The $N^\omega$ system as a development system for concurrent programs: $\delta N^\omega$

Dominique Méry

CRIN – CNRS and INRIA Lorraine, B.P. 239, 54506 Vandoeuvre France

Abstract

Méry, D., The $N^\omega$ system as a development system for concurrent programs: $\delta N^\omega$, Theoretical Computer Science 94 (1992) 311–334.

The development of programs is an activity that can be based on mathematical principles and on logical framework. This short paper contains rules allowing to derive concurrent programs with respect to a given programming language. These rules are built from more elementary rules. A short example is given. In some way, we identify concurrent programs with proofs. Our system $N^\omega$ is a reformulation of the framework proposed by Chandy and Misra (1988) but our reformulation is oriented towards a development framework in a formal way.

1. Introduction

The understanding of concurrency is a very interesting challenge but a very difficult one. The notions of concurrency or communication are real-life notions and, hence, need to be carefully and formally studied. A lot of frameworks have been built from sequential programming oriented ones or from logic. The temporal framework [21, 23, 13, 19, 2] has been exploited from concurrency and real-time aspects as a specification language and as a proof system framework. It allows to express invariance properties and eventuality properties but we can express crucial property as fairness using temporal logic [19, 10]. The relationship between regular $\omega$-language and temporal logic models is an interesting correspondence and a method of synthesis of concurrent programs has been developed by Wolper [23]. The temporal framework is powerful but it hides the adequate form of properties related to concurrency. In fact, we need a higher-order system to express transformations on proofs, and temporal logic refers to an implicit program contrary to dynamic logic [9], which has an explicit reference to programs. Nevertheless, dynamic logic is too close to syntax of programs and expressions of concurrency is not possible in the dynamic logic framework.
The semantical framework [20,16,17] characterizes the notions of concurrency in a formal and algebraic structure. This represents, in some sense, a generalization of automata theory to concurrency. People consider concurrency as an extension of sequential models. They reduce concurrency to sequentiality and nondeterminism. This procedure is not followed by everybody; others [1] propose a calculus where a concurrency operator is not reduced to nondeterminism and sequentiality. New directions are suggested by logicians as the linear logic [7] or a $\gamma$-calculus [3] which extends the $\lambda$-calculus to handle concurrent features. These new topics are under investigation and we may hope interesting results in the future. Chandy and Misra [4] have proposed a general framework to deal with the design of concurrent programs and mapping to given architectures. Their proposal has a really simple basis and the two main concepts are the nondeterminism and the fairness. A proof system allows to infer properties on UNITY programs. This framework needs to be better founded and we have completed and reformulated this system to obtain the $\\mathcal{M}\U$ system [14]. This first step is followed by a second step oriented towards design and transformation. This paper contains some steps in this direction for a very simple example and formal justifications of new rules. The paper is organized as follows. Section 2 contains preliminaries relative to definitions and a short introduction to UNITY. Section 3 gives different rules of $\\mathcal{M}\U$. In Section 4 we analyze the design process in $\\mathcal{M}\U$ and we prove some properties. Examples are given in Section 5. The paper ends by some concluding remarks and future directions.

2. Preliminaries

The programming language $\U$ is introduced and some aspects are specified.

**Definition 2.1.** Let $\imath$ be a set of variables, $\A$ a set of expressions over $\imath$ and $\B$ a set of conditions over $\A$ and $\imath$. A simple action $\alpha$ over $\imath$, $\A$, $\B$ is an expression like $v:=e_1 b_1 \cdots e_n b_n$ or $v=\prod_{i=1}^{n} e_i b_i$, where $v\in\imath$, $e_1,\ldots,e_n$ are in $\A$ and $b_1,\ldots,b_n$ are in $\B$. Moreover, $\imath^\imath[\alpha]=\{v\}$ and $\imath^\A[\alpha]=\bigcup_{i\in\{1,\ldots,n\}} \imath^\A[\imath b_i]$. A (general) action $\alpha$ over $\imath$, $\A$, $\B$ is an expression built according to the following rules:

1. $\alpha$ is a simple action.
2. If $\alpha_1,\ldots,\alpha_p$ are (general) actions and, for any $i,j$ in $\{1,\ldots,p\}$, $i\neq j$, $\imath^\imath[\alpha_i ] \cap \imath^\imath[\alpha_j ] = \emptyset$, then $\alpha_1 \cdots \alpha_p$ is a general action and $\imath^\imath[\alpha ] = \bigcup_{i\in\{1,\ldots,p\}} \imath^\imath[\alpha_i ]$.
3. If $\alpha_1,\ldots,\alpha_p$ are (general) actions so that $\imath^\imath[\alpha_1] = \cdots = \imath^\imath[\alpha_p]$, then $\prod_{i\in\{1,\ldots,p\}} \alpha_i \sim \gamma$, and $\imath^\imath[\prod_{i\in\{1,\ldots,p\}} \alpha_i \sim \gamma ] = \bigcup_{i\in\{1,\ldots,p\}} \imath^\imath[\alpha_i ]$.

Now, we define the language $\U$ as follows: a program $u$ of $\U$ is a pattern like $\varphi_u : [\alpha_1 \cdots \alpha_n ]$, where $\varphi_u$ specifies the condition to be satisfied by variables.
of actions as \( x_1, \ldots, x_n \). If \( u = [x_1 \cdots x_n] \) and \( v = [\beta_1 \cdots \beta_p] \), then \( u \cup v = [x_1 \cdots x_n \cup \beta_1 \cdots \cup \beta_p] \) and \( u \cup x = [x_1 \cdots x_n \cup x] \). An operational semantics can be defined for any \( u \) of \( \mathcal{V} \) but we will define it by a set of properties schemes as "\( p \xrightarrow{\alpha} q \)" where \( p \) and \( q \) are formulae and \( \alpha \) is an action of \( u \). We associate a weakest precondition operator with any action: \( \alpha \). Moreover, the execution of \( u \) is the activation of one and only one action chosen among \( \{x_1, \ldots, x_n\} \) under a fair choice. Hence, an execution of \( u \) is an infinite sequence of states connected by a transition relation sound with respect to \( \alpha \). A stable state may be considered as a terminal state: no action may modify the current values of the variables. A more complete and precise semantics is given in [15, 14]. In fact, we can prove the adequacy of our \( \mathcal{V} \) system with this semantics.

The language \( \mathcal{L} \) is a restricted version of the very general one proposed by Chandy and Misra [4]. We do not want to analyze the methodological interest of the declaration part that is very useful in the UNITY philosophy. An action \( \alpha \) of a program \( u \) or unity is always enabled but it may have no effect on program variables. The fairness hypothesis is better known as the justice hypothesis by [11] or the weak fairness by [19]. Francez [5] gives a very complete document on the fairness treatment. A short example can illustrate the previous short introduction and fix the different notions.

**Example 2.2** (gcd). \((x=x_0) \land (y=y_0) \land (x_0 \in \mathbb{N}) \land (y_0 \in \mathbb{N})\): \[x := x - y \sim x > y \sim Y := Y - X \sim X < Y\].

Under no fairness assumption, this program may diverge and may loop forever. A fair execution leads to a stable state satisfying \( x = y \).

3. The \( \mathcal{N} \mathcal{U} \) system

The \( \mathcal{N} \mathcal{U} \) system is a sound and semantically complete reformulation of a programming logic due to Chandy and Misra [4]. This reformulation is based on a semantical characterization of different properties for UNITY programs. \( \mathcal{N} \mathcal{U} \) is the kernel (level 0) of a more general method of concurrent programs developments. This kernel may be extended by specific syntactical operators relative to given properties. We introduce the \( \mathcal{N} \mathcal{U} \) system and refer the reader to the justification of it in [14].

\( \mathcal{N} \mathcal{U} \) uses several kinds of assertions relative to a unity, namely \( u \), that is assumed to be given:

- Formulae of \( \mathcal{L}[u] \) such as \( p, q, r, \ldots, p \lor q, p \land q, \sqrt{p}, p \rightarrow q, \ldots \)
- Transition formulae of \( \mathcal{T}[u] \) such as \( p \xrightarrow{\alpha} q, \forall \alpha : v. p \xrightarrow{\alpha} q, \exists_{\text{weak}} \alpha : v. p \xrightarrow{\alpha} q, \exists_{\text{strong}} \alpha : v. p \xrightarrow{\alpha} q, \) where \( v \) is a part of \( u \).
- Basic eventuality formulae of \( \mathcal{U}[u] \) like \( p \rightarrow q \).
- Eventuality formulae of \( \mathcal{E}[u] \) like \( p \rightarrow q \).
Invariance formulae of $\mathcal{L}[u]$ like $\text{always}(u, \varphi)(p)$.

$\mathcal{N} \&$ is used to infer properties of $u$ from axioms on transitions. The fairness rule is powerful and is the main rule of $\mathcal{N} \&$. $\mathcal{N} \&$ is made up of different parts like a first part for deriving transition formulae, a second part for deriving invariance formulae, a third part for deriving basic eventuality formulae and, finally, a fourth part for deriving eventuality formulae.

3.1. Transition formulae

A transition formula expresses a property of one program step. We have extended the class of Chandy and Misra's transition formulae [4] to express universal and existential properties that are more suitable for this purpose. Moreover, we have fixed a syntax for transition formulae and a wp-based axiomatics is proposed. The uniformization of transition formulae is achieved by $\varepsilon$, namely the empty action, that modelizes the implication.

Rule $\mathcal{N} \&$ 1 (from implication to transition).

$$\frac{p \rightarrow q}{p \varepsilon \rightarrow q}$$

Rule $\mathcal{N} \&$ 2 (axiom of precondition). Let $x$ be an action of $u$ and $(x \setminus q)$ be the precondition relative to $x$ and $q$.

$$\frac{(x \setminus q) \xrightarrow{[x]} q}{x \rightarrow q}$$

Rule $\mathcal{N} \&$ 3 (extension of context). Let $v, w$ be two parts of $u$ and $x \in v$.

$$\frac{p \xrightarrow{[x]} q}{p \xrightarrow{[x] \circ w} q}$$

Rule $\mathcal{N} \&$ 4 (strengthening–weakening).

$$\frac{r \xrightarrow{e} p, p \xrightarrow{[x]} q, q \xrightarrow{e} s}{r \xrightarrow{[x]} s}$$

Rule $\mathcal{N} \&$ 5 (introduction of the universal quantification).

$$\frac{r \xrightarrow{[x]} q}{\forall x: [x], p \xrightarrow{[x]} q}$$
Rule $\mathcal{N}$ $\mathcal{H}$ 6 (universal quantification).
\[
\forall x : u . p \xrightarrow{x : u} q, p \overset{\beta : [x]}{\rightarrow} q \\
\forall x : u \quad p \overset{x : u}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 7 (weakening of universal quantification).
\[
\forall x : u . p \xrightarrow{z : u} q \\
\forall y : u \quad p \overset{y : u}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 8 (weak existential quantification).
\[
p \xrightarrow{x : [x]} q \\
\exists_{\text{weak}} \beta : [x] . p \overset{\beta : [x]}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 9 (weakening of weak existential quantification).
\[
\exists_{\text{weak}} \beta : u . p \overset{\beta : u}{\rightarrow} q \\
\exists_{\text{weak}} y : u \quad p \overset{y : u}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 10 (strong existential quantification).
\[
p \xrightarrow{x : [x]} q \\
\exists_{\text{strong}} \beta : [x] . p \overset{\beta : [x]}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 11 (weakening of strong existential quantification).
\[
p_1 \xrightarrow{x : [x]} q, \exists_{\text{strong}} \beta : u . p_2 \overset{\beta : u}{\rightarrow} q \\
\exists_{\text{strong}} y : u \quad (p_1 \lor p_2) \overset{y : u}{\rightarrow} q
\]

Rule $\mathcal{N}$ $\mathcal{H}$ 12 (parallel execution).
\[
p \xrightarrow{x : u} q, p \overset{y : u}{\rightarrow} p, q \overset{\beta : u}{\rightarrow} q', p' \overset{\beta : u}{\rightarrow} p', q' \overset{\beta : u}{\rightarrow} q' \\
p \land p' \overset{x : [u]}{\rightarrow} q \land q'
\]

3.2. Invariance formulae

A lot of design methods are based on invariance properties as Gribomont’s method [8]. The method is a step by step refinement of program/invariant. These properties are the merge of the implication and of the general transition (universal) transition formulae.
Rule $\mathcal{N}$\&U 13 (invariance).
\[ \varphi \rightarrow p, \forall x: u.p \xrightarrow{x::u} p \]
\[ \text{always}(u, \varphi)(p) \]

3.3. Basic eventuality formulae

Among eventuality properties, basic eventuality properties express that a given formula holds until another will hold. In temporal framework [12], this property is expressed by the \textit{until} operator.

Rule $\mathcal{N}$\&U 14 (from implication to basic eventuality).
\[ p \rightarrow q \]
\[ p \xrightarrow{u} q \]

Rule $\mathcal{N}$\&U 15 (fairness).
\[ \forall x: u.p \land \neg q \xrightarrow{x::u} p \lor q, \exists x: u.p \land \neg q \xrightarrow{x::u} q \]
\[ p \xrightarrow{u} q \]

The existential operator will denote either the weak version, or the strong one. But this leads to two different kinds of fairness. The weak one corresponds to UNITY's fairness.

Rule $\mathcal{N}$\&U 16 (general induction).
\[ p_{u} \xrightarrow{u} (\bigcup_{\beta < x} p_{\beta}), 0 < x \leq z_{0} \]
\[ (\bigcup_{0 < x \leq z_{0}} p_{x})^{u} p_{0} \]

Rule $\mathcal{N}$\&U 17 (weakening of eventuality).
\[ p \xrightarrow{u} q, \text{always}(u, \varphi)(i), r \land i \xrightarrow{c::u} p, p \xrightarrow{c::u} r \land i \]
\[ r \xrightarrow{u} q \]

What does the fairness rule $\mathcal{N}$\&U (15) mean? First, it expresses that any action of \( u \) may lead from \( p \land \neg q \) to either \( p \) or \( q \) but the execution of some atomic action \( x \) of \( u \) in some state leads to \( q \). It means, for instance, that some actions are critical to reach \( q \). Under these two assumptions, we infer that \( q \) will eventually happen, from \( p \) continuously until \( q \) holds. The conclusion of the rule is correct if we restrict the set of traces to fair traces. A simple example can illustrate this rule in the weak version.

Example 3.1 (Chandy and Misra [4]). \( u \) is the unity. \( \varphi_{u} : [x := 0 \sim x > 0 \)
The NQ system as a development system: $\mathcal{N}_q$

The initial condition is $\varphi_0 \overset{\text{def}}{=} (x \in \text{integer}).$ We can infer the following properties:

1. $((x > 0) \land (x = 0)[0/x]) \lor ((x \leq 0) \land (x = 0)))$ \quad (Rule $\mathcal{N}_q$ 2).

2. $((x > 0) \land (x < 0)[0/x]) \lor ((x < 0) \land (x < 0))$ \quad (first-order theorem).

3. $x > 0 \rightarrow (((x > 0) \land (x = 0)[0/x]) \lor ((x \leq 0) \land (x = 0)))$ \quad (Rule $\mathcal{N}_q$ 3, (1), (3)).

4. $x < 0 \rightarrow ((x > 0) \land (x < 0)[0/x]) \lor ((x < 0) \land (x < 0))$ \quad (first-order theorem).

5. $x < 0 \rightarrow ((x > 0) \land (x < 0)[0/x]) \lor ((x < 0) \land (x < 0))$ \quad (first-order theorem).

6. $x < 0 \rightarrow (x - 0) \lor (x \neq 0)$ \quad (first-order theorem).

7. $x < 0 \rightarrow (x - 0) \lor (x \neq 0)$ \quad (first-order theorem).

8. $x < 0 \rightarrow ((x > 0) \land (x < 0)[0/x]) \lor ((x < 0) \land (x < 0))$ \quad (Rule $\mathcal{N}_q$ 4, (6), (7)).

$\mathcal{N}_q$-Theorem 1. $x > 0 \rightarrow x = 0.$

$\mathcal{N}_q$-Theorem 2. $x < 0 \rightarrow x = 0.$

The other atomic action of $[x := 0 \sim x > 0] x := 0 \sim x < 0]$ has almost identical properties.

$\mathcal{N}_q$-Theorem 3. $x < 0 \rightarrow x = 0.$

$\mathcal{N}_q$-Theorem 4. $x > 0 \rightarrow x = 0.$

The proofs in $\mathcal{N}_q$ is a substitution of $< \text{ by } >$ and $> \text{ by } <$ in the previous proof. Quantifications are used to build termination properties. We are going to use rules that will be introduced later. We prove the following theorem in $\mathcal{N}_q$.

$\mathcal{N}_q$-Theorem 5. $x \neq 0 \rightarrow (x = 0).$

Proof. (1) $x < 0 \rightarrow (x = 0)$ \quad (by $\mathcal{N}_q$-Theorem 2).

(2) $x > 0 \rightarrow x = 0$ \quad (by $\mathcal{N}_q$-Theorem 1).

(3) $x > 0 \rightarrow (x = 0)$ \quad (by Rule $\mathcal{N}_q$ 4, (2)).
In their system, Chandy and Misra proved the last statement without using an intermediate basic eventuality property. Yet, this property is in the kernel of the proof method. The two actions may modify the value of \( x \) according to the sign of \( x \) and the effect of the given action may not be modified. For instance, if the initial value of \( x \) is 5 and if the action \( \text{x:=0-x >0} \) is always executed, then the value of \( x \) always remains equal to 5. But, this trace is not fair and, necessarily, the second action is eventually executed. In another section, we will understand and illustrate the power of this rule that is really different with respect to Chandy and Misra’s one.

3.4. Eventuality formulae

This part deals with properties expressing that something eventually holds in the future of the program. Two rules are needed to introduce the \( \leftrightarrow \) operator. Other auxiliary rules are given because they are helpful in the proofs but they are provable in the minimal system.

**Rule \( \mathcal{N} \mathcal{W} 18 \) (from basic eventuality to eventuality).**

\[
\frac{p \leftrightarrow q \quad u}{p \leftrightarrow q}
\]
Rule $\mathcal{N}^\mathcal{W} 19$ (strengthening).

\[
p \vdash^u q, \text{always}(u, \varphi)(i), r \land i \vdash^u p \quad \frac{r \vdash^u q}{p \vdash^u q}
\]

Rule $\mathcal{N}^\mathcal{W} 20$ (from implication to eventuality).

\[
p \vdash^{\cdot \cdot \cdot \cdot} q
\]

\[
\frac{p \vdash^{\cdot \cdot \cdot \cdot} q}{p \vdash^{\cdot \cdot \cdot \cdot} q}
\]

Rule $\mathcal{N}^\mathcal{W} 21$ (derived induction).

\[
p_2 \vdash^u \bigcup_{\beta < \alpha} p_\beta, \quad 0 < \alpha \leq \alpha_0
\]

\[
\frac{(\bigcup_{0 < \alpha \leq \alpha_0} p_\alpha) \vdash^u p_0}{(\bigcup_{0 < \alpha \leq \alpha_0} p_\alpha) \vdash^u p_0}
\]

A lot of very useful and practical rules can be derived from the system above. The two following rules can be derived from $\mathcal{N}^\mathcal{W}$ and are not needed for the semantical completeness. These rules are useful and can be added after formal proofs. But the notion of proof must be defined. A proof in $\mathcal{N}^\mathcal{W}$ is a transfinite sequence of assertions belonging to $\mathcal{L}[u] \cup \mathcal{L}[u] \cup \mathcal{L}[u] \cup \mathcal{L}[u] \cup \mathcal{L}[u] \cup \mathcal{L}[u]$ denoted as $\mathcal{L}[u]$ and satisfying the following property: an element of this sequence is either an instance of axiom, or derived from previous elements of the sequences according to one rule of $\mathcal{N}^\mathcal{W}$.

**Notation.** A proof of $\Phi$ is denoted as $\Pi[\mathcal{N}^\mathcal{W}] \Phi$. Let $\Phi$ be any assertion for $u$. If $\Phi$ has a proof, it has a transfinite number of proofs because we can repeat any element for a transfinite number of times. We call size of a proof, the least length of proofs for any given assertion and we denote it by $C[\Phi]$.

The minimal proof of $\Phi$ is denoted as $\Pi[\Phi]$ and is written as: $\Phi_0, \ldots, \Phi_{|\Phi|}$. If $\Phi$ is not a theorem, the size is undefined. In our system, properties as $\text{"p} \rightarrow \text{q} \text{"}$ are proved outside of this system. So, in our framework, the size of a theorem as $\text{"p} \rightarrow \text{q} \text{"}$ is 0. A very simple property can be easily proved.

**Lemma 3.2.** If $\Pi[\Phi]$ is the proof of $\Phi$ and $\Phi'$ is in $\Pi[\Phi]$, then $C[\Phi'] < C[\Phi]$.

It means that we can use an induction principle to derive other rules. Moreover, inference rules can be read as functions transforming a set of proofs into another proof. The transfinite size of proofs is due to the very general rule of induction $\mathcal{N}^\mathcal{W}$ 16 but, in the pet examples, we use a finistic version of it. We have to describe the proof theory in a further work in preparation. Now, we can prove interesting properties or derived rules.
3.5. Auxiliary rules

**Rule N 22** (introduction of $\lor$ in transition formulae).
\[
\frac{p \rightsquigarrow r, q \rightsquigarrow r}{(p \lor q) \rightsquigarrow r}.
\]

**Rule N 23.**
\[
\frac{p \rightsquigarrow q}{(q \lor r)}.
\]

A more general form for this rule is better and introduces a general disjunction operator. This new rule will be stated in Section 4.

**Rule N 24.**
\[
\frac{p \rightsquigarrow q, q \rightsquigarrow r}{p \rightsquigarrow q \land r}.
\]

**Rule N 25.**
\[
\frac{\forall x:u. p \rightsquigarrow q, q \rightarrow r}{\forall x:u. p \rightarrow r}.
\]

**Rule N 26.**
\[
\frac{\exists_{\text{weak}} x:u. p \rightarrow r, \exists_{\text{weak}} x:u. q \rightarrow r}{\exists_{\text{weak}} x:u. (p \lor q) \rightarrow r}.
\]

**Rule N 27.**
\[
\frac{\exists x:u. p \rightarrow q, q \rightarrow r}{\exists x:u. p \rightarrow r}.
\]

**Rule N 28.**
\[
\frac{p \rightarrow q, \forall x:u. q \rightarrow r, r \rightarrow s}{\forall x:u. p \rightarrow s}.
\]
The $N\mathcal{W}$ system as a development system: $N\mathcal{W}$

**Rule $N\mathcal{W}$ 29.**

\[
\begin{align*}
p & \rightarrow q, \exists_{\text{weak}} x: u . q \\ r & \rightarrow s \rightarrow \exists_{\text{weak}} x: u . p \\
\end{align*}
\]

**Rule $N\mathcal{W}$ 30.** Let \( p_1, p_2, q \) be three formulae of $\mathcal{L}[u]$.

\[
\begin{align*}
\forall x: u . p_1 & \rightarrow q, \forall x: u . p_2 & \rightarrow q \\
\forall x: u . (p_1 \lor p_2) & \rightarrow q
\end{align*}
\]

**Rule $N\mathcal{W}$ 31.** Let \( p, q, r \) be three formulae of $\mathcal{L}[u]$.

\[
\begin{align*}
p & \rightarrow q, q \rightarrow r \\
\rightarrow & \rightarrow r
\end{align*}
\]

**Rule $N\mathcal{W}$ 32.** Let \( p, q, r \) be three formulae of $\mathcal{L}[u]$.

\[
\begin{align*}
p & \rightarrow q, q \rightarrow r \\
\rightarrow & \rightarrow r
\end{align*}
\]

A lot of rules can be derived from the kernel and these rules contain elements to derive the design of programs. We must mention that our example is only an illustrative one, but is good enough to illustrate the design method hidden behind $N\mathcal{W}$.

4. Development of concurrent programs in $N\mathcal{W}$

A development is a finite sequence of actions that operate on specifications. A specification is a collection of properties expressed by transition formulae, invariance formulae, basic eventuality formulae or eventuality formulae. A specification describes the current state of the product. It is always proved according to the $N\mathcal{W}$ rules. A specification has an operational basis, namely, a set of transition formulae that can be interpreted as an operational semantics. Generally, a specification is a static expression of some given computation and no dynamic aspect is specified. But, avoiding any discussion on the notion of specification and the notion of program, we use an operational style of specification based on assertions, as “\( p \rightarrow q \)”. The careful reader will notice that programs are flat objects and rules for hierarchy are to be built. The syntax of concurrent programs used by Chandy and Misra [4] is really simple but poor. It means that they use the operator $|$ to denote the concurrency of two actions, as “\( x \parallel \beta \)” means that “\( x \) and \( \beta \) start at the same instant and terminate at the same instant”. Our main motivation is to structure the flat programs to obtain expressions
of programs, such as \( u_1 \parallel u_2 \), but the meaning has to be more asynchronous. In some way, we want to build new programs from basic ones by combining them in a suitable way. This aspect may be seen as an extraction method from proofs [6]. Yet, we have to propose rules able to process as previously stated.

In the previous section, we have introduced the notion of proof of a given property in \( \mathcal{N} \). Proofs of concurrent programs are very complex tasks and require special proofs for interference freedom [18, 22]. The level of interference depends on the atomicity grain. The finer the grain, the more difficult is the proof. In our framework, we restrict our work to one level and we choose a given grain of atomicity. We are going to show how concurrent programs can appear.

If \( \Phi \) is any property expressible in our language, \( \vdash^\circ \Phi \) expresses the derivability of \( \Phi \) in \( \mathcal{N} \) without any assumption other than implicative properties expressed as \( \psi \rightarrow \phi \). Now, a \( \Gamma \)-theorem in \( \mathcal{N} \), \( \Phi \), denoted as \( \Gamma \vdash^\circ \Phi \), is derived using axioms and rules of \( \mathcal{N} \) and assertions of \( \Gamma \). Using this notion, a specification for \( u \) is a set of transition formulae, namely, \( \text{Spec}[u] \).

A specification is called an abstract operational specification because it is close to operational semantics but is abstract.

**Definition 4.1.** Let \( u \) be a unity. An abstract operational specification for \( u \), \( \text{Spec}[u] \), is a set of transition formulae for \( u \): \( \text{Spec}[u] = \{ p \xrightarrow{2;\mathcal{U}} q | p, q \in \mathcal{L}[u] \} \). An abstract operational specification is complete if, for any \( tf \) transition formulae of \( u \), \( \text{Spec}[u] \vdash^\circ tf \) if and only if \( \vdash^\circ tf \).

The \( \delta \mathcal{N} \) system is obtained from \( \mathcal{N} \) by deleting the Rule \( \mathcal{N} \) 2 and by adding the following rules.

**Rule \( \delta \mathcal{N} \) 1 (introduction of \( \lor \) in transition formulae).**

\[
\frac{p_i \xrightarrow{2;\mathcal{U}} r, i \in I}{(\lor_{i \in I} p_i) \xrightarrow{2;\mathcal{U}} r}.
\]

**Rule \( \delta \mathcal{N} \) 2 (introduction of \( \land \) in transition formulae).**

\[
\frac{p_i \xrightarrow{2;\mathcal{U}} r_i, i \in I}{(\land_{i \in I} p_i) \xrightarrow{2;\mathcal{U}} \land_{i \in I} r_i}.
\]

**Rule \( \delta \mathcal{N} \) 3.**

\[
p \land g_i \xrightarrow{a_i;[x_i]} r, i \in I, (p \land \sim \land_{i \in I} g_i) \xrightarrow{e:[\text{c}]} \text{c} \rightarrow t, \text{c} \rightarrow \prod_{i \in I} g_i \xrightarrow{\prod_{i \in I} g_i \rightarrow \text{c} \rightarrow \text{c}} r \rightarrow \prod_{i \in I} g_i \rightarrow \text{c} \rightarrow \text{c} \rightarrow r.
\]
The completeness of an abstract operational specification means that the set of transition formulae is as expressible as the \( wp \) specification. But, we will notice that our rules are instances of rules and have to be instantiated. The new system includes implicit properties of \( wp \) assertions.

When one writes \( "a : u" \), we define a hierarchy relation between \( a \) and \( u \). The basic atomic actions of a unity are actions as \( "x := e_1 \sim b_1 \ldots e_n \sim b_n" \), where \( x \) is a variable, \( e_i \) is an expression and \( b_i \) is a condition. Two actions \( a \) and \( \beta \) may be composed using the \( \parallel \) if they do not modify identical variables. The meaning of \( "a \parallel \beta" \) is that the two actions begin together and end together.

The analysis of proofs in \( \mathcal{N} \mathcal{W} \) (or in \( \delta \mathcal{N} \mathcal{W} \)) is guided by the notion of abstract operational specification. A proof of \( \phi \) in \( \mathcal{N} \mathcal{W} \) is denoted as \( \mathcal{N} \mathcal{W} \models \phi \) and a proof of \( \phi \) in \( \delta \mathcal{N} \mathcal{W} \) for \( \text{Spec} \) is denoted as \( \text{Spec} \models \phi \). The relation between the two kinds of proofs can be stated as follows.

**Theorem 4.2.** Let \( u \) be a unity and \( \phi \) a formula of \( \mathcal{G} \mathcal{L} [u] \). Let \( \text{Spec} [u] \) be a complete abstract operational specification for \( u \). \( \mathcal{N} \mathcal{W} \models \phi \) if and only if \( \text{Spec} [u] \models \phi \).

**Proof.** Let \( \text{Spec} [u] \) be a complete abstract operational specification for \( u \). A proof in \( \mathcal{N} \mathcal{W} \) uses transition formulae and implicative formulae.

Any transition formula is derived in \( \mathcal{N} \mathcal{W} \) from the Rule \( \mathcal{N} \mathcal{W}2 \), and the Rule \( \mathcal{N} \mathcal{W}2 \) can be derived from assumptions of the specification using \( \delta \mathcal{N} \mathcal{W} \). Hence, any proof in \( \mathcal{N} \mathcal{W} \) can be transformed into a proof in \( \delta \mathcal{N} \mathcal{W} \) by substituting any \( wp \) formula by the related proof.

Any proof in \( \delta \mathcal{N} \mathcal{W} \) can be transformed into a proof in \( \mathcal{N} \mathcal{W} \) by substituting any pattern of \( wp \) proof by the \( wp \) expression. \( \square \)

This theorem means that the two fundamental properties are the implicative ones and the transition ones. The proofs of implicative formulae are built by the user. Now,
we explore the parallel combinations of unities according to properties of proofs. A very strong hypothesis is stated to ensure a sound derivation. The composition of proofs and of programs can be expressed under proofs independence.

**Definition 4.3.** Let \( u_1 \) and \( u_2 \) be two different unities with their respective specifications, namely, \( \text{Spec}[u_1] \) and \( \text{Spec}[u_2] \). We say that \( u_1 \) and \( u_2 \) are independent with respect to their specifications if, for any \( i \) in \( \{1, 2\} \), for any \( j \) in \( \{1, 2\} \), \( i \neq j \) for any transition formula \( p \xrightarrow{\alpha_i; u_i} q \) satisfying

\[
\text{Spec}[u_i] \quad \vdash \quad p \xrightarrow{\alpha_i; u_i} q,
\]

\[
\text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \vdash \quad \forall \alpha : u_j. p \xrightarrow{\alpha_i; u_j} p
\]

and

\[
\text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \vdash \quad \forall \alpha : u_j. q \xrightarrow{\alpha_i; u_j} q.
\]

The independence of \( u_1 \) and \( u_2 \) with respect to \( \text{Spec}[u_1] \) and \( \text{Spec}[u_2] \) is denoted as \((u_1, \text{Spec}[u_1]) \perp (u_2, \text{Spec}[u_2])\). A sound parallel composition of \( u_1 \) and \( u_2 \) can be defined and is denoted as \( u_1 \parallel u_2 \).

**Theorem 4.4** (Independent composition). Let us assume that

1. \( \text{Spec}[u_1] \) and \( \text{Spec}[u_2] \) are two complete abstract operational specifications,
2. \((u_1, \text{Spec}[u_1]) \perp (u_2, \text{Spec}[u_2])\),
3. \( \text{Spec}[u_1] \vdash p_1 \xrightarrow{u_1} q_1 \),
4. \( \text{Spec}[u_2] \vdash p_2 \xrightarrow{u_2} q_2 \),
5. \( \text{Spec}[u_1] \vdash \text{always}(u_1, q_1)(q_1) \),
6. \( \text{Spec}[u_2] \vdash \text{always}(u_2, q_2)(q_2) \).

Then \( \text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash p_1 \land p_2 \xrightarrow{u_1 \parallel u_2} (q_1 \land q_2) \).

The conditions of this theorem are very strong but the composition of programs in a concurrent style must be carefully used.
Proof. The proof is divided into four cases:

(1) \( p_1 \rightarrow q_1 \) and \( p_2 \rightarrow q_2 \) are derived using the Rule \( N'N' \) 15.

(2) \( p_1 \rightarrow q_1 \) (\( p_2 \rightarrow q_2 \)) is derived using the Rule \( N'N' \) 15 and \( p_2 \rightarrow q_2 \) (\( p_1 \rightarrow q_1 \)) is derived using the Rule \( N'N' \) 16.

(3) \( p_1 \rightarrow q_1 \) and \( p_2 \rightarrow q_2 \) are derived using the Rule \( N'N' \) 16.

(4) \( p_1 \rightarrow q_1 \) (\( p_2 \rightarrow q_2 \)) is derived using the Rule \( N'N' \) 17 and \( p_2 \rightarrow q_2 \) (\( p_1 \rightarrow q_1 \)) is derived using either the Rule \( N'N' \) 15, or the Rule \( N'N' \) 16, or the Rule \( N'N' \) 17.

We use the notation “\( \Gamma \vdash \phi \)” meaning that there exists a proof of \( \phi \) in \( \delta N'N' \) from \( \Gamma \). We abusively use the set-theoretic notation \( \subseteq \).

Case 1: For any \( i \) in \( \{1, 2\} \),

\[
\text{Spec}[u_i] \vdash \forall x: u_i. (p_i \land \sim q_i) \rightarrow (p_i \lor q_i)
\]

and

\[
\text{Spec}[u_i] \vdash \exists x: u_i. (p_i \land \sim q_i) \rightarrow q_i.
\]

Under case assumptions and theorem assumptions, we can state that for any \( i \) in \( \{1, 2\} \), for any \( x \) in \( u_i \),

\[
\text{Spec}[u_i] \vdash (p_i \land \sim q_i) \rightarrow (p_i \lor q_i),
\]

and there exists a sequence \( (r^i_1)_{x \in v_1} \) and \( v_i \subseteq u_i \) so that

(1) \( \bigvee_{x \in v_1} r^i_1 = p_i \land \sim q_i \).

(2) \( \text{Spec}[u_i] \vdash r^i_1 \rightarrow q_i \), for any \( x \) in \( v_i \).

Since \( u_1 \) and \( u_2 \) are independent, we can apply the composition Rule \( N'N' \) 12.

\[
\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash p_1 \land p_2 \land \sim q_1 \land \sim q_2 \rightarrow (p_1 \lor q_1) \land (p_2 \lor q_2)
\]

(by a simple reasoning on implication).
Hence, by Rule $N \forall$ 4,

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash (p_1 \land p_2 \land \neg((p_1 \land q_2) \lor (q_1 \land p_2)))$$

$$\lor (q_1 \land p_2)) \xrightarrow{\alpha:u_1\parallel u_2} (p_1 \land p_2) \lor ((p_1 \land q_2) \lor (q_1 \land p_2)) \lor (q_1 \land p_2)),$$

for any $\alpha$ in $u_1 \parallel u_2$.

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash \forall \alpha:u_1\parallel u_2, (p_1 \land p_2 \land \neg((p_1 \land q_2)$$

$$\lor (q_1 \land p_2)) \lor (p_1 \land p_2) \lor ((p_1 \land q_2) \lor (q_1 \land p_2))$$

$$\lor (q_1 \land p_2))$$

(by successive applications of Rules $N \forall$ 5 and $N \forall$ 6).

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash (r_1^1 \land r_2^2) \xrightarrow{\alpha:1\parallel u_2} (q_1 \land q_2)$$

and

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash (r_1^1 \land r_2^2) \rightarrow (q_1 \land q_2)$$

(by the Rule $N \forall$ 12).

Now, we can infer the existential transition property:

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash \exists \gamma:(p_1 \land p_2) \rightarrow ((q_1 \land q_2) \lor (p_1 \land q_2))$$

(by the Rules $N \forall$ 10 and $N \forall$ 11).

Using the Rule $N \forall$ 15,

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash (p_1 \land p_2) \rightarrow ((q_1 \land q_2)$$

$$\lor (p_1 \land q_2) \lor (q_1 \land p_2))$$

Since $q_2$ is stable for $u_2$, and $q_1$ is stable for $u_1$, we have the properties:

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash q_2 \xrightarrow{\alpha:u_2} q_2$$

and

$$\text{Spec}[u_1] \cup \text{Spec}[u_2] \vdash q_1 \xrightarrow{\alpha:u_1} q_1.$$
According to the previous proof, we derive

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (p_1 \land q_2) \quad \overset{u_1 \cup u_2}{\iff} \quad (q_1 \land q_2) \]

and

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (q_1 \land p_2) \quad \overset{u_1 \cup u_2}{\iff} \quad (q_1 \land q_2). \]

Using the Rule 16, we can state that

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad ((p_1 \land q_2) \lor (q_1 \land p_2)) \quad \overset{u_1 \cup u_2}{\iff} \quad (q_1 \land q_2). \]

Finally,

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (p_1 \land p_2) \quad \overset{u_1 \cup u_2}{\iff} \quad ((p_1 \land q_2) \lor (q_1 \land p_2)) \]

and

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad ((p_1 \land q_2) \lor (q_1 \land p_2)) \quad \overset{u_1 \cup u_2}{\iff} \quad (q_1 \land q_2). \]

The transitivity of \( \iff \) leads to

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (p_1 \land p_2) \quad \overset{u_1 \cup u_2}{\iff} \quad (q_1 \land q_2). \]

Case 2:

\[ \text{Spec}[u_1] \quad \overset{\delta, \gamma}{\vdash} \quad \forall x : u_1 . \left( p_1 \land \sim q_1 \right) \quad \overset{u_1 \cup u_2}{\rightarrow} \quad (p_1 \lor q_1) \]

and

\[ \text{Spec}[u_1] \quad \overset{\delta, \gamma}{\vdash} \quad \exists x : u_1 . \left( p_1 \land \sim q_1 \right) \quad \overset{u_1 \cup u_2}{\rightarrow} \quad q_1 \]

and there exists a sequence \( (r_x)_{x \in \text{Ord}} \) satisfying

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad \left( \bigvee_{x \in \text{Ord}} r_x \right) \quad \overset{u_2}{\rightarrow} \quad r_0 \]

and

\[ p_2 = \bigvee_{x \in \text{Ord}} r_x, \quad r_0 = q_2. \]

Under the induction hypothesis, we derive the following statements:

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (p_1 \land r_\alpha) \quad \overset{u_1 \cup u_2}{\iff} \quad \left( q_1 \land \left( \bigvee_{\beta < x} r_\beta \right) \right) \]

and

\[ \text{Spec}[u_1] \cup \text{Spec}[u_2] \quad \overset{\delta, \gamma}{\vdash} \quad (q_1 \land r_\alpha) \quad \overset{u_1 \cup u_2}{\iff} \quad \left( q_1 \land \left( \bigvee_{\beta < x} r_\beta \right) \right) \]
Case 3: There exists a sequence \((r_\alpha)_{\alpha \in \text{Ord}}\) satisfying

\[
\text{Spec}\[u_1]\cup \text{Spec}\[u_2] \vdash (\bigvee_{\alpha \in \text{Ord}} r_\alpha) \rightarrow r_0
\]

and

\[
p_1 \overset{\text{def}}{=} \bigvee_{\alpha \in \text{Ord}} r_\alpha, \quad r_0 = q_1,
\]

and a sequence \((s_\alpha)_{\alpha \in \text{Ord}}\) satisfying

\[
\text{Spec}\[u_1]\cup \text{Spec}\[u_2] \vdash (\bigvee_{\alpha \in \text{Ord}} s_\alpha) \rightarrow s_0
\]

and

\[
p_2 \overset{\text{def}}{=} \bigvee_{\alpha \in \text{Ord}} s_\alpha, \quad s_0 = q_2.
\]

Since the formulae \(s_\alpha\) and \(r_\beta\) satisfy the conditions of the theorem, we can apply it:

\[
\text{Spec}\[u_1]\cup \text{Spec}\[u_2] \vdash (r_\alpha \land s_\beta) \overset{u_1||u_2}{\rightarrow} \left(\bigvee_{(\alpha', \beta') < (\alpha, \beta)} s_{\alpha'} \land r_{\beta'}\right).
\]

Hence, by induction rule \(\text{N} \vdash 16\), we can derive the theorem.

Case 4:

\[
\text{Spec}\[u_1] \vdash (p_1 \land i) \equiv p_1',
\]

\[
\text{Spec}\[u_1] \vdash \text{always}(u_1, \varphi_{u_1})(i),
\]

\[
\text{Spec}\[u_1] \vdash p_1' \overset{u_1}{\rightarrow} q_1,
\]

\[
\text{Spec}\[u_2] \vdash p_2 \overset{u_2}{\rightarrow} q_2.
\]
By derivation, we write:

\[ Spec[u_1] \cup Spec[u_2] \not\models (p_1 \land p_2 \land i) \equiv (p'_1 \land p_2), \]

\[ Spec[u_1] \cup Spec[u_2] \not\models always(u_1 || u_2, \varphi_{u_1||u_2})(i), \]

\[ Spec[u_1] \cup Spec[u_2] \not\models (p'_1 \land p_2) \iff (q_1 \land q_2). \]

Finally, we have to prove that the new system is complete with respect to the wp semantics. We think that some rules are not useful and can be cancelled but we have to do a careful study.

5. A simple case study: the sorting problem

A very classical example is detailed and we use Theorem 4.2 to deal with composition of programs with respect to its proofs. The sorting problem allows to show how to improve a given solution. Moreover, the initial (first) specification of this problem is simple.

The problem is stated in a pet but mathematical language as follows: “Let \( r \) be a list of elements. Build a list \( t \) of elements satisfying:

\[ \rightarrow t \text{ is a permutation of } r, \]
\[ \rightarrow t \text{ is ordered.} \]

We transform this statement into a formal and first-order language-based specification by a mental process that we do not know how to describe in a formal framework but we will try to solve this aspect in further investigations.

The following notations are specified:

- Let \( s \) be a mapping from \( \{1, \ldots, n\} \) with \( n \in \mathbb{N} \) to \( Y \). We assume that \( Y \) is a set of values and that there exists a total ordering on it denoted by \( < \). \( s \) will be denoted by \( s: \{1, \ldots, n\} \rightarrow Y \).

- Let \( s_1: \{1, \ldots, n\} \rightarrow Y, s_2: \{1, \ldots, n\} \rightarrow Y \).

\[ p(s_1, s_2) \overset{\text{def}}{=} (\exists \sigma: \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}, (\forall i \in \{1, \ldots, n\}, s_1(\sigma(i)) = s_2(i))). \]

It means that \( s_1 \) and \( s_2 \) are the same sequence but not in the same order.

- Let \( s: \{1, \ldots, n\} \rightarrow Y \).

\[ o(s) \overset{\text{def}}{=} (\forall i \in \{1, \ldots, n-1\}, (s(i) < s(i+1))). \]
It means that \( s \) is ordered with respect to \(<\).

Now we can state the formal specification of the sorting problem:

\[
\forall r: \{1, \ldots, n\} \rightarrow Y, \exists t: \{1, \ldots, n\} \rightarrow Y. p(r, t) \land o(t).
\]

We apply now an action of operationalization on the specification. It means that we
dynamize the computation process hidden in the static specification by introduction
of the process constant, namely, \( \pi \).

\[
\forall r: \{1, \ldots, n\} \rightarrow Y, \forall t: \{1, \ldots, n\} \rightarrow Y. p(r, t) \rightarrow p(r, t) \land o(t).
\]

The expression "\( \pi \)" is an abbreviation for "\( \pi: [\pi] \)". \( r \) and \( t \) are variables of \( \pi \), i.e. they
have a location into the memory and can be referenced. \( r \) and \( t \) are now computer
variable values. The initial condition of \( \pi \) is \( \phi_\pi \overset{\text{def}}{=} (t = r) \land I_\pi \) and
\( I_\pi \overset{\text{def}}{=} (t, r: \{1, \ldots, n\} \rightarrow Y) \land \langle < \text{ is an ordering over } Y \rangle \land (n \in \mathbb{N}) \rangle. \) The rule of trans-
formation from a static specification to a dynamic specification is called rule of
operationalization.

Using \( \mathcal{N} \mathcal{W} \), we can derive the following theorem.

**Theorem 5.1.** \( \{ p(r, t) \overset{\pi: [\pi]}{\rightarrow} p(r, t) \land o(t) \} \quad \overset{\delta \in \mathcal{W}}{\vdash} \quad p(r, t) \overset{\pi}{\rightarrow} p(r, t) \land o(t). \)

This theorem is derived using the rule of fairness (Rule \( \mathcal{N} \mathcal{W} 15 \)). Now we are going
to modify \( \pi \) to obtain a solution satisfying the same theorem but more complex than
this one. A transformation on \( \pi \) allows to obtain \( \pi_1 \) by noting that permutations
satisfy the following theorems.

**Theorem 5.2.** Let \( s: \{1, \ldots, n\} \rightarrow Y \). There exists one and only one permutation
\( \sigma: \{1, \ldots, n\} \rightarrow \{1, \ldots, n\} \) satisfying
\( o(\text{perm}(s, \sigma)) \), where \( \text{perm}:(\{1, \ldots, n\} \rightarrow Y) \times ((\{1, \ldots, n\} \rightarrow Y) \rightarrow (\{1, \ldots, n\} \rightarrow Y)) \rightarrow (\{1, \ldots, n\} \rightarrow Y) \) so that \( \text{perm}(s, \sigma)(i) = s(C_j(i)) \).

The permutation \( \sigma \) sorting \( s \) is denoted as \( \sigma_s \). We define a new action related to the
application of \( \sigma_s \) and denote it as \( \pi_{\sigma_s} \). \( \pi_{\sigma_s} \overset{\text{def}}{=} t := \text{perm}(t, \sigma_s) \). We can rewrite the
previous theorem as: \( p(r, t) \overset{\pi_{\sigma_s} \overset{[\pi_{\sigma_s}]}{\rightarrow}}{\rightarrow} p(r, t) \land o(t). \)

The set of permutations is finite and denoted by \( \Pi[\mathbb{N}] \). \( p(r, t) \) can be rewritten as
\( \bigcup_{s \in \text{Config}(r, \Pi[\mathbb{N}])} (t = s) \), where \( \text{Config}(r, \Pi[\mathbb{N}]) = \{t: \{1, \ldots, n\} \rightarrow \{1, \ldots, n\} / \exists \sigma \in \Pi[\mathbb{N}]. (t = \text{perm}(r, \sigma)) \} \). We derive the following properties:

\[
\begin{align*}
(1) \quad (p(r, r_1) \land (t = r_1)) \overset{\pi_{s_{r_1}} \overset{[\pi_{s_{r_1}}]}{\rightarrow}}{\rightarrow} p(r, t) \land o(t), \\
\ldots \\
(n!) \quad (p(r, r_n) \land (t = r_n)) \overset{\pi_{s_{r_n}} \overset{[\pi_{s_{r_n}}]}{\rightarrow}}{\rightarrow} p(r, t) \land o(t).
\end{align*}
\]
The $\mathcal{N}W$ system as a development system: $\delta N\mathcal{W}$

331

Using the Rule $\mathcal{N}W$ 10 and the equational property of $p(r, t)$, we derive

\[(n! + 1) \ni \exists_{\text{strong}} x: \left[\pi_{\sigma}, \ldots, \pi_{\sigma_{n}}\right], p(r, t) \rightarrow p(r, t) \land o(t).\]

We have used the strong version of existential quantification and we will obtain a strongly fairly executed program. If we want a weakly fair execution, we must strengthen the guard by the condition $o(o(t))$. This guard stabilizes the current state until the good action is finally executed by fairness assumption. But the weak existential quantification introduction rules. Moreover, the following universal transition properties can be stated:

\[(n! + 2) \forall x: \left[\pi_{\sigma}, \ldots, \pi_{\sigma_{n}}\right], p(r, t) \rightarrow p(r, t),\]

\[(n! + 3) p(r, t) \rightarrow p(r, t) \land o(t) \text{ (by Rule } \mathcal{N}W 15).\]

Hence, the unity $\pi_1 \overset{\text{def}}{=} \left[\pi_{\sigma}, \ldots, \pi_{\sigma_{n}}\right]$.

The unity $\pi_1$ satisfies the specification but the property "$p(r, t) \land o(t)$" is not stable and invariant. We add a guard to any assignment "$r := \text{perm}(r, \sigma)$" and this guard $g$ must satisfy:

- $p(r, t) \land g \land o(t) \rightarrow (p(r, t) \land o(t)).$
- $p(r, t) \land g \land \neg o(t) \rightarrow (p(r, t) \land o(t)) \lor (p(r, t) \land \neg o(t)).$
- $p(r, t) \land o(t) \land \neg g \rightarrow p(r, t) \land o(t).$
- $p(r, t) \land \neg o(t) \land \neg g \rightarrow (p(r, t) \land o(t)) \lor (p(r, t) \land \neg o(t)).$

A good candidate for $g$ is $o(t)$. Hence, $\pi_2$ is obtained from $\pi_1$ by modifying the guard in $\pi_1$: $\pi_2 = \pi_1[\Pi x \in \pi_1.x \sim \neg o(t)/x]$.

Now, we try to make simpler the expression of $\pi_2$ and we recall the following theorem.

**Theorem 5.3.** Any permutation $\sigma$ of $\Pi\left[n\right]$ is expressible as a finite composition of transposition of $\Pi\left[n\right]: \sigma = \tau_1 \circ \cdots \circ \tau_p$.

The first idea is to restrict the unity $\pi_2$ to transposition, but this solution is not correct: for any transposition $\tau$, $\tau \circ \tau = \text{Id}$. For any transposition $\tau$, we define by $I(\tau)$ and $J(\tau)$ the two elements of $\{1, \ldots, n\}$ so that $\tau(I(\tau)) = J(\tau)$ and $\tau(J(\tau)) = I(\tau)$ since $\tau$ is a transposition. The idempotence of transpositions must be weakened. The guard $g$ must be modified as follows: $\sim o(t) \land (t(I(\tau)) > t(J(\tau))).$

We obtain a new unity, namely $\pi_3$, and this satisfies the specification. The program or unity $\pi_3$ that we have obtained is nondeterministic and is fairly executed. We need the following notations to derive the proof of $\pi_3$. 

• \text{distance}(t(i), t(j)) = \text{abs}(i-j).
• \text{Min}(t[a,b]) = t(k) \ (k \in [a,b]) \wedge (\forall j \in [a,b]. t(k) < t(j).

Now, we can proceed to the proof:

1. \(\forall a \in \{0, \ldots, n\}, \forall b \in \{0, \ldots, n-1\}. T_{a,b}^{\text{def}} = p(r, t) \wedge o(t[1, n-a]) \wedge (t[1, n-a] < t[n-a+1, n]) \wedge (\text{distance}(\text{Min}(t[n-a+1, n]), t[n-a+1]) = b).\)

2. \(\forall a \in \{2, \ldots, n\}, \forall b \in \{1, \ldots, n-1\}. \forall \tau \in \text{Transposition}(n). T_{a,b} \xrightarrow{\tau, \pi_3} (T_{a,b-1} \lor (((a', b') < (a, b)) \wedge T_{a,b}^{	ext{weak}})) \lor T_{a,b} \text{ (property of } T_{a,b} \text{ using transition rules).}\)

3. \(\forall a \in \{2, \ldots, n\}, \forall b \in \{1, \ldots, n-1\}. \exists \tau \in \text{Transposition}(n). T_{a,b} \xrightarrow{\tau, \pi_3} (T_{a,b-1} \lor (((a', b') < (a, b)) \wedge T_{a,b}^{	ext{weak}})) \text{ (property of } T_{a,b} \text{ using transition rules).}\)

4. \(\forall a \in \{2, \ldots, n\}, \forall b \in \{1, \ldots, n-1\}. \forall \tau : \pi_3. T_{a,b} \xrightarrow{\tau, \pi_3} (T_{a,b-1} \lor (((a', b') < (a, b)) \wedge T_{a,b}^{	ext{weak}})) \lor T_{a,b} \text{ (by introduction of universal quantification).}\)

5. \(\forall a \in \{2, \ldots, n\}, \forall b \in \{1, \ldots, n-1\}. \exists \text{weak } \tau : \pi_3. T_{a,b} \xrightarrow{\tau, \pi_3} (T_{a,b-1} \lor (((a', b') < (a, b)) \wedge T_{a,b}^{	ext{weak}})) \text{ (by introduction of existential quantification).}\)

6. \(\forall \tau \in \{2, \ldots, n\}, \forall \tau \in \{1, \ldots, n-1\}. T_{a,b} \xrightarrow{\tau, \pi_3} (T_{a,b-1} \lor (((a', b') < (a, b)) \wedge T_{a,b}^{	ext{weak}})) \text{ (by fairness rule } \mathcal{N} \mathcal{W}_1 15).\)

7. \(\forall a, b. T_{a,b} \xrightarrow{\pi_3} T_{0,0} \text{ (by the rule of induction } \mathcal{N} \mathcal{W}_1 16).\)

8. \(p(r, t) \xrightarrow{\pi_3} p(r, t) \wedge o(t).\)

We analyze the specification above and identify a partition of actions into three disjoint sets:

\[\pi_3 = \tau_1 \cup \tau_2 \cup \tau_3,\] where
\[\tau_1 = \{\tau : \text{Transposition}(n) \mid I(\tau), J(\tau) \in A\},\]
\[\tau_2 = \{\tau : \text{Transposition}(n) \mid I(\tau), J(\tau) \in B\},\]
\[\tau_3 = \text{Transposition}(n) - \tau_1 - \tau_2,\]
\[A \cup B = \{1, \ldots, n\} \text{ and } A \cap B = \emptyset.\]

The choice of A and B may lead to different solutions and we first choose A and B so that A = \{1, \ldots, p\} and B = \{p + 1, \ldots, n\}. Hence, we use the specification of \(\pi_3\) for \(\tau_1\). It means that \(\tau_1\) sorts t from 1 to \(p\) and \(\tau_2\) sorts t from \(p+1\) to \(n\).

The actions of \(\tau_1\) and \(\tau_2\) satisfy the assumptions of Rule \(\mathcal{N} \mathcal{W}_2\). Hence, \(p(r, t) = p(r_1, t_1) \wedge p(r_2, t_2),\) where \(r_1 = r[A]\) and \(r_2 = r[B].\)

We derive the property
\[p(r, t) \xrightarrow{\tau_1 \cup \tau_2} p(r, t) \wedge o(t_1) \wedge o(t_2) \wedge t = t_1 \cdot t_2.\]

Now we have to find a unity \(\mu\) so that
\[p(r, t) \wedge o(t_1) \wedge o(t_2) \wedge t = t_1 \cdot t_2 \xrightarrow{\mu} p(r, t) \wedge o(t).\]
Hence, we have two specifications of \(\tau_1\) and \(\tau_2\). But they are now encapsulated and are not merged together. \(\mu\) merges the two lists \(t_1\) and \(t_2\). But, the unity \(\tau_3\) possibly exchanges the two values of \(t_1\) and \(t_2\). Hence, \(\mu\) satisfies the same specification as \(\tau_3\)...

No! The proof is too sketched and we have to require that \(t_1 \leq t_2\) to infer the global property.

Finally, the right unity \(\left\langle \tau_3;[\tau_1 || \tau_2]\right\rangle\) satisfies the specification of the sorting problem. And now we prove it as follows:

Some new notations are needed:

- \(\text{associate}(i) = \{ j \in \text{Dom}(t_2) | t_1(i) > t_2(j) \}\).
- \(\text{permutable}(i) = \{ \tau \in \text{Transposition}(n) | (I(\tau) = i) \land (J(\tau) \in \text{associate}(i)) \}\).
- \(\text{permutable} = \bigcup_{i \in \text{Dom}(t_1)} \text{permutable}(i)\).
- \(a(t) = \text{Card}(\{ i \in \text{Dom}(t_1) | \text{associate}(i) \neq \emptyset \})\).
- \(b(t) = \text{Sup} \{ i \in \text{Dom}(t_1) | \text{permutable}(i) \}\).
- \(R_{a,b} \triangleq p(r,t) \land (t = t_1 \cdot t_2) \land (a(t) = a) \land (b(t) = b)\).
- \(g_t = (t_1(I(\tau)) > t_2(J(\tau)))\).

\[
\begin{align*}
(1) & \quad \forall \tau \in \tau_3. R_{a,b} \xrightarrow{\tau \cdot g_t : \tau_3} (R_{a,b-1} \lor R_{a-1,b'} \land (b' < b)) \lor R_{a,b} \quad \text{(by property from definition of actions).} \\
(2) & \quad \exists \tau \in \tau_3. R_{a,b} \xrightarrow{\tau \cdot g_t : \tau_3} (R_{a,b-1} \lor R_{a-1,b'}) \quad \text{(by property from definition of actions).} \\
(3) & \quad \forall \tau \sim g_t : \tau_3. R_{a,b} \xrightarrow{\tau \cdot g_t : \tau_3} (R_{a,b-1} \lor R_{a-1,b'}) \lor R_{a,b} \quad \text{(by transition rules from 3.1).} \\
(4) & \quad \exists \tau \sim g_t : \tau_3. R_{a,b} \xrightarrow{\tau \cdot g_t : \tau_3} (R_{a,b-1} \lor R_{a-1,b'}) \quad \text{(by transition rules from 3.1).} \\
(5) & \quad R_{a,b} \xrightarrow{\tau_3} (R_{a,b-1} \lor R_{a-1,b'}) \quad \text{(by fairness rule N\&U 15).} \\
(6) & \quad \bigvee_{a,b} R_{a,b} \xrightarrow{\tau_3} R_{0,0} \quad \text{(by the induction rule N\&U 16).} \\
(7) & \quad R_{0,0} \Rightarrow p(r,t) \land (t = t_1 \cdot t_2) \land (t_1 < t_2) \quad \text{(by definition of R}_{0,0}\). \\
(8) & \quad p(r,t) \land (t = t_1 \cdot t_2) \xrightarrow{\tau_3} p(r,t) \land (t = t_1 \cdot t_2) \land (t_1 < t_2) .
\end{align*}
\]

Now, the composition of different properties according to our theorem leads to the correct solution for the sorting problem in a concurrent way.

6. Conclusion and future works

The design of programs is a very complex task but this task is yet more complex in the concurrency framework. This paper illustrates some experiments to use a formal system, namely N\&U, to deal with design, while it is usually used to prove properties of existing and effective programs. This work can be considered as a step towards a more
realistic programming language where the sequentiality and concurrency exist together. Our future works tend to specify notions as proofs, developments, specifications in a more precise way. Examples are yet to develop in this direction.

References