On hierarchically developing reactive systems

Naijun Zhan\textsuperscript{a,}\textsuperscript{*}, Mila Majster-Cederbaum\textsuperscript{b}

\textsuperscript{a} Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, South Fourth Street, No. 4, Zhong Guan Cun, 100190 Beijing, PR China
\textsuperscript{b} Lehrstuhl für Praktische Informatik II, Fakultät für Mathematik und Informatik, Universität Mannheim, D-727, 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.

© 2010 Elsevier Inc. All rights reserved.

**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, $\phi$ for its specification, $Q$ for the refinement of an abstract action $a$ in $P$, $\sim$ 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:

$$
\begin{align*}
P & \models \phi \\
\uplus & \\
P[a \sim Q] & \models \phi[a \sim Q]
\end{align*}
$$

* 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).

\textsuperscript{1} The first author is supported in part by the projects NKBPC-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.

Please cite this article in press as: N. Zhan, M. Majster-Cederbaum, On hierarchically developing reactive systems, Inform. Comput. (2010),
under some conditions, where \( \phi[a \sim Q] \) stands for substituting \( a \) and \( \lbrack a \rbrack \) in \( \phi \) by some formulae of the forms \( \lbrack a \rbrack_1 \lbrack a \rbrack_2 \ldots \lbrack a \rbrack_n \) and \( \lbrack a \rbrack_1 \lbrack a \rbrack_2 \ldots \lbrack a \rbrack_n \), respectively, where \( \langle a \rangle \) stands for the modality “for some \( a \)-successor” and \( \lbrack a \rbrack \) for the modality “for all \( a \)-successors”, and \( a_1 a_2 \ldots a_n \) is a run of \( Q \). These results support “a priori” verification in the following sense: assume that \( P \models \phi \) has been established and \( \phi \) is refined to \( \phi[a \sim Q] \) then we automatically obtain a process satisfying \( \phi[a \sim Q] \). The analogous remark is true when we refine \( P \) to \( P[a \sim Q] \), 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 \( P = a; b + a; c, \phi = \langle a \rangle, Q = \langle c \rangle ; \langle c \rangle ; \langle b \rangle \). It is obvious that \( P \models \phi \) and \( Q \models \langle \langle a \rangle \rangle [\langle c \rangle] [\langle b \rangle] \). It is expected that \( P[a \sim Q] = \langle \langle a \rangle \rangle [\langle c \rangle] [\langle b \rangle] \). 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 \( \Omega \) which maps the high-level specification \( \phi \) and the property \( \psi \) for the refinement of an abstract action \( a \) to a lower-level specification by substituting \( \psi \) for \( \langle a \rangle \) and \( \lbrack a \rbrack \) in \( \phi \). By choosing \( \psi \) appropriately, we can get the desired result. For example, in the above example, we can get \( \Omega \phi = \langle \langle a \rangle \rangle [\langle c \rangle] [\langle b \rangle] \). This 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 an

\[ \Omega \phi = \langle \langle a \rangle \rangle [\langle c \rangle] [\langle b \rangle] \]

The above theorem supports “a priori” verification in the following sense: assume that \( P \models \phi \) and \( \psi_1 \land \psi_2 \), then \( P[a \sim Q] = \Omega \phi \models \psi_1 \land \psi_2 \).

The above theorem supports “a priori” verification in the following sense: in the development process we start with \( P \models \phi \) and \( \psi_1 \land \psi_2 \), and either refine \( P \) and obtain automatically a (relevant) formula that is satisfied by \( P[a \sim Q] \); or, we refine \( \phi \) using \( \Omega \phi \models \psi_1 \land \psi_2 \) and \( \phi \), and obtain automatically a refined process \( P[a \sim Q] \) 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 \( \mu \)-calculus since most modal and temporal logics that are used as specification languages for concurrent systems can be defined in it. Unfortunately, the \( \mu \)-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 \( \phi \), and that \( Q \models \psi_1 \land \psi_2 \). After refining \( a \) by \( Q \) in \( P \), a specification for the refined system that one expects should be naturally \( \psi_1 \); \( \phi \) means that the behavior of the system can be divided into two successive segments such that the first segment satisfies \( \psi_1 \) and the second one meets \( \phi \). But \( \psi_1 \); \( \phi \) is no longer a formula of the \( \mu \)-calculus. Therefore, here we use FLC [42] as a specification language.

FLC is due to Müller-Olm [42], and is an extension of the \( \mu \)-calculus by introducing the chop operator “\( ; \)”. FLC is strictly more expressive than the \( \mu \)-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 \( \phi \) and the second meets \( \psi \). 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 \( \psi_1 \) 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, \((a || b)[a \sim a_1; a_2]\) means the parallel executions of \(a\) and \(b\) in which \(a\) is refined by \(a_1; a_2\). It is obvious that \(a || b\) satisfies \(a\), and \(a_1; a_2\) satisfies \(a_1\); \((a_2) \land \{b; ff\}\) which means that \(a_1; a_2\) firstly performs \(a_1\), then follows \(a_2\) but cannot perform \(b\). We expect that \(a || b\) meets \((a_1); ((a_2) \land \{b; ff\})\) 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**

\[
P :: \delta | \text{nil} | a \mid x \mid P; Q \mid P + Q \mid P \mid^A Q \mid \text{rec} \times. P \mid P[a \sim Q]
\]

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

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

- \(\delta\) is a deadlocked process that cannot proceed.
- \(\text{nil}\) is a process that does nothing but terminate. The difference between \(\delta\) and \(\text{nil}\) 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 \(\text{nil}\).
- \(x\) is used to evaluate process terms of the form \(\text{rec} \times. P\) (see the next section). In isolation, \(x\) behaves like \(\delta\).
- \(P + Q\) 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 \(Q\).
- \(P \mid^A Q\) means that the sub-systems \(P\) and \(Q\) can be performed concurrently. Each action in \(A\) has to be executed synchronously.
- \(\text{rec} \times. P\) denotes the system that executes the sub-system \(P\) recursively, i.e., \(\text{rec} \times. P = P[\text{rec} \times. P/x]\).
- \(P[a \sim Q]\) means that the system replaces the execution of the action \(a\) by the execution of the sub-system \(Q\) every time when the sub-system \(P\) 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 \(\text{bound}\) in a process term \(P\) iff it does occur within a sub-term of the form \(\text{rec} \times. P\), otherwise called \(\text{free}\). A process expression \(P\) is called \(\text{closed}\) iff all occurrences of each variable occurring in it are bound, otherwise called \(\text{open}\). We will use \(\text{fn}(P)\) to stand for the variables that have some free occurrence in \(P\), \(\text{bn}(P)\) for the variables that have some bound occurrence in \(P\). We say a process \(P\) is \(\text{terminated}\), if it does nothing but terminate (see Definition 2). A variable \(x \in \mathcal{X}\) is called \(\text{guarded}\) 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 if 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 F to stand for the set of closed terms of P in which neither δ, nor rec x occurs, that is, each process in F does not evolve to deadlock and has finite behavior.

As in [22], we require the following well-formedness conditions on 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 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 T1 for the process that may be refined and a transition system T2 for the refining process Q, and then replace each edge of T1 labeled by the action to be refined by T2.

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 P*, ranged over by s, . . . , is given by:

\[ s ::= \text{nil} \mid x \mid *s \mid s \mid P + Q \mid P \parallel Q \mid s[a \sim Q] \mid \text{rec } x. \ s \]

where \( a \in \mathcal{A}, x \in \mathcal{X}, P, Q \in \mathcal{P} \).

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 \( T \) be the least subset of \( \mathcal{P}^* \) such that

- \( \text{nil} \in T \);
- If \( s \in T \) then \( *s \in T \), \( \text{rec } x. \ s \in T \), and \( s[a \sim Q] \in T \);
- If \( s_1 \in T \) and \( s_2 \in T \) then \( s_1 \parallel s_2 \in T \), \( s_1 + s_2 \in T \) and \( s_1 ; s_2 \in T \).

where \( Q \in \mathcal{P} \).

We say a state \( s \) is terminated if \( s \in T \).

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

- If \( s \in T \) 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 \( s_1 \parallel s_2 \in \mathcal{A} \) and \( s_1 ; s_2 \in \mathcal{A} \);
- If \( s \in \mathcal{A} \) then \( s[a \sim Q] \in \mathcal{A} \).

where \( Q \in \mathcal{P} \).

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

We use \( T(s) \) as a shorthand for \( s \in T \) 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:

\[
\text{Act } a \xrightarrow{a} \text{nil} \quad \text{Nd } \frac{P \xrightarrow{a} s}{P + Q \xrightarrow{a} s, Q + P \xrightarrow{a} s}
\]
that may synchronize with the environment. The other rules can be conceived as usual. The above rules guarantee that the

For any closed term \( s \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)

\( \Rightarrow \)
Because a concrete process has a priority in parallel with an abstract process, $\sim$ is not preserved by $\parallel_A$. For example, $a_1; a_2 \sim [a \sim a_1; a_2]$, but $(a_1; a_2) \not\parallel [a \sim a_1; a_2] \parallel b$. 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 $\triangleq_A$. Obviously, $\sim_A$ is an equivalence relation over $\Pi^\star$.

The following theorem indicates that $\sim_A$ is a congruence too and a proper subset of $\sim$.

**Theorem 1.** $\sim_A$ is a congruence over $\Pi^\star$ and $\sim_A \subset \sim$.

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

- **Case 1:** $L = x, \delta, \text{nil}, a$.
  It is obvious.

- **Case 2:** $L = \cdot$.
  Since $s_1 \sim_A s_2$, there exists a strong bisimulation $R_A$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_A$. Let $R_2 \equiv \{ (s, s') \mid (s, s') \in R_A \}$. It is easy to show that $R_2$ is a strong bisimulation w.r.t. $\mathcal{A}$.

- **Case 3:** $L = s$ (the symmetric case is left to readers).
  Since $s_1 \sim_A s_2$, there exists a strong bisimulation $R_A$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_A$. Let $R_2 = R_A \cup R \cup l I = l d_\mathcal{A}$, where $R$ is a congruence too and a proper subset of $\Pi^\star$. The symmetry of $R_2$ can be derived from that of $R_A$ and $l d_\mathcal{A}$. Therefore, in order to prove $R_2$ is a strong bisimulation w.r.t. $\mathcal{A}$, we only need to prove $R_2$ is preserved by transitions subject to $\mathcal{A}$. Let $(s_1', s_2') \in R_2$, then (i) $(s_1', s_2') = (s_1; s_2; s)$ for some $(s_1, s_2) \in R_A$ or (ii) $(s_1', s_2') = (s', s')$ for some $(s', s') \in l d_\mathcal{A}$.

For (i), if $s_1' \xrightarrow{a} s_2'$, only if $s_1 \xrightarrow{a} s_1$ and $s_1' \equiv s_1$; by Seq-1 or $T(s_1), s \xrightarrow{a} s'$, and $s_1' \equiv s'$ by Seq-2. For the former, because $(s_1, s_2) \in R_A$, there exists $s_1$ such that $s_2 \xrightarrow{a} s_2$, $\mathcal{A}(s_1) \iff \mathcal{A}(s_1)$ and $(s_1, s_2) \in R_A$. By Seq-1, $s_1 \xrightarrow{a} s_1$; $s \xrightarrow{a} s_1$. By the definition of $R_A$, $(s_1, s_2, s') \in R_A$ and $\mathcal{A}(s_1, s_2) \iff \mathcal{A}(s_1, s_2)$. For the latter, we have $T(s_1)$ since $T(s_1)$ and $(s_1, s_2) \in R_A$. By Seq-2, $s \xrightarrow{a} s_1$. It is obvious that $(s', s') \in R_A$.

For (ii), it is easy to show $R_2$ is closed under transitions subject to $\mathcal{A}$ as well.

- **Case 4:** $L = +$ (symmetrically, $P + \cdot$ is left to readers), where $P = \mathcal{P}$.
  If $s_1, s_2 \not\in \mathcal{P}$, then $s_1 + P$ and $s_2 + P$ are not well defined. Hence the claim is true. Otherwise, since $s_1 \sim_A s_2$, there exists a strong bisimulation $R_A$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_A$.

$R_2 = R_A \cup R \cup l d_\mathcal{A}$. It is easy to show that $R_2$ is a strong bisimulation w.r.t. $\mathcal{A}$.

- **Case 5:** $L = |a \sim Q|$. Since $s_1 \sim_A s_2$, there exists a strong bisimulation $R_A$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_A$. Let $R_1 \equiv \{(s_1[a \sim Q], s_2[a \sim Q]) \mid (s_1, s_2) \in R_A\}, R_2 \equiv R_1 \cup \bigcup_{(s_1, s_2) \in R_1} l d_{Q_1} \cup l d_{Q_2}$.

It is easy to show that $R_2$ is symmetric. So we only need to prove $R_2$ is closed under transitions subject to $\mathcal{A}$ in order to show that $R_2$ is a strong bisimulation w.r.t. $\mathcal{A}$. Assume $(s_1, s_2) \in R_2$. Then (i) $\exists (s_1, s_1') \in R_2 \land s_1' \equiv s_1[a \sim Q] \land s_1' \equiv s_1[a \sim Q]$; or (ii) $\exists (s_1, s_1') \in l d_{Q_1} \land (s_1, s_1') \in R_2 \land s_1' \equiv s_1$; or (iii) $\exists (s_1, s_1') \in l d_{Q_2} \land s_1' \equiv s_1$. For (i), $s_1' \xrightarrow{a} s_1''$ only if $a \neq a$, and $s_1'' \equiv s_1[a \sim Q]$ by Ref-1; or (b) $s_1 \xrightarrow{a} s_1, Q \xrightarrow{a} Q'$ and $s_1'' \equiv s_1[a \sim Q]$ by Ref-2. For (a), since $(s_1, s_2) \in R_A$, there exists $s_1$ such that $s_2 \xrightarrow{a} s_1$ and $(s_1, s_2) \in R_A$. By Ref-1, $s_2[a \sim Q] \xrightarrow{b} s_2[a \sim Q]$. From the definition of $R_2$, $(s_1[a \sim Q], s_2[a \sim Q]) \in R_2$; for (b), as $(s_1, s_2) \in R_A$, there exists $s_2$ such that $s_2 \xrightarrow{a} s_1, Q \xrightarrow{a} Q$ and $s_2 \equiv s_1[a \sim Q]$ by Ref-2. For (a), since $(s_1, s_2) \in R_A$, there exists $s_1$ such that $s_2 \xrightarrow{a} s_1$ and $(s_1, s_2) \in R_A$. By Ref-1, $s_2[a \sim Q] \xrightarrow{b} s_2[a \sim Q]$. From the definition of $R_2$, $(s_1, s_2) \equiv s_1[a \sim Q], Q \equiv s_1[a \sim Q]$; or (ii) $\exists (s_1, s_2) \in R_2 \land s_1' \equiv s_1 \land s_2' \equiv s_2$; or (iii) $\exists (s_1, s_2) \land s_1' \equiv s_1 \land s_2' \equiv s_2$. Similarly to the above case, we can show $R_2$ is a strong bisimulation w.r.t. $\mathcal{A}$.

- **Case 6:** $L = \cdot |a \sim Q|$. (symmetrically, $\cdot |a \sim Q|$ is left to readers.)
  For $s_1 \sim_A s_2$, there exists a strong bisimulation $R_A$ w.r.t. $\mathcal{A}$ such that $(s_1, s_2) \in R_A$. Let $R_2 \equiv l d_{Q_1} \cup l d_{Q_2} \cup \{ (s', s') \mid (s, s') \in l d_{Q_1} \land (s', s') \in R_A \land (\mathcal{A}(s') \land \mathcal{A}(s_1)) \}$. The symmetry of $R_2$ is easy to get by that of $R_A$ and $l d_{Q_1}$.

Suppose $(s', s') \in l d_{Q_1} \land (s, s') \in R_A \land (\mathcal{A}(s') \land \mathcal{A}(s_1))$. Then (i) $\exists (s, s') \in l d_{Q_1} \land s' \equiv s \land s' \equiv s'$; or (ii) $\exists (s, s') \in R_A \land s_1 \equiv s_1 \land s_2 \equiv s_2$; or (iii) $\exists (s, s') \in l d_{Q_1} \land s_1 \equiv s_1 \land s_2 \equiv s_2$. For (i), $s' \equiv s'$; for (ii), $\mathcal{A}(s') \land \mathcal{A}(s_1)$. For (i) and (ii), it is easy
to show that $R'_A$ is closed under transitions subject to $A$. As for (iii), $s_1 \xrightarrow{a} s'_1$ only if (a) $s' \xrightarrow{a} s''$ and $a \not\in A \cap ab(s_1)$ by \text{ASyn-1}; or (b) $s_1 \xrightarrow{a} s'_1 \land a \not\in A \cup ab(s')$ by \text{ASyn-1} again; or (c) $s' \xrightarrow{a} s'' \land s_1 \xrightarrow{a} s'_1 \land a \in A \land A(s') \land A(s_1)$ according to \text{Syn}. For (a), applying \text{ASyn-1}, we have $s' \parallel A s_2 \xrightarrow{a} s'' \parallel A s_2$. From the definition of $R'_A$, $(s'' \parallel A s_1, s' \parallel A s_2) \in R'_A$. Furthermore, it is clear that $A(s'' \parallel A s_1)$ iff $A(s'' \parallel A s_2)$. Similarly, we can prove that in the cases (b) and (c), $R'_A$ is closed under transitions subject to $A$ as well.

Thus $s \parallel A s_1 \sim_A s \parallel A s_2$ since if $A(s) \lor \left(A(s_1) \land A(s_2)\right)$ then $(s \parallel A s_1, s \parallel A s_2) \in R'_A$; otherwise $s \parallel A s_1$ and $s \parallel A s_2$ both are not well formed.

• $C[-] \equiv \text{rec } x \cdot$

Let $R''_A \equiv \left(\text{rec } x. s_1/y, G[\text{rec } x. s_2/y]\right) \parallel G$ contains at most the variable $y$. Using the standard way, induction on the structure of $G$ (see [41]), we can prove $R''_A$ is a strong bisimulation w.r.t. $A$. Taking $G \equiv y$, it follows that $\text{rec } x. s_1 \sim_A \text{rec } x. s_2$.

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 **normalized** 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 $\mu$-calculus by introducing the chop operator “;”, which can express non-regular properties [42]. It is therefore strictly more powerful than the $\mu$-calculus, since it was proved in [21] that only regular properties can be defined in the $\mu$-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 $\phi_1$ and the second meets $\phi_2$. In addition, $\tau^2$ 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 $\sqrt{.}$, 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., $\sqrt{.}$ characterizes all terminated processes and does not hold for any deadlocked process, while $\land_{a \in \mathcal{Act}}[a]$; $ff$ 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 $\mathbf{Var}$ of variables, $tt$ and $ff$ be two propositional constants as usual, and $\sqrt{.}$ be another one that is used to indicate if a process is terminated.

Formulae of FLC are generated according to the following grammar:

$$\phi ::= tt | ff | \sqrt{.} | \tau | X | [a] | \langle a \rangle | \phi_1 \land \phi_2 | \phi_1 \lor \phi_2 | \phi_1; \phi_2 | \mu X.\phi | \nu X.\phi$$

where $X \in \mathbf{Var}$ and $a \in \mathcal{Act}$.

In the sequel, we use $@a$ to stand for $\langle a \rangle$ or $[a]$, $p$ for $tt, ff$ or $\sqrt{.}$, and $\sigma$ for $v$ or $\mu$, $\mathcal{Act}(\phi)$ for all actions that occur in $\phi$.

As in the modal $\mu$-calculus, the two fixpoint operators $\nu X.\phi$ and $\mu X.\phi$ 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 $\mathcal{fn}(\phi)$ to stand for the set of the variables which have free occurrence in $\phi$ and $\mathcal{bn}(\phi)$ for the set of the variables that have bound occurrence in $\phi$.

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

1. $@a$ and $p$ are guards;
2. if $\phi$ and $\psi$ are guards, so are $\phi \land \psi$ and $\phi \lor \psi$;
3. if $\phi$ is a guard, so are $\phi; \psi$ and $\sigma X.\phi$, where $\psi$ is any formula of FLC.

---

2 In [42], $\tau$ 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 \( C_{FLC} \) the set of formulae of FLC that are closed and guarded.

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

\[
f \subseteq f' \text{ iff } f(A) \subseteq f'(A) \text{ for all } A \subseteq S
\]

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: \( (2^S \rightarrow 2^S) \rightarrow (2^S \rightarrow 2^S) \) exist. They are used to interpret fixed point formulae of FLC.

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

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

\[
\begin{align*}
C_{FLC}^\rho(tt)(A) &= S \\
C_{FLC}^\rho(ff)(A) &= \emptyset \\
C_{FLC}^\rho(\sqrt{.})(A) &= \{ P \mid P \in S \wedge \tau(P) \} \\
C_{FLC}^\rho(\tau)(A) &= A \\
C_{FLC}^\rho(\varnothing)(A) &= \rho(X) \\
C_{FLC}^\rho([a])(A) &= \{ P \mid \neg \tau(P) \wedge \exists P' : P \overset{a}{\rightarrow} P' \rightarrow P' \in A \} \\
C_{FLC}^\rho([a])(A) &= \{ P \mid \exists P' : P \overset{a}{\rightarrow} P' \wedge P' \in A \} \\
C_{FLC}^\rho(\phi_1 \land \phi_2)(A) &= C_{FLC}^\rho(\phi_1)(A) \cap C_{FLC}^\rho(\phi_2)(A) \\
C_{FLC}^\rho(\phi_1 \lor \phi_2)(A) &= C_{FLC}^\rho(\phi_1)(A) \cup C_{FLC}^\rho(\phi_2)(A) \\
C_{FLC}^\rho(\phi_1 \cdot \phi_2)(A) &= C_{FLC}^\rho(\phi_1) \cdot C_{FLC}^\rho(\phi_2) \\
C_{FLC}^\rho(\muX.\phi)(A) &= \cap\{ f \in MPTT \mid C_{FLC}^{\rho[X \sim f]}(\phi) \subseteq f \} \\
C_{FLC}^\rho(\nuX.\phi)(A) &= \cup\{ f \in MPTT \mid C_{FLC}^{\rho[X \sim f]}(\phi) \supseteq f \}
\end{align*}
\]

where \( A \subseteq S \), and \( \cdot \) stands for the compositional operator over functions.

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

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

**Lemma 2.** If \( X \notin fn(\phi) \), then \( C_{FLC}^\rho(\phi)(A) = C_{FLC}^{\rho[X \sim f]}(\phi)(A) \) for any environment \( \rho \), \( A \subseteq S \) and \( f \in MPTT \).

The lemma below claims that the interpretation of a formula \( \phi \) is monotonic on environments, i.e.,

**Lemma 3.** If \( \rho \subseteq \rho' \), then \( C_{FLC}^\rho(\phi)(A) \subseteq C_{FLC}^{\rho'}(\phi)(A) \) for any \( \phi \) of FLC and \( A \subseteq S \), where \( \rho \subseteq \rho' \) means for any \( X \) and \( A \subseteq S \), \( \rho(X)(A) \subseteq \rho'(X)(A) \).
In the sequel, in order to avoid the excessive use of parentheses, we assume $\alpha$ has higher priority than $\lor$ and $\land$, $\lor$ and $\land$ have higher priority than $\nu X$ and $\mu X$, while $\nu X$ and $\mu X$ 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

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

where $i, I, \alpha \in \mathbb{On}$, the ordinals.

For convenience, in what follows we use $\phi[\psi/\chi]$ to stand for substituting $\psi$ for each occurrence of $\chi$ in $\phi$, where $\chi \in \{X, \sqrt{\_}, \langle a \rangle, \lceil a \rceil\}$.

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

$$
\begin{align*}
C_0^T(\bigwedge_{i \in I} \phi_i)(A) &= \bigcap_{i \in I} C_0^T(\phi_i)(A), \\
C_0^T(\bigvee_{i \in I} \phi_i)(A) &= \bigcup_{i \in I} C_0^T(\phi_i)(A), \\
C_0^T(\mu^0X.\phi_1) &= C_0^T(\phi_1), \\
C_0^T(\mu^\alpha X.\phi_1) &= C_0^T(\phi_1[\mu^{\alpha}X.\phi_1/X]), \\
C_0^T(\nu^\alpha X.\phi_1) &= C_0^T(\nu^\alpha X.\phi_1), \\
C_0^T(\nu^{\alpha+1} X.\phi_1) &= C_0^T(\nu^{\alpha} X.\phi_1), \\
C_0^T(\nu^{\lambda} X.\phi_1) &= C_0^T(\nu^{\lambda} X.\phi_1),
\end{align*}
$$

where $A \subseteq S$, $\alpha, \lambda \in \mathbb{On}$ 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 $\alpha$ such that $\mu X.\phi \Leftrightarrow \mu^{\alpha}X.\phi$. Analogously, for any $\nu X.\phi$, there exists an ordinal $\alpha$ 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, $\alpha$ can be replaced by $\omega$. 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 $\alpha$ in Proposition 1 can be replaced by $2^{\omega_1}$, where $\omega_1$ stands for the first uncountable limit ordinal, because the cardinality of $\{f : 2^5 \hookrightarrow 2^5 \mid f \text{ is a monotonic w.r.t. the inclusion ordering on } 2^5\}$ is at most $2^{\omega_1}$.

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

We will use $\mathcal{L}_{AFLC}$ 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 $\mu$-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 $\phi$ 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**

\begin{align*}
\text{N} & : \phi \leftrightarrow \phi; \tau \leftrightarrow \phi \\
\text{T1} & : \forall [a]; tt \leftrightarrow tt \\
\text{T2} & : \sqrt{\land [a]}; tt \leftrightarrow ff \\
\text{C} & : (\phi; \psi); \varphi \leftrightarrow \phi; (\psi; \varphi) \\
\text{DC} & : (\lor_{i \in I} \phi_i); \varphi \leftrightarrow \lor_{i \in I} (\phi_i; \varphi)
\end{align*}

For any given formula \(\phi\), we define its beginning atomic sub-formulae, denoted \(BSub(\phi)\), as:

\[BSub(\phi) \equiv \begin{cases} 
\{\phi\} & \text{if } \phi = p, X, \otimes, \tau \\
\bigcup_{i \in I} BSub(\phi_i) & \text{if } \phi = \land_{i \in I} \phi_i \text{ or } \phi = \lor_{i \in I} \phi_i \\
BSub(\phi_1) & \text{if } \phi = \phi_1; \phi_2 \text{ and } \tau \not\in BSub(\phi_1) \\
(\text{BSub}(\phi_1) - \{\tau\}) \cup \text{BSub}(\phi_2) & \text{if } \phi = \phi_1; \phi_2 \text{ and } \tau \in \text{BSub}(\phi_1) \\
\text{BSub}(\phi_1) & \text{if } \phi = \sigma^n X.\phi_1
\end{cases}\]

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

\[ESub(\phi) \equiv \begin{cases} 
\{\phi\} & \text{if } \phi = p, X, \otimes, \tau \\
\bigcup_{i \in I} ESub(\phi_i) & \text{if } \phi = \land_{i \in I} \phi_i \text{ or } \phi = \lor_{i \in I} \phi_i \\
ESub(\phi_2) & \text{if } \phi = \phi_1; \phi_2 \text{ and } \tau \not\in ESub(\phi_2) \\
(ESub(\phi_2) - \{\tau\}) \cup ESub(\phi_1) & \text{if } \phi = \phi_1; \phi_2 \text{ and } \tau \in ESub(\phi_2) \\
ESub(\phi_1) & \text{if } \phi = \sigma^n X.\phi_1
\end{cases}\]

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

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

\[BSub((a') \lor \tau); ((a); (b) \land [c]; (e); [f]); ((\tau \lor [b'])) = \{(a'), (a), [c]\}\text{, whereas}
\[ESub((a') \lor \tau); ((a); (b) \land [c]; (e); [f]); ((\tau \lor [b'])) = \{(b), [f], [b']\}\]

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

**Proof.** Using Lemma 4, by induction on \(\phi\).

**Definition 9.** A formula \(\phi\) is said to be existential if for any \(a \in Act, \lceil a \rceil \not\in BSub(\phi)\). We use \(\mathcal{E}\mathcal{F}\) to stand for the set of existential formulae. Dually, a formula \(\phi\) is said to be universal if for any \(a \in Act, \lceil a \rceil \not\in BSub(\phi)\). We use \(\mathcal{U}\mathcal{F}\) to stand for the set of universal formulae. A formula is called a property formula if \(\phi \Leftrightarrow \phi_1 \land \phi_2\), where \(\phi_1 \in \mathcal{E}\mathcal{F}\) and \(\phi_2 \in \mathcal{U}\mathcal{F}\). The set of property formulae is denoted by \(\mathcal{P}\mathcal{F}\).

According to Definition 9, the following propositions are obvious.

**Proposition 3.** \(\mathcal{E}\mathcal{F}, \mathcal{U}\mathcal{F}\) are closed under all operators of the logic. That is, \(\land_{i \in I} \phi_i \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) and \(\lor_{i \in I} \phi_i \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) iff \(\phi_i \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) for any \(i \in I; \phi_1 \land \phi_2 \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) iff \(\phi_1 \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) and \(\phi_2 \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\); \(\sigma^a X.\phi, \sigma X.\phi \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\) iff \(\phi \in \mathcal{E}\mathcal{F}(\mathcal{U}\mathcal{F})\).

Proposition 4

1. If \( \phi \in \mathcal{E} \) and \( \tau \not\in BSub(\phi) \), then for any formula \( \psi \) of AFLC, \( \phi, \psi \in \mathcal{E} \).

2. If \( \phi \in \mathcal{U} \) and \( \tau \not\in BSub(\phi) \), then for any formula \( \psi \) of AFLC, \( \phi, \psi \in \mathcal{U} \).

Proposition 5

1. For any closed formula \( \phi \in \mathcal{U} \), if \( P_1 |\phi, P_2 |\phi \) and \( P_1 + P_2 \) is well formed, then \( P_1 + P_2 |\phi \).

2. For any closed formula \( \phi \in \mathcal{E} \), if \( P_1 |\phi \) and \( P_1 + P_2 \) is well formed, then \( P_1 + P_2 |\phi \).

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

Lemma 7. Let \( \phi \) be a closed formula in \( \mathcal{U} \). If \( c \not\in Act(\phi) \), \( \phi \not\leftrightarrow ff \) and \( \tau \), \( \sqrt{\phi} \not\in BSub(\phi) \), then \( c; P \models \phi; \psi \).

Proof. By induction on \( \phi \).

- \( \phi = \mathcal{U} \), \( \alpha \), \( \tau \)

 Impossible according to the condition and the definition of \( \mathcal{U} \).

- \( \phi = \mathcal{E} \)

 It is trivial by Definition 7.

- \( \phi = [a] \)

 As \( c \not\in Act(\phi) \), it follows that \( a \neq c \). According to Definition 7, it is easy to see that \( c; P \models [a]; \psi \).

- \( \phi = \bigwedge_{i \in I} \phi_i \)

 Because \( \phi \in \mathcal{U} \), \( \phi \not\leftrightarrow ff \) and \( \tau \), \( \sqrt{\phi} \not\in BSub(\phi) \), we obtain that \( \phi_i \in \mathcal{U} \), \( \phi_i \not\leftrightarrow ff \) and \( \tau \), \( \sqrt{\phi} \not\in BSub(\phi_i) \) for each \( i \in I \). Moreover, it can be shown that \( c \not\in Act(\phi_i) \) as \( c \not\in Act(\phi) \) for all \( i \in I \). Thus, by the induction hypothesis, it derives that

\[ c; P \models \phi_i; \psi \], \( \psi \in \mathcal{E} \), \( i.e., \), \( c; P \models (\bigwedge_{i \in I} \phi_i); \psi \) by IC.

- \( \phi = \mathcal{E} \)

 Similarly to the above case.

- \( \phi = \phi_1; \phi_2 \)

 By Lemma 6, without loss of generality,

\[ \phi_1 \leftrightarrow \bigwedge_{i=1}^{k} \left( \tau \wedge \bigwedge_{j=1}^{m} \psi_{1,j} \right) \lor \bigwedge_{i=k+1}^{m} \left( \bigwedge_{j=1}^{n} \psi_{i,j} \right) \]

where \( USF(\psi_{1,j}) \) and \( \psi \not\in BSub(\psi_{1,j}) \) 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 \( \bigwedge_{j=1}^{n} \psi_{h,j} \not\leftrightarrow ff \). Thus, \( \phi_1; \phi_2 \leftrightarrow \bigwedge_{i=1}^{k} (\tau \wedge \bigwedge_{j=1}^{n} \psi_{1,j}) \lor \bigwedge_{i=k+1}^{m} (\bigwedge_{j=1}^{n} \psi_{i,j}) \).

- \( \phi = \mathcal{U} \), \( \alpha \), \( \tau \)

 By DC and \( \phi \) since \( \phi \notin BSub(\phi) \) by the definition and the assumption. As \( c \not\in Act(\phi) \), we have \( c \not\in Act(\phi_2) \).

- \( \phi = \mathcal{E} \)

 Therefore, by the induction hypothesis, we obtain \( c; P \models (\bigwedge_{j=1}^{n} \psi_{h,j}) \lor \psi \) and \( c; P \models \phi_2; \psi \). Using N and IC, we have \( c; P \models (\tau \wedge \bigwedge_{j=1}^{n} \psi_{h,j}) \lor \phi_2; \psi \). According to Definition 7, we get \( c; P \models (\phi_1; \phi_2); \psi \).

- \( \phi = v^\beta X.\phi' \)

 This case proceeds by induction on \( \alpha \).

\[ \alpha = 0 \] It is trivial.

\[ \alpha = \lambda \], \( \lambda \) is a limit ordinal.

 Since \( v^\beta X.\phi' \in \mathcal{U} \), \( \phi \not\leftrightarrow ff \), \( \psi \}, \( c \not\in Act(v^\beta X.\phi') \) and \( \tau \), \( \sqrt{\phi} \not\in BSub(v^\beta X.\phi') \), we obtain that \( v^\beta X.\phi' \) meets the listed requirements as well as each \( \beta < \lambda \). By the local induction hypothesis, we have for any \( \beta < \lambda \), \( c; P \models (v^\beta X.\phi'); \psi \). By IC, it follows that \( c; P \models (v^\beta X.\phi'); \psi \).

\[ \alpha = \beta + 1 \]

Thus, \( v^{\beta+1} X.\phi' \leftrightarrow \phi'[v^\beta X.\phi'] \). It is easy to see that \( \phi' \in \mathcal{U} \), \( c \not\in Act(\phi'[v^\beta X.\phi']) \), \( \phi'[v^\beta X.\phi'] \not\leftrightarrow ff \), \( \sqrt{\phi} \not\in BSub(\phi'[v^\beta X.\phi']) \) according to the assumption. We will prove this sub-case by the case analysis on the structure of \( \phi' \).

- \( \phi' = tt \), \( \tau \), \( [a] \)

 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 $\phi = \phi_1; \phi_2$.

• $\phi = \sigma X.\phi''$

This sub-sub-case can be proved same as the cases when $\phi = \nu X.\phi'$ and $\phi = \mu X.\phi'$.

• $\phi = \mu X.\phi'$

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 $\varphi$ with $\tau$.

**Corollary 1.** Let $\phi$ be closed in $\mathcal{L}$. If $c \notin \text{Act}(\phi)$, $\phi \not\equiv \mathcal{F}$ and $\sqrt{\cdot}$, $\tau \not\in \text{BSub}(\phi)$, 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 $\prec$. Therefore, we require that $\phi_1; \phi_2$ is well founded.

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

**Proof.** The proof proceeds by induction on the structure of $\phi$ w.r.t. $\prec$.

**Base cases:**

• $\phi = \tau$

Since $P \models \tau; \sqrt{\cdot}$, it follows that $P \models \sqrt{\cdot}$ by N. Hence, $T(P)$. Thus, $P; Q \models \phi[\tau / \sqrt{\cdot}], \psi$ by Seq-2, N and the assumption $Q \models \psi$.

• $\phi = tt$ or $\mathcal{F}$

It is easy.

• $\phi = \sqrt{\cdot}$

From $P \models \phi; \sqrt{\cdot}$, we have $P \models \sqrt{\cdot}$ by P1. Thus, $T(P)$. Hence, $P; Q \models \phi[\tau / \sqrt{\cdot}], \psi$ by Seq-2, N and the assumption $Q \models \psi$. 

Induction hypothesis:

Since $P \models (a)$, it follows that there exists $P'$ such that $P \rightarrow a P'$ and $T(P')$. Thus, $P; Q \rightarrow P$ by Seq-2. Furthermore, because $Q \models \psi$, we obtain that $P; Q \models \phi(\tau/\sqrt{\tau}); \psi$.

$\phi = [a]$

It is easy to show that $\neg T(P)$ and $\forall P'. P \rightarrow a P'$ implies $T(P')$ because $P \models [a]$; $\sqrt{\tau}$. On the other hand, by Seq-2, it can be shown that $\forall R P; Q \rightarrow P \rightarrow R \rightarrow Q$. By Theorem 2, $P; Q \models [a]; \psi$.

Induction steps:

$\phi = \Lambda_{i \in\ell} \phi_i$

Since $P \models (\Lambda_{i \in\ell} \phi_i); \sqrt{\tau}$, it follows that $P \models \phi_i; \sqrt{\tau}$ for any $i \in I$ by IC. Besides, it is clear that $\phi_i$ is PNF since $\phi$ is PNF. Whence, we have that $P; Q \models \phi_i(\tau/\sqrt{\tau}); \psi$ by (IH) for each $i \in I$. Thus, we have $P; Q \models (\Lambda \phi_i(\tau/\sqrt{\tau})); \psi$ from IC.

$\phi = \vee_{i \in\ell} \phi_i$

Similar to the above case.

$\phi = \phi_1 \land \phi_2$ (W.l.o.g., assume $\phi_2 \neq \psi$ by $\tau$)

This case is committed by induction on $\phi_1$ w.r.t. $<$ as follows:

$\phi_1 = tt or \overline{tt}$

It is easy to show by $P_1$.

$\phi_1 = \sqrt{\tau}$

This violates the assumption that $\sqrt{\tau}$ does not occur in $\phi$, and needs therefore not be considered.

$\phi_1 = \tau$

For $P \models (\phi_1; \phi_2); \sqrt{\tau}$, it follows that $P \models \phi_2; \sqrt{\tau}$ by N. Since $\phi_2$ is a proper sub-formula of $\phi$, we get $P; Q \models \phi_2(\tau/\sqrt{\tau}); \psi$ by (IH). Therefore, $P; Q \models (\phi_1; \phi_2)(\tau/\sqrt{\tau}); \psi$ by N.

$\phi_1 = [a]$

As $P \models (a); \phi_2; \sqrt{\tau}$, there exists $P'$ such that $P \rightarrow a P'$ and $P' \models \phi_2; \sqrt{\tau}$. Because $\phi_2 \neq \psi$, it follows that $P; Q \models \phi_2(\tau/\sqrt{\tau}); \psi$ by (IH). Thus, by Seq-1, $P; Q \models ([a]; \phi_2)(\tau/\sqrt{\tau}); \psi$.

$\phi_1 = \Lambda_{i \in\ell} \phi_i$

Since $P \models (\Lambda_{i \in\ell} \phi_i); \sqrt{\tau}$, it follows $P \models \Lambda_{i \in\ell}((\phi_i; \phi_2); \sqrt{\tau})$ from IC. Therefore $P \models (\phi_i; \phi_2); \sqrt{\tau}$ for each $i \in I$. Obviously, $\phi_1; \phi_2 < \Lambda_{i \in\ell} \phi_i; \phi_2$. It follows that $P; Q \models (\phi_i; \phi_2)(\tau/\sqrt{\tau}); \psi$ for any $i \in I$ from (IH). Thus, $P; Q \models ((\Lambda_{i \in\ell} \phi_i); \phi_2)(\tau/\sqrt{\tau}); \psi$ by IC.

$\phi_1 = \vee_{i \in\ell} \phi_i$

Similar to the above case.

$\phi_1 = \phi_1; \phi''$

By C, $(\phi_1; \phi'' \Leftrightarrow \phi_1; \phi'' \Leftrightarrow \phi_1; \phi_2)$. Thus, it follows that $P \models \phi_1; (\phi'' \Leftrightarrow \phi_2); \sqrt{\tau}$ because $P \models (\phi'; \phi''); \phi_2; \sqrt{\tau}$. On the other hand, it is easy to see that $(\phi'; \phi'') < (\phi'; \phi'') \phi_2; \sqrt{\tau}$ by the definition of $<$. So, we obtain that $P; Q \models (\phi'; (\phi'' \phi_2))(\tau/\sqrt{\tau}); \psi$ by (IH) and therefore $P; Q \models ((\phi'; \phi''); \phi_2)(\tau/\sqrt{\tau}); \psi$ by applying C.

$\phi_1 = v^{\lambda} X \phi'$

We will use the following claim to justify this case, i.e.,

**Claim 1.** If $P' \models (v^{\gamma} X.\sigma_{\gamma_1}^{\beta_1+1} X_{\gamma_1} \cdot \sigma_{\beta_2+1}^{n+1} X_{\beta_2} \cdot \phi_2); \sqrt{\tau}$ and $Q' \models \chi$, then $P'; Q' \models ([v^{\lambda} X.\sigma_{\lambda_1}^{\beta_1+1} X_{\lambda_1} \cdot \sigma_{\beta_2+1}^{n+1} X_{\beta_2} \cdot \phi_2); \sqrt{\tau} \chi$).

**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 $\lambda$ is a limit ordinal

   Thus, $P' \models (v^{\lambda} X.\sigma_{\lambda_1}^{\beta_1+1} X_{\lambda_1} \cdot \sigma_{\beta_2+1}^{n+1} X_{\beta_2}; \phi_2); \sqrt{\tau}$ iff $\forall \beta < \lambda. P' \models (v^{\beta} X.\sigma_{\beta_1+1}^{\lambda_1} \cdot \sigma_{\beta_2+1}^{n+1} X_{\beta_2}; \phi_2); \sqrt{\tau}$. By the local induction hypothesis, it immediately follows

   $P'; Q' \models (v^{\lambda} X.\sigma_{\lambda_1}^{\beta_1+1} X_{\lambda_1} \cdot \sigma_{\beta_2+1}^{n+1} X_{\beta_2}; \phi_2)(\tau/\sqrt{\tau}); \chi$.

3. $\alpha = \beta + 1$
By the semantics of AFLC,
\[ v^{\beta_1+1}\sigma_1^{\beta_1+1}X_1, \ldots, \sigma_n^{\beta_n+1}X_n, \psi \]
\[ \Leftrightarrow \varphi[\varphi'/X]\{\varphi'_1/X_1\} \cdots \{\varphi'_n/X_n\} \], where,
\[ \varphi' = v^\beta X, \sigma_1^{\beta_1+1}X_1, \ldots, \sigma_n^{\beta_n+1}X_n, \varphi, \]
\[ \varphi'_1 = \sigma_1 X_1, \sigma_2^{\beta_2+1}X_2, \ldots, \sigma_n^{\beta_n+1}X_n, \varphi[\varphi'/X], \]
\[ \vdots \]
\[ \varphi'_i = \sigma_i^{\beta_i}X_i, \sigma_{i+1}^{\beta_{i+1}}X_{i+1}, \ldots, \sigma_n^{\beta_n+1}X_n, \varphi[\varphi'/X]\{\varphi'_1/X_1\} \cdots \{\varphi'_{i-1}/X_{i-1}\}, \]
\[ \vdots \]
\[ \varphi'_n = \sigma_n^{\beta_n}X_n, \varphi[\varphi'/X]\{\varphi'_1/X_1\} \cdots \{\varphi'_{n-1}/X_{n-1}\}. \]

For brevity, we use \( \{\varphi^*\} \) 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 \( \varphi \).
(a) \( \varphi = p \)
   It is straightforward.
(b) \( \varphi = \tau \)
   Using (IH), it is easy to show.
(c) \( \varphi = X \) or \( \varphi = Y \) where \( i = 1, \ldots, n \)
   It is trivial by the local induction hypothesis.
(d) \( \varphi = \@ \)
   Similar to the subcases when \( \phi_1 = \@ \).
(e) \( \varphi = \phi'_1 \vee \phi'_2 \)
   Thus, \( P' \models (\phi'_1[\varphi^*]; \phi'_2); \chi \) for \( i = 1, 2 \) as \( P' \models (\varphi'; \phi_2); \chi \).
   Applying (IH), we get \( P'; Q' \models (\phi'_1[\varphi^*]; \phi'_2); \chi \).
   Hence,
   \[ P'; Q' \models (v^{\beta_1+1}\sigma_1^{\beta_1+1}X_1, \ldots, \sigma_n^{\beta_n+1}X_n, \varphi)[\varphi'/X]; \chi \]

By applying (IH). By C, it follows that
\[ P'; Q' \models (v^{\alpha}X, \sigma_1^{\beta_1+1}X_1, \ldots, \sigma_n^{\beta_n+1}X_n, \varphi); \chi \]

(h) \( \varphi' = \sigma Y, \phi'' \) or \( \sigma^{\lambda} Y, \phi \) where \( \lambda \) 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
P; Q \sim P and \phi[τ/\sqrt{}]; ψ′ ⇔ ϕ[τ/\sqrt{}], it is easy to see that P; Q \models \phi[τ/\sqrt{}]; ψ′ from P \models ϕ; √. However, obviously Q \not\models ψ′.

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 \(〈a\rangle\) 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., \(〈a\rangle\) 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, \(P ≡ a; b + a; c \models 〈a〉; 〈b〉, a_1; 〈a_2〉, 〈a_2〉, 〈a_2〉\), but \(P[a ∼ a_1; a_2] \not\models ([〈a_1〉; 〈a_2〉); 〈b〉\), since in the high-level specification, \(〈a〉; 〈b〉\) 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., \(ψ ∈ \mathcal{P.F}\). \([a]\) will be replaced by \(ψ_2\), and \(〈a〉\) 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, \(\mathcal{P.F}\) is powerful enough to define the properties of reactive systems.

Therefore, we define the refinement mapping as follows.

**Definition 10.** Suppose \(ψ\) is a high-level specification, \(a\) is an abstract action to be refined, and \(ψ_1 \in \mathcal{E.F}\) and \(ψ_2 \in \mathcal{U.F}\) and \(ψ_1 \land ψ_2 \in \mathcal{P.F}\) is the description of the refinement of \(a\). We define the refinement mapping, denoted by \(Ω(φ, ψ_1, ψ_2, a)\), as follows:

\[
Ω(φ, ψ_1, ψ_2, a) ≡ φ[ψ_1[τ/\sqrt{}]/〈a〉, ψ_2[τ/\sqrt{}]/[a]].
\]

Sometimes, for brevity, we directly write \(Ω(φ, ψ, a)\) for \(Ω(φ, ψ_1, ψ_2, a)\) if \(ψ = ψ_1 \land ψ_2 \in \mathcal{P.F}\), where \(ψ_1 \in \mathcal{E.F}\) and \(ψ_2 \in \mathcal{U.F}\), 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

\[
Ω(φ_1{φ_2/X}, ψ, a) ⇔ Ω(φ_1, ψ, a){Ω(φ_2, ψ, a/X}).
\]

**Lemma 9**

1. \(Ω(φ, ψ, a) ⇔ φ, if φ = p, τ or |\emptyset|, where a ≠ b;\)
2. \(Ω(〈a〉, ψ, a) ⇔ ψ_1[τ/\sqrt{}];\)
3. \(Ω([a], ψ, a) ⇔ ψ_2[τ/\sqrt{}];\)
4. \(Ω(∧_{i∈I} φ_i, ψ, a) ⇔ ∧_{i∈I} Ω(φ_i, ψ, a)\)
5. \(Ω(∨_{i∈I} φ_i, ψ, a) ⇔ ∨_{i∈I} Ω(φ_i, ψ, a)\)
6. \(Ω(φ_1; φ_2, ψ, a) ⇔ Ω(φ_1, ψ, a); Ω(φ_2, ψ, a)\)
7. \(Ω(σ^α.Xφ, ψ, a) ⇔ σ^α.XΩ(φ, ψ, a), where X is not free in ψ\)
8. \(Ω(σXφ, ψ, a) ⇔ σXΩ(φ, ψ, a), where X is not free in ψ\).

**Theorem 4 (Applicability).** If \(φ ∈ FLC\) and \(ψ ∈ \mathcal{P.F}\), then \(Ω(φ, ψ, a) ∈ FLC\). If \(φ, ψ ∈ \mathcal{P.F}\), then \(Ω(φ, ψ, a) ∈ \mathcal{P.F}\).
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:

\[ \phi \equiv \nu X. \left( \langle \text{leave}_\text{Paris} \rangle; \langle \text{fr}_\text{thr}_\text{Channel} \rangle; \langle \text{arrive}_\text{in}_\text{London} \rangle; \langle \text{work} \rangle; \right. \]
\[ \left. \langle \text{leave}_\text{London} \rangle; \langle \text{gb}_\text{thr}_\text{Channel} \rangle; \langle \text{arrive}_\text{in}_\text{Paris} \rangle \right) \]

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

\[ \psi_1 \equiv \mu X. ((\langle \text{contact}_\text{Customers} \rangle \lor \langle \text{meet}_\text{Customers} \rangle) \lor (\langle \text{finish}_\text{Work} \rangle) \lor X). \]

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:

\[ \psi_x \equiv [\langle \text{x_load} \rangle; \langle \text{x_departure} \rangle; \langle \text{cross}_\text{Channel} \rangle; \langle \text{x_arrival} \rangle; \langle \text{x_unload} \rangle]. \]

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

\[ \psi_2 \equiv [\langle \text{finish}_\text{loading} \rangle; \langle \text{engine}_\text{on} \rangle; \langle \text{bye}_\text{by} \rangle \]

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

\[ \psi_3 \equiv (\langle \text{sit}_\text{down} \rangle; \langle \text{stand}_\text{up} \rangle; \langle \text{keep}_\text{idle} \rangle) \lor X \lor (\langle \text{keep}_\text{idle} \rangle). \]

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

\[ \Omega(\phi, \Omega(\psi, \psi_2, x\text{_departure}), \psi_3, \text{cross}_\text{Channel}), x\text{_thr}_\text{Channel}, \psi_1, \text{work}), \]

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

Note that in the above example, if the parameter \( \psi_1 \) or \( \psi_2 \) of \( \Omega \) 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 \( Q \models \psi ; \sqrt{.} \) then \( P[ a \sim Q ] \models \Omega(\phi, \psi, a) \). 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 \( P[a \sim Q] \). Or, we refine \( \phi \) using \( \Omega(\phi, \psi, a) \) and obtain automatically a refined process \( P[a \sim Q] \) 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 \( (\text{Act}(P) \cup \text{Act}(\phi)) \cap (\text{Act}(Q) \cup \text{Act}(\psi)) = \emptyset \), because of the following considerations:

  1. **(i)** No deadlock will be introduced or destroyed because of action refinement.
  2. **(ii)** It is impossible that \( P \models a \sim Q \) will fail to satisfy \( \Omega(\phi, \psi, a) \) because \( \phi \) involves \( Q \). For instance, let \( a \equiv b, \phi \equiv [a]; \langle b \rangle \wedge (\langle c \rangle; \langle d \rangle); Q \equiv c; e \) and \( \psi \equiv [c]; \langle e \rangle \). It is obvious that \( P \models \phi \) and \( Q \models \psi ; \sqrt{.} \), but \( P[a \sim Q] \not\models \Omega(\phi, \psi, a) \).
(iii) Symmetrically, it is impossible that \(P[a \sim Q]\) will fail to satisfy \(\Omega(\phi, \psi, a)\) because \(\psi\) involves \(P\). For example, let \(P: a; b + b; a, \phi = [a]; (b), Q = c; e \text{ and } \psi = [c]; (e) \land [b]; (d)\). It is obvious that \(P \models \phi\) and \(Q \models \psi; \sqrt{\cdot}\), but \(P[a \sim Q] \not\models \Omega(\phi, \psi, a)\).

It is clear that the disjoint action condition can guarantee the above three requirements. Furthermore, it is possible that \(\psi\) 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 \(a; b + a; c \models (a); (b) \text{ and } a_1; a_2 \models (a_1)\), but 
\(a; b + a; c[a \sim a_1; a_2] \not\models (a_1)\) (b). In order to solve such a problem, we require that \(\psi\) describes complete executions of \(Q\), i.e., \(Q \models \psi; \sqrt{\cdot}\).

Finally, as argued in Theorem 3, it is required that \(\phi\) and \(\psi\) are PNF. Now, we can exactly state the Refinement Theorem as follows:

**Theorem 5 (Refinement Theorem).** Let \(\phi \in \mathcal{L}_{AFLC}\), \(\psi_1 \in \mathcal{E}\mathcal{F}\) and \(\psi_2 \in \mathcal{U}\mathcal{F}\), where \(\phi, \psi_1\) and \(\psi_2\) are PNF. If \((\text{Act}(P) \cup \text{Act}(\phi)) \cap (\text{Act}(\psi_1) \cup \text{Act}(\psi_2) \cup \text{Act}(Q)) = \emptyset\), \(Q \models (\psi_1 \land \psi_2); \sqrt{\cdot}\), and \(P \models \phi\) then \(P[a \sim Q] \models \Omega(\phi, \psi_1, \psi_2, a)\).

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:

\[
\text{Sys} \triangleq \text{fr\_Channel} \parallel [\text{fr\_thr\_Channel}] \text{Salesman} \parallel [\text{gb\_thr\_Channel}] \text{gb\_Channel},
\]

where \(x\_Channel \triangleq \text{rec} y \times \text{thr\_Channel}; y, \) and

\[
\text{Salesman} \triangleq \text{rec} x. \text{leave\_Paris;} \text{fr\_thr\_Channel}; \text{arrive\_in\_London}; \\
\text{work;} \text{leave\_London;} \text{gb\_thr\_Channel}; \text{arrive\_in\_Paris}; x.
\]

It is easy to check that \(\text{Sys} \models \phi\).

Then, we can refine “work” by Subsys1 which is defined by

\[
\text{Subsys1} \triangleq \text{rec} x. ((\text{contact\_Customers} + \text{meet\_Customers}); x + \text{finish\_Work}).
\]

Using model-checking or other methods, it can be verified that \(\text{Subsys1} \models \psi_1; \sqrt{\cdot}\). Then, “\(x\_thr\_Channel\)” may be implemented by \(\text{Subsys}_x \triangleq \text{x\_load} \parallel [\text{x\_load}] \text{Channel},\)

where \(\text{Channel} \triangleq \text{fr\_Platform} \parallel [\text{fr\_arrival, fr\_departure}] \text{Hovercraft} \\
\parallel [\text{gb\_arrival, gb\_departure}] \text{gb\_Platform},\)

where \(\text{Hovercraft} \triangleq \text{fr\_departure;} \text{cross\_Channel;} \text{gb\_arrival+} \\
\text{gb\_departure;} \text{cross\_Channel;} \text{fr\_arrival},\)

\[
x\_Platform \triangleq x\_load; x\_departure + x\_arrival; x\_unload.
\]

It is not hard to show that \(\text{Subsys}_x \models \psi_x; \sqrt{\cdot}\).

Furthermore, we may refine “\(x\_departure\)” by Subsys2 and “\(\text{cross\_Channel}\)” by Subsys3, where,

\[
\text{Subsys2} \triangleq \text{finish\_loading; engine\_on; bye\_bye,} \\
\text{Subsys3} \triangleq \text{sit\_down;} \\
\text{rec} x. ((\text{coffee} + \text{tea}) \parallel [\text{newspaper}]; x + \text{keep\_idle}); \text{stand\_up}.
\]

Certainly, it can be proved that \(\text{Subsys2} \models \psi_2; \sqrt{\cdot}\) and \(\text{Subsys3} \models \psi_3; \sqrt{\cdot}\).
Thus, the final system may be obtained as

\[
\text{Sys} \begin{bmatrix}
\text{work} \sim \text{Subsys}_1, \\
\text{x\_thr\_Channel} \sim \text{Subsys}_x \\
\text{x\_departure} \sim \text{Subsys}_2, \\
\text{cross\_Channel} \sim \text{Subsys}_3
\end{bmatrix},
\]

where \( x \in \{ \text{fr, gb} \} \).

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 \( \phi \) w.r.t. \(<\).

**Basic cases:**

- \( \phi = p, \tau, (\text{EF}) \), where \( b \neq a \)

  It is straightforward by Definition 10.

- \( \phi = [a] \)

  Thus, from Lemma 9, \( \Omega([a], \psi_1, \psi_2, a) \Leftrightarrow \psi_2[\tau/\sqrt{\ }]. \)

  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

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

  Furthermore, after multiple applications of Theorem 1, we get

  \[
P[a \sim Q] \sim \sum_{i=1}^m *Q; P[a \sim Q] + \sum_{i=1}^n *Q; P'_i[a \sim Q] + \sum_{i=1}^\ell b_i; Q[a \sim Q] + \sum_{i=1}^h *c_i; Q'_i[a \sim Q]. \tag{1}
\]

  As \((\text{Act}(\phi) \cup \text{Act}(P)) \cap (\text{Act}(\psi_1 \cup \psi_2) \cup \text{Act}(Q)) = \emptyset\), it follows that \( b_i \notin \text{Act}(\psi_2) \cup \text{Act}(Q) \) for \( 1 \leq i \leq \ell \) and \( c_i \notin \text{Act}(\psi_2) \cup \text{Act}(Q) \) for \( 1 \leq i \leq h \). Moreover, by the well-formedness condition (iii) listed in Section 2.1, it is obvious that \( \neg T(Q) \). Since \( Q \models \psi_2; \sqrt{\ } \) and \( \psi_2 \in \mathcal{UF} \), it follows that \( \psi_2[\tau/\sqrt{\ }/ \mathcal{UF}, \psi_2[\tau/\sqrt{\ }] \not\models ff \) and \( \sqrt{\ } \notin \text{BSub}(\psi_2[\tau/\sqrt{\ }]). \)

  By Lemma 6, without loss of generality,

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

  where \( \text{USF}(\psi_{i,j}) \) and \( \tau \notin \text{BSub}(\psi_{i,j}) \) for all \( 1 \leq i \leq m' \) and \( 1 \leq j \leq n_i \). So, there exists \( k < i' \leq m' \) such that \( Q \models \left( \bigvee_{j=1}^{n_{i'}} \psi_{i',j} \right); \sqrt{\ } \). Thus, by Corollary 1, it can be shown that for any \( 1 \leq i \leq \ell, b_i; Q[a \sim Q] \models \left( \bigvee_{j=1}^{n_i} \psi_{i,j} \right); \sqrt{\ } \) and for each \( 1 \leq i \leq h, *c_i; Q[a \sim Q] \models \left( \bigvee_{j=1}^{n_i} \psi_{i,j} \right); \sqrt{\ } \). It follows that for any \( 1 \leq i \leq \ell, b_i; Q[a \sim Q] \models \psi_2[\tau/\sqrt{\ }/ \mathcal{UF}, \psi_2[\tau/\sqrt{\ }] \not\models ff \) and \( \sqrt{\ } \notin \text{BSub}(\psi_2[\tau/\sqrt{\ }]). \)

  As \( P_i[a \sim Q] \models \tau \) by Definition 7 and \(*Q \models \psi_2[\tau/\sqrt{\ }/ \mathcal{UF}, \psi_2[\tau/\sqrt{\ }] \not\models ff \), from Lemma 1 and Theorem 2, applying Theorem 3, we get

  \[
  *Q; P_i[a \sim Q] \models \psi_2[\tau/\sqrt{\ }] \text{ for any } 1 \leq i \leq m. \text{ According to N, } *Q; P_i[a \sim Q] \models \psi_2[\tau/\sqrt{\ }] \text{ for any } 1 \leq i \leq m. \text{ Similarly, it can be shown that } *Q; P'_i[a \sim Q] \models \psi_2[\tau/\sqrt{\ }] \text{ for any } 1 \leq i \leq n. \text{ Therefore, we get } P[a \sim Q] \models \psi_2[\tau/\sqrt{\ }].
\]

**• \( \phi = \langle a \rangle \)**

  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 \((\text{Act}(P_1) \cup \text{Act}(\varphi)) \cap (\text{Act}(\chi_1 \land \chi_2) \cup \text{Act}(P_2)) = \emptyset, P_1 \models \varphi \) and \( P_2 \models (\chi_1 \land \chi_2) \); \( \sqrt{\ } \), then if \( \varphi' \prec \varphi, (\text{Act}(P_1) \cup \text{Act}(\varphi')) \cap (\text{Act}(\chi_1 \land \chi_2) \cup \text{Act}(P_2)) = \emptyset \), and \( P_1 \models \varphi' \), then \( P_1[a \sim P_2] \models \Omega(\varphi', \chi_1, \chi_2, a) \).

**Induction steps:**

- \( \phi = \bigwedge_{i \in I} \phi_i \)
  - Since \( P \models \phi \) for each \( i \in I \), it is easy to see that for each \( i \in I \), \((\text{Act}(P) \cup \text{Act}(\phi_i)) \cap (\text{Act}(\psi_1 \land \psi_2) \cup \text{Act}(\psi)) = \emptyset \), and therefore, from (IH), we get \( P[a \sim Q] \models \Omega(\phi_i, \psi_1, \psi_2, a) \). This entails \( P[a \sim Q] \models \Omega(\bigwedge_{i \in I} \phi_i, \psi_1, \psi_2, a) \) by Lemma 9.
- Case \( \phi = \bigvee_{i \in I} \phi_i \)
  - Similar to the above case.
- \( \phi = \varphi_1 \land \varphi_2 \)
  - This case can be proved by induction on \( \varphi_1 \) w.r.t. \( \prec \).
    - \( \varphi_1 = tt, ff, \sqrt{\ } \)
      - It is trivial.
    - \( \varphi_1 = \tau \)
      - Since \( P \models \varphi_1 \land \varphi_2 \) and \( \varphi_1 \prec \varphi_2 \) by N, it concludes that \( P \models \varphi_2 \).
      - Furthermore, it is obvious that \( \varphi_2 \prec \phi \) and \((\text{Act}(P) \cup \text{Act}(\varphi_2)) \cap (\text{Act}(\psi_1 \land \psi_2) \cup \text{Act}(\psi)) = \emptyset \). It follows \( P[a \sim Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a) \) by (IH).
    - \( \varphi_1 = [b] \)
      - Since \( P \models [b] \), \( \varphi_2 \), there exists \( P' \) such that \( P \Rightarrow P' \) and \( P' \models \varphi_2 \).
      - It is clear that \( \varphi_2 \prec \phi \) and \((\text{Act}(P') \cup \text{Act}(\psi_2)) \cap (\text{Act}(\psi_1 \land \psi_2) \cup \text{Act}(\psi)) = \emptyset \), so it follows that \( P'[a \sim Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a) \) by (IH).
      - This entails \( P'[a \sim Q] \models \Omega([b] \); \( \varphi_2 \), \( \psi_1 \), \( \psi_2 \), \( a) \) from Ref-1 and Lemma 9.
    - \( \varphi_1 = [a] \)
      - Thus, \( \Omega([a] \); \( \varphi_2 \), \( \psi_1 \), \( \psi_2 \), \( a) \) \( \models \varphi_2 \tau/\sqrt{\ } \)
      - \( \Omega(\varphi_2, \psi_1, \psi_2, a) \) from Lemma 9.

From (1) and \((\text{Act}(\phi) \cup \text{Act}(P)) \cap (\text{Act}(\psi_1 \land \psi_2) \cup \text{Act}(\psi)) = \emptyset \), it follows that \((b_1, c_1) \notin \text{Act}(\psi_1 \land \psi_2) \cup \text{Act}(\psi)\) for \( 1 \leq i \leq \ell \) and \( 1 \leq j \leq k \). Moreover, we get \( \varphi_2 \tau/\sqrt{\ } \models Q \cup \varphi_2 \tau/\sqrt{\ } \not\models \) and \( \not\models \text{BSub}(\varphi_2 \tau/\sqrt{\ }) \), as \( Q \neq \text{nil} \), \( Q \models \varphi_2 \) and \( \varphi_2 \in \mathcal{UF} \).

On the other hand, \( \psi_2 \tau/\sqrt{\ } \models \psi_2 \tau/\sqrt{\ } \models k \sum_{i=1}^{\ell} (\tau \land \bigwedge_{j=1}^{n_i} \psi_{i,j} \lor m \sum_{i=k+1}^{n_f} \bigwedge_{j=1}^{n_i} \psi_{i,j}) \) (3)

by Lemma 6, where \( \psi_{i,j} \in \mathcal{UF}, \text{USF}(\psi_{i,j}) \), and \( \tau \notin \text{BSub}(\psi_{i,j}) \) for \( 1 \leq i \leq m \) and \( 1 \leq j \leq n_i \). Since \( \tau \models \sqrt{\ } \), it follows that \( \exists k < h \leq m \) such that \( \lambda_{i=1}^{n_i} \psi_{i,j} \not\models \) \( \not\models \). Thus, by Lemma 7, it can be shown that for any \( 1 \leq k \leq \ell \),

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

and

\( \star c_k \); \( Q_k'[a \sim Q] \models (\bigwedge_{j=1}^{n_f} \psi_{i,j}) ; \Omega(\varphi_2, \psi_1, \psi_2, a) \)

for each \( 1 \leq k \leq h \).

Furthermore, applying DC, we get for any \( 1 \leq k \leq \ell \),

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

and

\( \star c_k \); \( Q_k'[a \sim Q] \models \psi_2 \tau/\sqrt{\ } ; \Omega(\varphi_2, \psi_1, \psi_2, a) \)

for each \( 1 \leq k \leq h \).
For $1 \leq k \leq m$, $P_k[a \sim Q] \models \Omega(\varphi_2, \psi_1, \psi_2, a)$ by (IH). Besides, as $Q \models (\psi_1 \land \psi_2) \land /\sqrt/,\,$ according to Theorem 3, Theorem 2 and (IH), $\ast Q; P'_k[a \sim Q] \models \psi_2[\tau /\sqrt/] \land \Omega(\varphi_2, \psi_1, \psi_2, a)$ for any $1 \leq k \leq n$. Therefore, we get

$$P[a \sim Q] \models \psi_2[\tau /\sqrt/] \land \Omega(\varphi_2, \psi_1, \psi_2, a)$$

from (1), Proposition 5 and Theorem 2.

- $\varphi_1 = (a)$

Similar to the above case.

- $\varphi_1 = \bigwedge_{i \in I} \varphi_{1i}$

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

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

by Lemma 9.

- $\varphi_1 = \bigvee_{i \in I} \varphi_{1i}$

Similar to the above case.

- $\varphi_1 = \varphi_{11} \land \varphi_{12}$

Since $(\varphi_{11}, \varphi_{12}, \varphi_2) < (\varphi_{11}, \varphi_{12}, \varphi_2)$, it follows

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

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

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

- $\varphi_1 = \sigma^X X. \varphi_{11}$ or $\sigma X. \varphi_{11}$

Similar to the subcase when $\psi_1 = v^\alpha X. \phi'$ in the proof for Theorem 3.

- $\phi = \sigma^X X. \phi_1$ or $\sigma X. \phi_1$

Similar to the subcase when $\psi_1 = v^\alpha X. \phi'$ in the proof for Theorem 3. □

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 $\phi[a \sim Q]$ from a high-level specification $\phi$ 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 $\phi, \phi_1, \phi_2 \in \text{FLC}, \phi[a \sim Q]$ is defined as:

$$\phi[a \sim Q] \deq \begin{cases} 
\phi & \text{if } \phi = p, X, \emptyset, \tau \text{ where } b \neq a \\
E(Q) & \text{if } \phi = (a) \\
U(Q) & \text{if } \phi = [a] \\
\phi_1[a \sim Q] \land \phi_2[a \sim Q] & \text{if } \phi = \phi_1 \land \phi_2 \\
\phi_1[a \sim Q] \lor \phi_2[a \sim Q] & \text{if } \phi = \phi_1 \lor \phi_2 \\
\phi_1[a \sim Q] ; \phi_2[a \sim Q] & \text{if } \phi = \phi_1 ; \phi_2 \\
\sigma X. \phi_1[a \sim Q] & \text{if } \phi = \sigma X. \phi_1 
\end{cases}$$

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

\[
E(Q) \equiv \begin{cases} 
\langle a' \rangle & \text{if } Q = a' \\
E(Q_1) \land E(Q_2) & \text{if } Q = Q_1 + Q_2 \\
E(Q_1); E(Q_2) & \text{if } Q = Q_1; Q_2 \\
\bigwedge_{Q_1 \to Q_1' \land b \notin A} (b) ; E(Q_1' \parallel_A Q_2) \\
\bigwedge_{Q_1 \to Q_2' \land b \notin A} (b) ; E(Q_1 \parallel_A Q_2') \\
\bigwedge_{Q_2 \to Q_2' \land b \notin A} (b) ; E(Q_1' \parallel_A Q_2') \\
\bigwedge_{Q_1 \to Q_1' \land Q_2 \to Q_2' \land b \notin A} (b) ; E(Q_1' \parallel_A Q_2') \\
E(Q_1)[a' \sim Q_2] & \text{if } Q = Q_1[a' \sim Q_2].
\end{cases}
\]

\[
U(Q) \equiv \begin{cases} 
\langle a' \rangle & \text{if } Q = a' \\
U(Q_1) \land U(Q_2) & \text{if } Q = Q_1 + Q_2 \\
U(Q_1); U(Q_2) & \text{if } Q = Q_1; Q_2 \\
\bigwedge_{Q_1 \to Q_1' \land b \notin A} [b] ; U(Q_1' \parallel_A Q_2) \\
\bigwedge_{Q_2 \to Q_2' \land b \notin A} [b] ; U(Q_1 \parallel_A Q_2') \\
\bigwedge_{Q_1 \to Q_2' \land Q_2 \to Q_2' \land b \notin A} [b] ; U(Q_1' \parallel_A Q_2') \\
U(Q_1)[a' \sim Q_2] & \text{if } Q = Q_1[a' \sim Q_2].
\end{cases}
\]

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

**Lemma 10.** For any \( Q \in \mathcal{F} \), \( E(Q) \land U(Q) \in \mathcal{P} \mathcal{F} \), \( Q \models E(Q) \land U(Q) \) and \( Q \models (E(Q) \land U(Q)) ; \sqrt{\cdot} \).

**Theorem 6.** For any \( Q \in \mathcal{F} \), \( \Omega(\phi, E(Q) \land U(Q), a) \iff \phi[a \sim Q] \).

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

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

\[
\phi[a \sim Q] \equiv \mu X.(\langle a \rangle ; [a] ; X) ; [a \sim a' ; b' + a' ; c'] \\
\iff \mu X.((\langle a' \rangle ; (b') \land (a') \land (c')) ; ([a'] ; [b'] \land [a'] ; [c']) ; X) \\
\iff \mu X.(E(Q) ; U(Q) ; X) \\
\iff \Omega(\phi, E(Q) \land U(Q), a).
\]

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 \( P \models \phi ; \sqrt{\cdot} \), where \( \phi \) is PNF, and \( Q \models \psi \), we can get a composite specification like \( \phi \{ \tau / \sqrt{\cdot} \} \psi \) 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 \( \mu \)-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 \( \mathcal{P} \) and \( \mu \)-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 [34], that is solving equation systems of the following form:

\[
C_1(X) \sim P_1, \ldots, C_n(X) \sim P_n
\]

where \( C_i \)'s 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.