for FLC). Let $Q, Q_1, Q_2 \in \mathcal{F}$ and $\phi, \phi_1, \phi_2 \in FLC$, $\phi[a \rightsquigarrow Q]$ is defined as:

$$\phi[a \rightsquigarrow Q] \ \widehat{=}\ \begin{cases} \phi & \text{if } \phi = p, X, \textcircled{b}, \tau \text{ where } b \neq a \\ E(Q) & \text{if } \phi = \langle a \rangle \\ U(Q) & \text{if } \phi = [a] \\ \phi_1[a \rightsquigarrow Q] \wedge \phi_2[a \rightsquigarrow Q] & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \phi_1[a \rightsquigarrow Q] \vee \phi_2[a \rightsquigarrow Q] & \text{if } \phi = \phi_1 \vee \phi_2 \\ \phi_1[a \rightsquigarrow Q]; \phi_2[a \rightsquigarrow Q] & \text{if } \phi = \phi_1; \phi_2 \\ \sigma X.\phi_1[a \rightsquigarrow Q] & \text{if } \phi = \sigma X.\phi_1 \end{cases}$$

where $E(Q)$ and $U(Q)$ are defined as

$$E(Q) \mathrel{\widehat{=}} \begin{cases} \langle a' \rangle & \text{if } Q = a' \\ E(Q_1) \wedge E(Q_2) & \text{if } Q = Q_1 + Q_2 \\ E(Q_1); E(Q_2) & \text{if } Q = Q_1; Q_2 \\ \left.\begin{array}{l} \quad \bigwedge_{Q_1 \xrightarrow{b} Q_1' \wedge b \notin A} \langle b \rangle; E(Q_1' \parallel_A Q_2) \\ \wedge \bigwedge_{Q_2 \xrightarrow{b} Q_2' \wedge b \notin A} \langle b \rangle; E(Q_1 \parallel_A Q_2') \\ \wedge \bigwedge_{Q_1 \xrightarrow{b} Q_1' \wedge Q_2 \xrightarrow{b} Q_2' \wedge b \in A} \langle b \rangle; E(Q_1' \parallel_A Q_2') \end{array}\right\} & \text{if } Q = Q_1 \parallel_A Q_2 \\ E(Q_1)[a' \rightsquigarrow Q_2] & \text{if } Q = Q_1[a' \rightsquigarrow Q_2], \end{cases}$$

$$U(Q) \mathrel{\widehat{=}} \begin{cases} [a'] & \text{if } Q = a' \\ U(Q_1) \wedge U(Q_2) & \text{if } Q = Q_1 + Q_2 \\ U(Q_1); U(Q_2) & \text{if } Q = Q_1; Q_2 \\ \left.\begin{array}{l} \quad \bigwedge_{Q_1 \xrightarrow{b} Q_1' \wedge b \notin A} [b]; U(Q_1' \parallel_A Q_2) \\ \wedge \bigwedge_{Q_2 \xrightarrow{b} Q_2' \wedge b \notin A} [b]; U(Q_1 \parallel_A Q_2') \\ \wedge \bigwedge_{Q_1 \xrightarrow{b} Q_1' \wedge Q_2 \xrightarrow{b} Q_2' \wedge b \in A} [b]; U(Q_1' \parallel_A Q_2') \end{array}\right\} & \text{if } Q = Q_1 \parallel_A Q_2 \\ U(Q_1)[a' \rightsquigarrow Q_2] & \text{if } Q = Q_1[a' \rightsquigarrow Q_2]. \end{cases}$$

From Definition 11, the following results are easy to prove.

Lemma 10. *For any* $Q \in \mathcal{F}$, $E(Q) \wedge U(Q) \in \mathcal{PF}$, $Q \models E(Q) \wedge U(Q)$ *and* $Q \models (E(Q) \wedge U(Q)); \surd$.

Theorem 6. *For any* $Q \in \mathcal{F}$, $\Omega(\phi, E(Q) \wedge U(Q), a) \Leftrightarrow \phi[a \rightsquigarrow Q]$.

In order to help readers to understand SAR, $E(Q)$ and $U(Q)$, we give the following example.

Example 4. Let $\phi \mathrel{\widehat{=}} \mu X.\langle a \rangle; [a]; X$, and $Q \mathrel{\widehat{=}} a'; b' + a'; c'$. So, by Definition 11, $E(Q) = \langle a' \rangle; \langle b' \rangle \wedge \langle a' \rangle; \langle c' \rangle$, $U(Q) = [a']; [b'] \wedge [a']; [c']$. It is clear that $Q \models E(Q) \wedge U(Q)$ and $Q \models (E(Q) \wedge U(Q)); \surd$. Furthermore, by Definition 11, we have

$$\begin{aligned} \phi[a \rightsquigarrow Q] &\mathrel{\widehat{=}} \mu X.(\langle a \rangle; [a]; X)[a \rightsquigarrow a'; b' + a'; c'] \\ &\Leftrightarrow \mu X.((\langle a' \rangle; \langle b' \rangle \wedge \langle a' \rangle; \langle c' \rangle); ([a']; [b'] \wedge [a']; [c']); X) \\ &\Leftrightarrow \mu X.(E(Q); U(Q); X) \\ &\Leftrightarrow \Omega(\phi, E(Q) \wedge U(Q), a). \end{aligned}$$

8. Concluding remarks

In this paper, we proposed an approach on combining hierarchical specification of a complex system and its hierarchical implementation in order to simplify the verification of large systems. To this end, we defined a refinement mapping from a high-level specification and the properties of the refinement of an abstract action to a lower-level specification, which preserves the type of properties to be refined. Furthermore, a correspondence between hierarchical specifications and hierarchical implementations that supports "a priori" verification in system design was established. All results were illustrated by the example of a salesman.

In addition, in our framework, horizontally composing specifications can also be dealt with, for example, supposing $P \models \phi; \surd$, where ϕ is PNF, and $Q \models \psi$, we can get a composite specification like $\phi\{\tau/\surd\}; \psi$ for $P; Q$. The detailed discussion related to this topic can be found in [57].

Similar results are shown in [27,38,39], but in their approaches, a refined specification is obtained from the original specification and the refining process Q. Therefore, interesting expected properties of the refined system cannot be derived using their approaches. Moreover, the refinement of an abstract action is restricted to be a finite process, whereas, in our approach an abstract action can be refined by any process which is not bisimilar to a terminated process. From a constructing specification point of view, we proved that their approaches can be seen as a special case of our method. In [3], Abadi and Plotkin argued that composing, refining specifications of reactive systems can be seen as some sound rules of a logic.

As for the work on horizontal composition, we can mention the following: In [23,35] the non-deterministic operator "+" was directly introduced into the modal μ-calculus like logics so that the resulting logics have compositionality; The compositionality of linear temporal logic [45] was discussed in [11,12] by introducing the chop into the logic, moreover, some further logic properties of the extension were investigated in [47]; While a compositional proof system for checking satisfaction relation for given process P and μ-calculus formula was established in [6]; The compositionality of a fixpoint logic in assume-guarantee style was investigated in [51]; A connection between the logic connectives of FLC and the operators of BPA (Basic Process Algebra) was established in [57] so that FLC can be used to specify complex systems in a compositional manner like process algebra.

Also, there is work concerning the converse of the problem considered in this paper, i.e., decomposition. For example, Larsen and Liu in [33,37] systematically studied the decomposition problem of regular properties, i.e., given a combined system and its specification, reducing the specification of the combined system to the specifications of the system's components such that the system satisfies the specification if and only if each of the components meets the corresponding derived specification. While they also investigated the typical decomposition problem within equational specification formalisms in [34], that is solving equation systems of the following form:

$$C_1(X) \sim P_1, \ldots, C_n(X) \sim P_n$$

where C_is belong to a class of contexts which can be described as action transducers, P_i are arbitrary process, $\sim$ is the bisimulation equivalence, and X is the unknown process to be found. In [17,18], Clarke and Emerson show how to synthesize processes (skeletons) $P_1, P_2, \ldots, P_n$ such that their parallel composition $P_1 \parallel \cdots \parallel P_n$ satisfies a given formula of CTL.

In the above, all the discussed work uses the action-based approach. In the literature, there has been much work on refinement of reactive systems using the state-based approach, e.g., [1,10,24,25,29,30,55]. In this approach, refinement is usually studied in the framework of Back's action systems [8,9], or in Lamport's Temporal Logic of Actions (TLA) [30]. To show that a program P_h (at a higher level of abstraction) is refined by another program P_l (at a lower level of abstraction), denoted by $P_h \sqsubseteq P_l$, it is in general by finding a refinement mapping from the state space of P_l to that of P_h, so that the execution of P_h "simulates" that of P_l. The correctness of a refinement is justified by showing that every behavior of the lower-level system is also a behavior of the higher-level system. For example, Abadi and Lamport in [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.

As shown in [2], the two ways to specify reactive systems, i.e., the state-based approach and the action-based approach, have the same expressive power. This means we can always find a correspondence between the state-based approach and the action-based approach. For example, the data refinement proposed by Back et al. [8–10] and the one due to Lamport in [30] corresponds to the *trace refinement* in the action-based approach; while in [25,29,55] the authors investigated how to handle *failure/divergence refinement* in the state-based approach. To the best of our knowledge, all refinement theories studied in the state-based approach, i.e., all kinds of data refinements, correspond to certain trace-based refinements in the action-based approach. However, in the action-based approach, there are lots of refinements, normally called *bisimulation*, e.g., *weak* and *strong bisimulation* [41] and *branching bisimulation* [53]. So far, we cannot see how to handle these bisimulations in the state-based approach, which are finer than trace-based refinement.

In this paper, we use the standard interleaving setting, so we only consider the case of atomic action refinement for models because the standard bisimulation notion is not preserved by non-atomic action refinement in this setting. Although the philosophy that believes atomicity should be broken down at different abstraction levels is popular in the process algebra community, atomic action refinement considered in the interleaving setting has more applications in practice. For example, in the Parallel Java Language most refinements are implemented as atomic ones [54]. Obviously, our approach will be very useful in Java programming. The basic idea is by combining with JML [36], we can derive the property that should be satisfied by the system at lower-level (replacing each method call with the method's body) from the property (assertion) of the system at higher-level (considering each method call as an abstract action) and the properties (assertions) satisfied by the called methods. The detailed implementation of the idea is part of our future work. What is more, we believe our approach can be applied to the case of non-atomic action refinement, too, if a suitable logic which is interpreted over some truly concurrent settings such as event-structures is available. We would like to leave this problem as another future work.

Acknowledgments

The authors thank Prof. Robert van Glabeek for his comments on the early version of this paper helping us to improve the presentation so much. We also thank the anonymous referee for his comments, criticisms, in particular, pointing out some technical problems in the early version.

References

[1] M. Abadi, L. Lamport, The existence of refinement mappings, Theoretical Computer Science 82 (1991) 253–284.
[2] M. Abadi, L. Lamport, Composing specifications, ACM Transactions on Programming Languages and Systems 15 (1) (1993) 73–132.
[3] M. Abadi, G. Plotkin, A logical view of composition and refinement, Theoretical Computer Science 114 (1993) 3–30.
[4] L. Aceto, M. Hennessy, Towards action refinement in process algebra, in: Proceedings of LICS'89, 1989, pp. 138–145.
[5] L. Aceto, M. Hennessy, Termination, deadlock, and divergence, Journal of ACM 39 (1) (1992) 147–187.
[6] H.R. Andersen, C. Stirling, G. Winskel, A compositional proof system for the modal mu-calculus, in: Proceedings of LICS'94, 1994, pp. 144–153.
[7] B. Alpern, F.B. Schneider, Defining liveness, Information Processing Letters 21 (1995) 185–191.
[8] R.J. Back, Refinement calculus, part II: parallel and reactive programs, in: Proceedings of REX Workshop: Proceedings on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, LNCS, vol. 430, 1989, pp. 42–66.
[9] R.J. Back, Refinement calculus, part I: sequential nondeterministic programs, in: Proceedings of REX Workshop: Proceedings on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, LNCS, vol. 430, 1989, pp. 67–93.
[10] R.J. Back, J. von Wright, Refinement Calculus: A Systematic Introduction, Springer, 1998.
[11] H. Barringer, R. Kuiper, A. Pnueli, Now you may compose temporal logic specifications, in: Proceedings of 16th STOC, 1984, pp. 51–63.
[12] H. Barringer, R. Kuiper, A. Pnueli, A compositional temporal approach to a CSP-like language, in: Proceedings of IFIP conference on the Role of Abstract Models in Information Processing, 1985, pp. 207–227.
[13] A. Bouajjani, J.C. Fernandez, S. Graf, C. Rodriguez, J. Sifakis, Safety for branching time semantics, in: Proceedings of ICALP'91, LNCS, vol. 510, 1991, pp. 76–92.
[14] G. Boudol, Atomic actions, Bulletin European Association of the Computer Science 38 (1989) 136–144.
[15] J.C. Bradfield, Verifying Temporal Properties of Systems, Birkhäuser Boston, Mass, 1992.
[16] Business Process Execution Language for Web Service (WS-BPEL), version 2.0, April 2007. Available from: <http://docs.oasis-open.org/wsbpel/2.0/os/wsbpel-v2.0-os.html/>.
[17] E.M. Clarke, E.A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic, in: IBM Workshop on Logics of Programs, LNCS, vol. 131, 1981, pp. 52–71.
[18] E.M. Clarke, E.A. Emerson, Using branching time temporal logic to synthesize synchronization skelotons, Science of Computer Programming 2 (3) (1982) 241–266.
[19] P. Degano, R. Gorrieri, Atomic Refinement in Process Description Languages, TR 17-91 HP Pisa Center, 1991.
[20] A. Elmagarmid, Database Transaction Models for Advanced Applications, Elsevier, 1992.
[21] E.A. Emerson, C.S. Jutla, Tree automata, μ-calculus, and determinacy, in: Proceedings of 33rd IEEE Symp. on Found. of Comp. Sci., 1991, pp. 368–377.
[22] R. Gorrieri, A. Rensink, Action refinement, Handbook of Process Algebra, Elsevier Science, 2001, pp. 1047–1147.
[23] S. Graf, J. Sifakis, A logic for the description of non-deterministic programs and their properties, Information and Control 68 (1986) 254–270.
[24] J. He, C.A.R. Hoare, J. Sanders, Data refinement refined, in: Proceedings of ESOP'86, LNCS, vol. 213, 1986, pp. 187–196.
[25] J. He, Process simulation and refinement, Formal Aspects of Computing 1 (3) (1989) 229–241.
[26] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[27] M. Huhn, Action refinement and properties inheritance in systems of sequential agents, in: Proceedings of CONCUR'96, LNCS, vol. 1119, 1996, pp. 639–654.
[28] D. Janin, I. Walukiewicz, On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic, in: Proceedings of CONCUR'96, LNCS, vol. 1119, 1996, pp. 263–277.
[29] M.B. Josephs, A state-based approach to communicating processes, Distributed Computing 3 (1) (1988) 9–18.
[30] L. Lamport, Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2002.
[31] M. Lange, C. Stirling, Model checking fixed point logic with chop, in: Proceedings of FOSSACS'02, LNCS, vol. 2303, 2002, pp. 250–263.
[32] M. Lange, Local model checking games for fixed point logic with chop, in: Proceedings of CONCUR'02, LNCS, vol. 2421, 2002, pp. 240–254.
[33] K.G. Larsen, X.X. Liu, Compositionality through an operational semantics of contexts, in: Proceedings of ICALP'90, LNCS, vol. 443, 1990, pp. 526–539.
[34] K.G. Larsen, X.X. Liu, Equation solving using modal transition systems, in: Proceedings of LICS'90, 1990, pp. 108–117.
[35] K.G. Larsen, B. Thomsen, A modal process logic, in: Proceedings of LICS'88, 1988, pp. 203–210.
[36] G.T. Leavens, A.T. Baker, C. Ruby, JML: a notation for detailed design; in: Haim Kilov, Bernhard Rumpe, Ian Simmonds (Eds.), Behavioral Specifications of Businesses and Systems, Kluwer, 1999, pp. 175–188 (Chapter 12).
[37] X.X. Liu, Specification and Decomposition in Concurrency, Ph.D. Thesis, Department of Mathematics and Computer Science, Aalborg University Center, Demark, 1992.
[38] M. Majster-Cederbaum, F. Salger, Correctness by construction: towards verification in hierarchical system development, in: Proceedings of SPIN'00, LNCS, vol. 1885, 2000, pp. 163–180.
[39] M. Majster-Cederbaum, F. Salger, Towards the hierarchical verification of reactive systems, Theoretical Computer Science 318 (3) (2003) 243–296.
[40] M. Majster-Cederbaum, N.J. Zhan, H. Fecher, Action refinement from a logical point view, in: Proceedings of VMCAI'03, LNCS, vol. 2575, 2003, pp. 253–267.
[41] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[42] M. Müller-Olm, A Modal Fixpoint Logic with Chop, in: Proceedings of STACS'99, LNCS, vol. 1563, 1999, pp. 510–520.
[43] P. Niebert, A ν-calculus with local views for systems of sequential agents, in: Proceedings of MFCS'95, LNCS, vol. 969, 1995, pp. 563–573.
[44] M. Nielsen, U. Engberg, K.S. Larsen, Fully abstract models for a process language with refinement, in: Proceedings of REX School on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS, vol. 354, 1989, pp. 523–548.
[45] A. Pnueli, The temporal logic of programs, in: Proceedings of 18th STOC, 1977, pp. 232–239.
[46] A. Rensink, Models and Methods for Action Refinement, Ph.D. thesis, University of Twente, Enschede, The Netherlands, 1993.
[47] R. Rosner, A. Pnueli, A choppy logic, in: Proceedings of LICS'86, 1986, pp. 306–313.
[48] J. Sifakis, Research directions for concurrency, ACM Computing Surveys 28 (4es) (1996) 55.
[49] C. Stirling, Modal and Temporal Properties of Processes, Springer (Texts in Computer Science), 2001.
[50] A. Tarski, A lattice-theoretical fixpoint theorem and its application, Pacific Journal of Mathematics 5 (1955) 285–309.
[51] M. Viswanathan, R. Viswanathan, Foundations for circular compositional reasoning, in: Proceedings of of ICALP'01, LNCS, vol. 2076, 2001, pp. 835–847.
[52] R.J. van Glabbeek, F.W. Vaandrager, Petri nets models for algebraic theories of concurrency, in: Proceedings of PARLE'87, vol. II (Parallel Languages), LNCS, vol. 259, 1987, pp. 224–242.
[53] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, Journal of ACM 43 (3) (1996) 555–600.
[54] L. Wang, S.D. Stoller, Runtime analysis of atomicity for multi-threaded programs, IEEE Transactions on Software Engineering 32 (2) (2006) 93–110.
[55] J. Woodcock, C. Morgan, Refinement of state-based concurrent systems, in: Proceedings of VDM '90: Proceedings of the Third International Symposium of VDM Europe on VDM and Z – Formal Methods in Software Development, LNCS, vol. 428, 1990, pp. 340–351.
[56] N.J. Zhan, Combining hierarchical specification with hierarchical implementation, in: Proceedings of of ASIAN'03, LNCS, vol. 2896, 2003, pp. 110–124.
[57] N.J. Zhan, J.Z. Wu, Compositionality of fixpoint logic with chop, in: Proceedings of ICTAC'05, LNCS, vol. 3722, 2005, pp. 136–150.