Electronic Notes in Theoretical Computer Science 67 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume67.html 20 pages

A Branching Time Logical System for Open Distributed Systems Development

Carlos H. C. Duarte [1,2]

BNDES, Av. Chile 100, Rio de Janeiro, RJ, Brasil, 20001-970
Universidade Estácio de Sá, Rua do Bispo 83, Rio de Janeiro, RJ, Brazil, 20261-902

Tom Maibaum [3]

Department of Computer Science, King's College London, Strand, London, UK, WC2R 2LS

Abstract

We propose a new first-order many-sorted branching-time logical system with equality devoted to support the development of open distributed systems. The inherent characteristics of this family of systems, which ought to be treated in rigorous software development processes, are used throughout the paper to motivate the specific features of our formalism. We present a solution of the drinking philosophers problem as a way of illustrating the application of this new logical system.

Keywords: Temporal Logic; Specification; Verification; Distributed Systems; Software Development.

1 Introduction

Since a seminal work of Amir Pnueli in the late seventies [20], temporal logics have been applied with great success in many branches of Computer Science, including requirement engineering, software specification and verification, stepwise refinement and systematic testing. Regarding the development of concurrent systems, Manna and Pnueli initially showed how temporal proof systems could be associated to programming languages in a natural way [17]. Next, Barringer solved an important problem of composability [5], making it possible to rely on the structure of concurrent programs in proving temporal properties. Subsequently, Fiadeiro and Maibaum raised the abstraction level of his work by also showing that concurrent software systems could be specified and verified in a modular way based on temporal theory presentations [10]. This chain of results concerns different but connected stages of rigorous software development processes.

[1] Partially supported by CNPq research grant number 301037/00-0.

[2] E-mail: carlos.duarte@computer.org

[3] E-mail: tom@maibaum.org

The choice to adopt specific software development approaches and the necessity to treat particular application domains have played a central role in the decision to either choose an existing temporal logical system or develop a new one. For instance, Sernadas and others developed a linear time logical system particularly tailored to object-oriented approaches [21]. Lamport and Abadi have applied the temporal logic of actions in a multitude of domains [15], treating in particular the development of distributed fault-tolerant systems. What is involved in this kind of decision is the identification of the inherent characteristics of the approach or domain at hand, together with an assessment of each logical system — in terms of its language, models, axiom schemes, inference rules and automated support — in order to determine if these inherent characteristics are supported by the formalism in ways that somehow facilitate software development.

Nowadays, with the advent of networking technologies making it possible to deploy computing power over wide geographic areas, there exists a growing interest in software development approaches that not only help identifying distribution as a problem requirement but also allow software engineers to introduce distribution as a design or implementation decision during the development process. *Distributed systems* may be characterised as collections of loosely interconnected not necessarily co-located objects. Objects in this context are an abstraction of more concrete entities such as autonomous agents or computing units. Incidentally, our definition admits concurrent systems as a particular case. Distributed objects may be put together in the form of components, which have explicit interfaces with their surrounding environment. Those distributed systems or components which retain almost none control over their environment are said to be *open*, an important property not only to allow design independence but also to enforce composability, incrementability and reuse of the respective software artifacts.

Despite their current appeal, open distributed systems appear to receive little support directed towards software development in existing logical systems. This seems to be the case mostly due to the complexity of their inherent characteristics. Distributed objects may interact through diverse modes such as message passing or resource sharing. The number of participants may vary in each interaction mode. The internal and external configuration of a component may be fixed at design time or change dynamically. Ideally, a logical system devoted to open distributed systems development should permit the specification and verification of all the details present in distributed system models.

In this paper, we propose a new temporal logical system which we consider appropriate for developing open distributed systems in a rigorous way. Our technical contribution is organised in two parts. In Section 2, we describe the semantic models and the proof calculus of our logical system. In Section 3, we illustrate these formal constructions presenting a solution of the drinking philosophers problem [6]. We conclude the paper with some remarks concerning related and future research.

2 A Logical System for Open Distributed Systems

2.1 Languages and Models

Our logical system is organised in terms of a family of languages, each of which defined using logical and extra-logical symbols. We depart from the collection of first-order languages and adopt a countably infinite family of logical variables $\mathcal{V}$. It is possible to represent in this way the potentially infinite number of messages exchanged by the objects of some distributed systems, something out of reach of conventional propositional temporal logical systems as extensively studied by Sistla [22], Koymans [12] and others. We also adopt many-sorted languages, since sort symbols appear to make logical constructions more readable and structured. Sorts and all the other extra-logical symbols belong to signatures, which are defined below:

Definition 2.1 (Signature) A *signature* $\Delta = (\Sigma, \Gamma, \mathcal{A})$ is a triple of disjoint and finite sets of symbols such that:

- $\Sigma = (S, \Omega)$ is a universe signature: S is a set of *sort symbols* and Ω is an $S^*_{fin} \times S$-indexed family of *function symbols*;
- Γ is an S^*_{fin}-indexed family of *action symbols*;
- $\mathcal{A}$ is an $S^*_{fin} \times S$-indexed family of *attribute symbols.*

The indexing of signature symbols using sorts induces a type over each language expression. A functional construction taking arguments of sort $\langle s_1, \ldots, s_n \rangle$ and producing results of sort s, i.e. indexed by $S^*_{fin} \times S$, is said to have type $\mathit{type}(f) = \langle s_1, \ldots, s_n \rangle \to s$.

Each kind of signature symbol has a different interpretation. Sort symbols represent non-empty collections of values. The symbols in Σ, as well as the variables in $\mathcal{V}$, are total and rigid, i.e. they always have a meaning which does not vary as time passes. On the other hand, the elements of $\Gamma \cup \mathcal{A}$ are flexible, meaning that they have a time dependent interpretation. Action symbols in Γ denote the occurrence of instantaneous events. They are different from the attribute symbols in $\mathcal{A}$, which are total and functional, representing current state. Variables, functions, actions and attributes with this kind of interpretation appear quite often in open distributed systems representations.

Before formalising the meaning of signature symbols, we need to clarify the nature of the adopted time flows. The usual view that the whole universe came into existence at once leads us to consider that all time flows have their beginning at the same instant. Note that distributed systems usually have a distinguished point in time from which they begin to present some observable behaviour and the adopted flows of time allow us to represent this characteristic easily. For the same reason, we choose to deal only with infinite time flows here, which also comply with the general situation in distributed systems of not necessarily having to exhibit only stable properties from a specific point in time onwards. Concerning distributed programs, an example of this kind of property is termination, a unusual requirement in this domain which turns out to be representable using the adopted time flows as well. We restrict our attention to discrete flows here, for the sake of simplicity.

Temporal flows are normally classified according to their linear or branching nature. Most temporal logics of programs, although based on linear time flows, also rely on an additional enabledness predicate to represent the possibility of some subsequent computations. The design of distributed systems which are open also appears to require a way of expressing possibility, not just to represent possible computation steps but also to establish looser connections with their environment, which are an inherent part of each problem. This is so because requiring the environment to behave in a particular manner may contradict the openness property. However, we only know how to express possibility using branching time flows. In fact, possibility is a kind of property defined by current and/or future events, but it is also conceivable to take into account time flows which branch towards the past. Considering that in distributed systems the past is regarded as the necessary sequence of events to justify the present situation and also that in initiallised time flows it is usually possible to adopt an expressively complete set of future time connectives to express any relevant property, we are relieved from treating the additional complexity of the past here.

Two plausible and completely distinct views concerning the definition of branching time are available in the literature [7,26]. The so-called Peircean view advocates that a formula asserting the eventual occurrence of a proposition is true at a given moment x if and only if the proposition is true at some moment in some future of x. Conversely, the Ockhamist view argues that it is meaningless to discuss the truth value of a formula unless additional information concerning the actual future is provided. To clarify this distinction, a metaphor can be defined. Assume that a system and two omniscient observers are given, Eager and Lazy. Both see the system evolving almost as defined in the previous paragraphs, in that each time flow has an initial instant, is discrete and infinite. Eager, who adopts a Peircean view, politely ignores everything else he knows and follows the system closely, adopting as his own current moment in time that of the observed system. According to his perceptions, what will happen in the future spans as many branches of undetermined possibilities. Lazy, on the contrary, adopts an Ockhamist view and prefers to stay outside any conceivable time frame. He can only see the distinct behaviours of the system as a set of linear terminated sequences. For him, what could have otherwise been the case at some moment of a behaviour of the system is defined in terms of other possible behaviours. Comparing these two conceptually distinct views, we can conclude that what is regarded as a branching-time logic depends on the chosen kind of observer. Both are reasonable views that allow us to talk about possibility.

Mathematical models for branching time abound in the literature. The research on semantic models of temporal logics has resulted in a wide range of mathematical structures [24]. In the realm of concurrent systems, event structures and transition systems have been preferred [23]. Here, we adopt structures of the second kind to capture an Ockhamist view of branching time, since we can define in this way a linear time fragment of the logic based on future time connectives and consequently rely on a standard set of axiom schemes and inference rules, which is rather standard. Therefore, we represent branching time structures as sets of linear sequences of worlds and capture branching points in time through the initial points of distinction in such sequences.

Definition 2.2 (Branching Structure) A *branching-time structure* or *frame* is a tuple $(\alpha,\ \alpha_0,\ \rho,\ \Lambda)$ where:

- α and $\alpha_0 \subseteq \alpha$ are sets of *worlds* and *initial worlds* respectively;
- $\rho : \alpha \to \mathcal{P}(\alpha)$ is the *accessibility relation* (a powerset function);
- Λ is a non-empty set of *possible behaviours*. Each $L \in \Lambda$ is a function such that:

(i) $\mathsf{dom}\ L \subseteq \alpha$ and $\mathsf{cod}\ L \overset{\text{def}}{=} \mathbf{N}$;
(ii) $\forall w \in \mathsf{dom}\ L \cdot L(w) = 0 \leftrightarrow w \in \alpha_0$;
(iii) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w) = L(w') \to w = w'$;
(iv) $\forall n \in \mathsf{cod}\ L \cdot \exists w \in \mathsf{dom}\ L \cdot L(w) = n$; and
(v) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w') = L(w) + 1 \to w' \in \rho(w)$.

Each sequence of worlds determining a behaviour in Λ (not necessarily of any computer program) is in a one to one correspondence with the natural numbers, according to items (1), (3) and (4). Hence, every $L \in \Lambda$ is invertible and we use this fact to define the meaning of the adopted branching-time modality.

Based on branching-time structures, signature symbols are interpreted as follows:

Definition 2.3 (Interpretation Structure) An *interpretation structure* for a signature $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, $\Sigma = (S, \Omega)$, is a tuple $\theta = (B, U, G, A)$ where:

- B is a branching-time structure;
- U maps each $s \in S$ to a non-empty collection s_U and each $f \in \Omega$, $type(f) = \langle s_1, \dots, s_n\rangle \to s$, to $f_U : s_{1_U} \times \dots \times s_{n_U} \to s_U$;
- G maps each $g \in \mathcal{A}$, $type(g) = \langle s_1, \dots, s_n\rangle \to s$, to a $G(g) : s_{1_U} \times \dots \times s_{n_U} \to \alpha \to s_U$;
- A maps each $a \in \Gamma$, $type(a) = \langle s_1, \dots, s_n\rangle$, to $A(a) : s_{1_U} \times \dots \times s_{n_U} \to \mathcal{P}(\alpha)$.

Whenever α appears in the interpretation of some symbols, this is connected to their flexible, time-dependent meaning. Consequently, attribute symbols denote total functions which vary with time and action symbols denote events that may happen at some time instants in parallel among themselves or with respect to other events belonging to enclosing environments. On the other hand, sort symbols are interpreted as fixed sets of values and function symbols as immutable total functions.

Terms stand for meaningful values. In their definition, we assume given a partial classification Ξ_Δ for each signature Δ. Ξ_Δ associates to some variables sort symbols in Δ. For a symbol $s \in S_\Delta$, we also represent the set of s-classified variables as $\Xi_s \overset{\text{def}}{=} \{v \in \mathcal{V} | \Xi_\Delta(v) = s\}$. The set of terms generated by a signature Δ is defined below:

Definition 2.4 (Terms) The S-indexed set of *terms* $T(\Xi_\Delta)$ is defined as follows, provided that $x \in \Xi_s \cup \Omega_s \cup \mathcal{A}_s$; $f \in \Omega_\Delta$, $g \in \mathcal{A}_\Delta$ and $type(f) = type(g) = \langle s_1, \dots, s_n\rangle \to s$; and $t_i \in T(\Xi_\Delta)_{s_i}$, $i \in \{1, \dots, n\}$:

$$t ::= x \mid f(t_1, \dots, t_n) \mid g(t_1, \dots, t_n)$$

That is, a term is a variable, a nullary function or attribute symbol[4], or a term is a function or attribute symbol applied to a sequence of terms. Note that the indexing of signature symbols using sorts extends to terms in a natural way.

We interpret terms as defined below. Because we have a first-order logic, we need to define first how logical variables are assigned to the elements of quantification domains:

Definition 2.5 (Assignment) Given a structure $\theta = (B, U, G, A)$ for a signature $\Delta = ((S, \Omega), \mathcal{A}, \Gamma)$, an *assignment* N for θ maps each Ξ_s to s_u, for each $s \in S_\Delta$.

Definition 2.6 (Interpretation of Terms) Given an interpretation structure $\theta = (B, U, G, A)$ for a signature $\Delta = ((S, \Omega), \mathcal{A}, \Gamma)$ and an assignment N for θ, $[]^{\theta,N} : \alpha \to s_U$ defined below is an *interpretation of terms* of sort $s \in S$ at a world $w \in \alpha_B$:

- $[\![x]\!]^{\theta,N}(w) \overset{\text{def}}{=} N(x)$ if $x \in \Xi_s$;
- $[\![f(t_1, \dots, t_n)]\!]^{\theta,N}(w) \overset{\text{def}}{=} f_U([\![t_1]\!]^{\theta,N}(w), \dots, [t_n]^{\theta,N}(w))$;
- $[\![g(t_1, \dots, t_n)]\!]^{\theta,N}(w) \overset{\text{def}}{=} (G(g)([\![t_1]\!]^{\theta,N}(w), \dots, [t_n]^{\theta,N}(w)))(w)$.

[4] Nullary attribute symbols are called flexible constants in [3].

Formulae are written in terms of logical and extra-logical symbols and appear in specifications and proofs. The set of logical symbols consists in the set of variables $\mathcal{V}$ together with some classical and temporal connectives. Apart form the uses of logical equality, universal quantifier and propositional connectives $\rightarrow$ and $\neg$, atomic formulae are the "beginning of time" connective **beg** and the application of an action symbol over a sequence of terms. Define the set of formulae those considered atomic together with applications of the non-strict "in all alternative current instants" connective **A** and the strict "strong until" connective **V**.

Definition 2.7 (Formulae) The set $F(\Xi_\Delta)$ of *formulae* is defined by the production rule below, provided that $c \in \Gamma_\Delta$, $type(c) = \langle s_1, \ldots, s_n \rangle$; $t_i \in T(\Xi_\Delta)_{s_i}$, $i \in \{1, \ldots, n\}$; $x \in \Xi_s$ and $q_j \in F(\Xi_\Delta)$, $j \in \{1, 2\}$:

$$q ::= (t_1 = t_2) \mid q_1 \rightarrow q_2 \mid \neg q_1 \mid \forall x \cdot q_1 \mid c(t_1, \ldots, t_n) \mid \mathbf{beg} \mid \mathbf{A}(q_1) \mid (q_1)\mathbf{V}(q_2)$$

The set of expressions is defined as follows: $E(\Xi_\Delta) \stackrel{\text{def}}{=} T(\Xi_\Delta) \cup F(\Xi_\Delta)$.

The nullary connective **beg** denotes the beginning of time in each behaviour. Given formulae p and q, $\mathbf{A}(p)$ means that p happens in any instant that could succeed the actual past history and $(p)\mathbf{V}(q)$ means that p occurs strictly in the future (i.e. after the current moment) and q happens from the next instant until but not necessarily including the moment of that occurrence[5]. We prefer to use this strict strong until operator **V** as proposed by Kamp because other usual linear future time connectives all become definable in this way. The inclusion of the universal modality **A** in our languages makes the dual **E** definable, the modality of possibility.

Our branching-time modality is interpreted using an equivalence relation $\simeq$ over behaviour prefixes. We define this relation in a pointwise manner, saying that two worlds of a frame are equivalent if and only if all the attribute and action symbols have identical interpretation in both worlds. The satisfaction of logical formulae is defined below:

Definition 2.8 (Satisfaction of Formulae) Given a signature Δ, the *satisfaction* of a Δ-formula at world w_i of a behaviour L (i.e., $w_i \in \mathsf{dom}\ L$) by a structure $\theta = (B, U, G, A)$ with assignment N is defined as follows:

- $(\theta, N, L, w_i) \models a(t_1, \ldots, t_n)$ iff $w_i \in A(a)([\![t_1]\!]^{\theta,N}(w_i), \ldots, [\![t_n]\!]^{\theta,N}(w_i))$;
- $(\theta, N, L, w_i) \models \neg p$ iff it is not the case that $(\theta, N, L, w_i) \models p$;
- $(\theta, N, L, w_i) \models p \rightarrow q$ iff $(\theta, N, L, w_i) \models p$ implies $(\theta, N, L, w_i) \models q$;
- $(\theta, N, L, w_i) \models \forall x \cdot p$ iff for every $v \in \mathsf{cod}\ N$ and every assignment N_v for θ such that $N_v(y) = N(y)$ if $y \neq x$ and $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models p$;
- $(\theta, N, L, w_i) \models (t_1 = t_2)$ iff $[\![t_1]\!]^{\theta,N}(w_i) = [\![t_2]\!]^{\theta,N}(w_i)$;
- $(\theta, N, L, w_i) \models \mathbf{beg}$ iff $L(w_i) = 0$;
- $(\theta, N, L, w_i) \models p\mathbf{V}q$ iff there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$, $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models q$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$;
- $(\theta, N, L, w_i) \models \mathbf{A}p$ iff for every $L_j \in \Lambda$ such that $w_k \simeq (L_j^{-1} \circ L)(w_k)$ for each $w_k \in \mathsf{dom}\ L$ with $L(w_k) < L(w_i)$, $(\theta, N, L_j, (L_j^{-1} \circ L)(w_i)) \models p$.

[5] $p\mathbf{V}q$ (Kamp) $\equiv$ $\mathbf{V}(p, q)$ (Gabbay [11]) $\equiv$ $q\hat{\mathbf{U}}p$ (Manna and Pnueli [18]).

We adopt interpretation structures as models of logical formulae. Whenever a formula has a model, the sets of worlds α and α_0 in the underlying frame are not empty.

Now we can define the usual scale of degrees of validity. Definition 2.8 corresponds to *satisfiability.* We say that a Δ-formula p is *(locally) true* in an interpretation structure $\theta = (B, U, A, G)$ for Δ at world s of a behaviour L if and only if for every assignment N, $(\theta, N, L, s) \models p$. In addition, p is said to be *valid* in θ if and only if it is true in each possible behaviour L and world s. A formula is considered to be *universally valid* if and only if it is valid in any possible structure for Δ. A *semantic consequence or entailment relation* $\Psi \models_{\Delta} p$ can be simply defined stating that q is valid in θ for every $q \in \Psi$ implies that p is valid in θ, for any admissible structure θ for Δ.

Other classical connectives such as $\bot$, $\top$, $\wedge$, $\vee$ and $\leftrightarrow$ are defined as usual. The following definitions of temporal connectives are also helpful in specifications and proofs:

- $p\mathbf{U}q \overset{\text{def}}{=} q \vee (p \wedge q\mathbf{V}p)$ (p happens until q does);
- $\mathbf{X}p \overset{\text{def}}{=} p\mathbf{V}\bot$ (p happens in the next instant);
- $\mathbf{F}p \overset{\text{def}}{=} \top\mathbf{U}p$ (p happens now or sometime in the future);
- $\mathbf{G}p \overset{\text{def}}{=} \neg\mathbf{F}(\neg p)$ (henceforth p happens);
- $p\mathbf{W}q \overset{\text{def}}{=} \mathbf{G}p \vee p\mathbf{U}q$ (p happens until q does, even if q never happens);
- $\mathbf{E}p \overset{\text{def}}{=} \neg\mathbf{A}(\neg p)$ (p is currently possible).

2.2 Axiom schemes and inference rules

It should be clear at this point that we are dealing with a first-order many-sorted branching-time logical system with equality. To complete its definition, we shall define in what follows the *logical consequence relation* $\vdash_{\Delta}$ for each admissible signature Δ, in an attempt to create a proof-theoretic approximation of the semantic consequence relation defined in the previous section.

We consider that $\Psi \vdash_{\Delta} p$ is a binary relation defined by a non-empty set of *derivations* $Pr_{\Delta}(\Psi, p)$ of the *conclusion* p from the set of *hypotheses* Ψ. Pr is in turn generated by $Ax(\Delta)$, a set of *axioms*, and $\mid\!\sim_{\Delta}$, a *derivability relation.* $Ax(\Delta)$ only contains instances of *axiom schemes*, while $\mid\!\sim_{\Delta}$ is generated by applications of *inference rules.* Derivations are directed acyclic graphs with vertices labelled with formulae and edges directed from the hypotheses to the conclusion, such that there are no branching paths and, if p is a label, $p \in Ax(\Delta)$, $p \in \Psi$ or $\Psi' \mid\!\sim_{\Delta} p$ for a set of *premises* Ψ' derived from Ψ. In order to make derivations more readable, we also consider that source vertices have an incoming edge, labelled with the name of the corresponding axiom scheme or with **Hyp** in the case of hypotheses, and that standard edges are labelled with the name of the respective justifying inference rule.

Our axiomatisation begins with the classical propositional fragment of the logic. Schematic variables $\{p, q, r, s\} \subseteq F(\Xi_{\Delta})$ are used for a given classification Ξ_{Δ}:

(A1-I) $\mid\!\sim_{\Delta} p \to (q \to p)$ (weakening);

(A2-I) $\mid\!\sim_{\Delta} (p \to (q \to r)) \to ((p \to q) \to (p \to r))$ (distribution);

(A3-N) $\mid\!\sim_{\Delta} (\neg p \to \neg q) \to (q \to p)$;

(R1-MP) $\{p, p \to q\} \mid\!\sim_{\Delta} q$.

As an example of a proof in Hilbert-style, we verify in what follows the derived inference rule **HS** (hypothetical syllogism). Since we are not going to state and prove a deduction theorem for our logical system (any such theorem would not have the usual formulation anyway due to the temporal character of our system), this inference rule should be extensively used. The rule is stated and derived as follows:

Lemma 2.9 (Inference Rule HS) The following inference rule for any $p \in F(\Delta)$ is derivable: **(HS)** $\{p \to q, q \to r\} \mid\!\sim_{\Delta} p \to r$.
Proof:
1. $p \to q$ **Hyp**
2. $q \to r$ **Hyp**
3. $(q \to r) \to (p \to (q \to r))$ **A1-I**
4. $p \to (q \to r)$ **R1-MP** 2, 3
5. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$ **A2-I**
6. $(p \to q) \to (p \to r)$ **R1-MP** 4, 5
7. $p \to r$ **R1-MP** 1, 6 ■

Now we turn to the more interesting axiomatisation of the linear time fragment of the logic. In the following axiom schemes and inference rules, we adopt the abbreviations of the non-strict connectives defined in the previous section:

(A4-GV) $\mid\!\sim_{\Delta} \mathbf{G}(p \to q) \to (p\mathbf{V}r \to q\mathbf{V}r)$;

(A5-GV) $\mid\!\sim_{\Delta} \mathbf{G}(p \to q) \to (r\mathbf{V}p \to r\mathbf{V}q)$;

(A6-V) $\mid\!\sim_{\Delta} p\mathbf{V}q \to p\mathbf{V}(q \wedge p\mathbf{V}q)$;

(A7-V) $\mid\!\sim_{\Delta} (p \wedge q\mathbf{V}p)\mathbf{V}p \to q\mathbf{V}p$;

(A8-V) $\mid\!\sim_{\Delta} p\mathbf{V}q \wedge r\mathbf{V}s \to (p \wedge r)\mathbf{V}(q \wedge s) \vee (p \wedge s)\mathbf{V}(q \wedge s) \vee (q \wedge r)\mathbf{V}(q \wedge s)$;

(A9-V) $\mid\!\sim_{\Delta} (p \vee q)\mathbf{V}r \to p\mathbf{V}r \vee q\mathbf{V}r$;

(A10-GX) $\mid\!\sim_{\Delta} \mathbf{G}(p \to \mathbf{X}p) \to (\mathbf{X}p \to \mathbf{G}p)$;

(A11-X) $\mid\!\sim_{\Delta} \mathbf{X}\top$;

(A12-Xbeg) $\mid\!\sim_{\Delta} \neg\mathbf{X}(\mathbf{beg})$;

(R2-G) $\{p\} \mid\!\sim_{\Delta} \mathbf{G}p$;

(R3-begG) $\{\mathbf{beg} \to \mathbf{G}p\} \mid\!\sim_{\Delta} p$.

These are obtained from axiomatisations which also consider the existence of a strong strict since connective, as discussed in [11], by removing this past-time connective and including **beg** instead. Schemes **A4**, **A5** and **A9** together with **R2** guarantee that we have a normal modal logic, which can be interpreted over relational structures. **A6-7** ensure the transitivity of these relations and we enter in this way the realm of temporal logics. **A8** in the presence of the other schemes implies that time is linearly ordered towards the future. In particular, due to our choice of initialised time flows, this is true everywhere. We also adopt **A10** to capture temporal induction. We use **A11** not only to guarantee that time flows do not have endpoints but also to ensure that there is always a next instant, capturing discrete time. Scheme **A12** says that no instant precedes the initial one. Rule **R2** is the usual temporal generalisation and **R3** may be called **begG**-elimination.

The reader may want to verify, using **HS** after having derived a replacement lemma for the above fragment of the logic, that schemes **A1-10** plus rules **R1-2** entail all the propositional theorems of the consequence relation of Manna and Pnueli in [17]. We use $\mathrel{|\!\sim}^{MP}_{\Delta}$ to denote the provability of such theorems here. Nevertheless, our extension of their logic is not conservative since we prefer to adopt flows of time with fixed characteristics in order to minimise the possibility of generating inconsistencies in composing open distributed system specifications. This would be the case if two composed specifications could assume respectively flows with and without endpoints, for instance.

The application of temporal logics in the design of software systems has been streamlined by the separation of temporal properties in two families and the respective development of reasoning principles [2]. Liveness or progress properties stating what a system eventually performs offer great challenges to verification methods and are usually treated using an additional well founded induction rule [8]. Safety properties, which define what a system always ensures, are verified using the following derived inference rule:

Theorem 2.10 (Inference Rule IND) The following inference rule for any $p \in F(\Delta)$ is derivable: **(IND)** $\{\mathbf{beg} \to p, \mathbf{G}(p \to \mathbf{X}p)\} \mathrel{|\!\sim}_{\Delta} p$.

Proof:

1. $\mathbf{beg} \to p$ **Hyp**
2. $\mathbf{G}(p \to \mathbf{X}p)$ **Hyp**
3. $\mathbf{G}(p \to \mathbf{X}p) \to (p \to \mathbf{G}p)$ $\mathrel{|\!\sim}^{MP}_{\Delta}$
4. $p \to \mathbf{G}p$ **R1-MP** 2, 3
5. $\mathbf{beg} \to \mathbf{G}p$ **HS** 1, 4
6. p **R3-begG** 5 ■

It is important to stress that representing anchored induction using **IND** is a necessity since this rule cannot be rephrased as an axiom scheme. As remarked by Kröger [13], adopting a similar scheme would trivialise the whole logic. In [18], this is overcome as above, but considering that **beg** is definable in terms of past time connectives. In TLA, **beg** has no logical counterpart, but canonical specifications are expected to define an initialisation condition.

We choose to axiomatise our branching-time modality as follows:

(A13-A) $\mathrel{|\!\sim}_{\Delta} \mathbf{A}(p \to q) \to (\mathbf{A}p \to \mathbf{A}q)$;

(A14-A) $\mathrel{|\!\sim}_{\Delta} \mathbf{A}p \to p$;

(A15-EA) $\mathrel{|\!\sim}_{\Delta} \mathbf{E}p \to \mathbf{AE}p$;

(A16-EV) $\mathrel{|\!\sim}_{\Delta} (\mathbf{E}p)\mathbf{V}q \to \mathbf{E}(p\mathbf{V}q)$;

(A17-AV) $\mathrel{|\!\sim}_{\Delta} \mathbf{A}(p \to \mathbf{X}(q\mathbf{U}p)) \to (p \to \mathbf{XA}(q\mathbf{U}p))$;

(A18-Ebeg) $\mathrel{|\!\sim}_{\Delta} \mathbf{E}(\mathbf{beg}) \to \mathbf{beg}$;

(R4-A) $\{p\} \mathrel{|\!\sim}_{\Delta} \mathbf{A}p$.

The axiom schemes above define a full branching-time modality, in the sense that there is no restriction on using **A** in any context. Axiom schemes **A13-15** and rule **R4-A** for modal generalisation, although slightly different from the usual formulation, make of **A** an S5 modality. Scheme **A18** says that the possibility of the current moment being initial forces it to be the case, meaning that all behaviours are at first synchronised, i.e., the level of their initial worlds is the same. **A16** requires in addition that a behaviour possesses an alternative in a future moment only if its subsequent history up to but not including that point could also be realised by the alternative behaviour. Schema **A17** means that our notion of possibility is relatively invariant, not becoming more and more restrictive with the passing of time in selecting possible alternatives to the current world.

As remarked above, first-order logic is required in representing message passing systems faithfully. We assume that our logical system is equipped with the usual *Free* variable function, writing $p \in E(\Xi_\Delta)$ as $p(x, y)$ to stress whenever $\{x, y\} \subseteq \mathit{Free}(p)$. We also assume the existence of a map $\{\cdot\}$ associating each Δ with a *substitution relation* $\{\cdot\}_\Delta \subseteq (E(\Xi_\Delta) \times E(\Xi_\Delta) \times E(\Xi_\Delta)) \times E(\Xi_\Delta)$. For $\{p, q, r\} \subseteq E(\Xi_\Delta)$, the set $\{s \in E(\Xi_\Delta) | (p, q, r)\{\cdot\}_\Delta(s)\}$ represented by its generic element $p\{q\backslash r\}$ denoting some substitution of q for r in p is defined in a straightforward way omitted here. The usual *substitution function* $[\cdot]_\Delta : F(\Xi_\Delta) \times \mathcal{V} \times T(\Xi_\Delta) \to F(\Xi_\Delta)$ becomes a fixed point of this substitution relation (in which all the possible substitutions have already been made): $p[x\backslash t] = q$ iff $q \in p\{x\backslash t\}$ and $q\{x\backslash t\} = \{q\}$. We say that r is free for q in p if and only if $\mathit{Free}(r) \subseteq \mathit{Free}(p\{q\backslash r\})$. As usual, we only allow a substitution $p\{q\backslash r\}$ to be effected if r is free for q in p. With these definitions at hand, we can extend our axiomatisation to capture the first-order fragment as follows:

(A19-$\forall$) $|\!\sim_\Delta \forall x \cdot p(x) \to p[x\backslash t]$;

(A20-$\forall$) $|\!\sim_\Delta \forall x \cdot (p \to q(x)) \to (p \to \forall x \cdot q(x))$, provided that $x \notin \mathit{Free}(p)$;

(A21-EQ) $|\!\sim_\Delta t = t$;

(A22-EQ) $|\!\sim_\Delta t_1 = t_2 \to (a\{t\backslash t_1\} \to a\{t\backslash t_2\})$, provided that a is an atomic formula;

(R5-$\forall$) $\{p \to q\} |\!\sim_\Delta p \to \forall x \cdot q(x)$, provided that $x \notin \mathit{Free}(p)$.

The axiomatisation above is standard. The only exception is perhaps the use of the substitution relation in **A22-EQ** to allow the proof of theorems like $x = y \to (f(x) = y \to f(y) = x)$, $f \in \Omega_\Delta$, which cannot be derived based only on the substitution function. In [21], the same notion is adopted.

To conclude our axiomatisation, it remains to show how the other connectives interact with equality and the first-order quantifiers. This is defined as follows, provided that t is free for x in p and each t_i does not contain attribute symbols:

(A23-$\exists$V) $|\!\sim_\Delta (\exists x \cdot p(x))\mathbf{V}q \to \exists x \cdot (p(x)\mathbf{V}q)$;

(A24-EQG) $|\!\sim_\Delta t_1 = t_2 \to \mathbf{G}(t_1 = t_2)$;

(A25-NEQG) $|\!\sim_\Delta t_1 \neq t_2 \to \mathbf{G}(t_1 \neq t_2)$;

(A26-$\forall$A) $|\!\sim_\Delta \forall x \cdot \mathbf{A}(p(x)) \to \mathbf{A}(\forall x \cdot p(x))$;

(A27-EQA) $|\!\sim_\Delta t_1 = t_2 \to \mathbf{A}(t_1 = t_2)$;

(A28-NEQG) $|\!\sim_\Delta t_1 \neq t_2 \to \mathbf{A}(t_1 \neq t_2)$.

A23 is a Barcan formula saying that quantification domains do not vary with the passing of time. It entails the more conventional $\forall x \cdot \mathbf{G}p(x) \to \mathbf{G}(\forall x \cdot p(x))$. Note its similarity with **A16**, although in that case the converse is not valid. **A24-5** mean that we are adopting as rigid those terms which do not include attribute symbols. Because of the side condition in these schemes, we loose the substitutivity property which would allow us to substitute formulas by logically equivalent ones in any context. **A26-8** play roles similar to **A23-5** with respect to the branching-time modality.

To conclude our proof-theoretic studies, we state without proof that **REPL**, a replacement rule, is derivable. This is verified by structural induction on the logical language based on the monotonicity of our connectives. For **A**, this means that $\vdash_{\Delta} \mathbf{A}(p \to q) \to (\mathbf{A}p \to \mathbf{A}q)$. Although applicable only to the fragment of the logic without flexible symbols, this rule is often useful in practice:

Proposition 2.11 (Replacement Rule) Given a signature Δ, the following inference rule for any $\{p, q\} \cup \{x, y\} \subseteq F(\Delta)$ not containing flexible symbols is derivable: **(REPL)** $\{x \leftrightarrow y\} \mathrel{|\!\!\sim}_{\Delta} p[q\backslash x] \leftrightarrow p[q\backslash y]$. ■

Having provided a rigorous definition for the model and proof theories of our logical system, we can now assess some of its features:

Theorem 2.12 (Strong Soundness) $\Psi \vdash_{\Delta} p$ implies $\Psi \models_{\Delta} p$.

This is verified in the usual way, based on the notion of satisfaction, by ensuring that each logical axiom is universally valid and each inference rule preserves validity, meaning that valid premises imply valid conclusions. An additional structural induction argument on our Hilbert-style proofs guarantees that each entailment preserves validity, i.e. $\Psi \vdash_{\Delta} p$ implies $\Psi \models_{\Delta} p$ for any set of sentences $\Psi \cup \{p\}$. The complete proof appears in [8]. ■

Concerning completeness, it is not difficult to see that, because it is possible to code Peano arithmetic in a theory, our logical system is incomplete [14,19]. Still, it may be possible to find similar interpretation structures to recover completeness, perhaps along the lines suggested in [3]. Before proceeding with the study of our first-order framework, it appears to be necessary to determine whether or not the propositional fragment of the logic is medium complete, meaning that the converse of the statement above holds for any finite Ψ. We already know that such fragment is not compact and therefore strong completeness fails again. Research in this direction is under way.

2.3 Structuring concepts

We adopt a stepwise development process inspired by research on the theory of abstract data types [16]. Abstract specifications are gradually refined until a concrete implementation is produced. Open distributed system specifications are structured in terms of temporal theory presentations, which are defined as follows:

Definition 2.13 (Theory presentation) A *theory presentation* is a pair $\Phi = (\Delta, \Psi)$, where Δ is a signature and Ψ is a finite set of Δ-formulae (the *presentation axioms*).

Defining open distributed systems using theory presentations, their structure and relationships can be respectively captured using particular signature and theory morphisms, categorical notions which are introduced below:

Definition 2.14 (Signature Morphism) Given two signatures $\Delta_1 = (\Sigma_1, \mathcal{A}_1, \Gamma_1)$ and $\Delta_2 = (\Sigma_2, \mathcal{A}_2, \Gamma_2)$, a *signature morphism* $\tau : \Delta_1 \to \Delta_2$ consists of:

- a morphism of algebraic structures $\tau_v : \Sigma_1 \to \Sigma_2$, i.e. for each $s \in S_{\Delta_1}$ there is a unique $\tau_v(s) \in S_{\Delta_2}$ and for each $f \in \Omega_{\Delta_1}$ such that $type(f) = \langle s_1, \ldots, s_n \rangle \to s$ there is a unique $\tau_v(g) \in \Omega_{\Delta_2}$ such that $type(\tau_v(g)) = \langle \tau_v(s_1), \ldots, \tau_v(s_n) \rangle \to \tau_v(s)$;

- a morphism of set-theoretic nature $\tau_\alpha : \mathcal{A}_1 \to \mathcal{A}_2$, i.e. for each $g \in \mathcal{A}_1$ such that $type(g) = \langle s_1, \ldots, s_n\rangle \to s$ there is a unique $\tau_\alpha(g) \in \mathcal{A}_2$ such that $type(\tau_\alpha(g)) = \langle \tau_v(s_1), \ldots, \tau_v(s_n)\rangle \to \tau_v(s)$;
- a morphism of set-theoretic nature $\tau_\gamma : \Gamma_1 \to \Gamma_2$, i.e. for each $c \in \Gamma_1$ such that $type(c) = \langle s_1,\ldots, s_n\rangle$, there is a unique $\tau_\gamma(c) \in \Gamma_2$ such that $type(\tau_\gamma(c)) = \langle \tau_v(s_1),\ldots, \tau_v(s_n)\rangle$.

Signature morphisms translate the symbols of a signature into those of another one. It is straightforward to extend this notion to provide compositional definitions for the translation of classifications, terms, formulae and sets thereof under a given signature morphism. These are omitted here.

Definition 2.15 (Presentation Morphism) Given two presentations $\Phi_1 = (\Delta_1, \Psi_1)$ and $\Phi_2 = (\Delta_2, \Psi_2)$, a *presentation morphism* $\tau : \Phi_1 \to \Phi_2$ is a signature morphism lifted to formulae such that $\Psi_2 \vdash_{\Delta_2} \tau(p)$ for every $p \in \Psi_1$ (preservation of consequences).

The constraint requiring the preservation of consequences of each presentation axiom, based on the logical consequence relation $\vdash$, forbids that some of these consequences be forgotten when a presentation is considered as part of another one, allowing the definition of composable software artifacts. This is best captured by the following result:

Proposition 2.16 (Categories of Signatures and Presentations) The collections of signatures and theory presentations, together with the corresponding morphisms, define co-complete categories **Sig** and **Pres** respectively. ■

3 Example: The Drinking Philosophers Problem

In this section, we study the drinking philosophers problem [6], a generalisation of resource allocation problems in distributed systems. A group of philosophers is gathered around a table to drink from some bottles of beverage. Each philosopher may be tranquil, thirsty or drinking. Depending on the bottles he shares with his neighbours and on the desired beverages, a philosopher may generate conflicts, since the usage of bottles is considered to be mutually exclusive. Any solution of this problem must guarantee symmetry, that philosophers must all obey the same rules, and progress, that thirsty philosophers eventually drink the desired beverages, among other less demanding requirements.

A better known but restricted formulation of resource allocation is the dining philosophers problem, in which pairs of neighbours share a single chopstick. Due to their access to this shared resource, such philosophers necessarily have mutually exclusive actions. As in the drinkers case, any solution must guarantee symmetry and progress, but the behaviour of each philosopher cycles through a series of distinghished states, from thinking to hungry to eating. A modular solution of a simplified version of this problem in terms of temporal theory presentations was presented in [10]. Therein, channels are used to guarantee the mutually exclusive usage of chopsticks (actually forks). Moreover, the local progress assumptions that each philosopher eventually becomes hungry and later raises his chopsticks is used in the derivation of a global property ensuring that they recurrently cycle between eating and thinking states. This theorem is not straigtforward to verify, since individual philosophers are not required to lower their chopsticks, but this property emerges when they are put together in the appropriate manner to form a system. By the way, to ensure the eventual lowering of chopsticks in the local context of single philosophers, some additional assumptions have to be made, as outlined in [4], stating not only the willingness of each philosopher to do so whenever eating but also a strong fairness assumption called politeness, that requires the eventual occurrence of this action whenever it is always eventually possible.

We use the complete solution of the diners problem outlined above (based on [10] and [4]) as part of the implementation of our drinkers solution. That is, we propose a solution in which philosophers can simultaneously drink and eat, this last activity being performed just due to an implementation decision, with philosophers having local attributes to record their current state in each of these processes. As outlined in [6], this strategy is normally adopted to establish a distinction between philosophers in resolving conflicts, by considering that eating philosophers always receive priority in drinking conflicts. In our case, we also use this strategy to guarantee progress in our solution. Symmetry is ensured by requiring that all philosophers comply with the same specification.

The specification, configuration and verification of the proposed solution of the drinking philosophers problem is presented in the Appendix. The main result presented therein is the proof of correctness of our solution, which is summarised by the provability of the following theorem, formulated in the context of a theory presentation D-Phil, which describes drinking philosophers:

$$\vdash \ \mathit{thirsty}(x) = \mathrm{TRUE} \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{TRUE}) \tag{1}$$

This example is very illustrative as a means of justifying the features of our logical system. Sentence (1) is of first-order nature and the entire problem is formulated using this framework because, differently from the diners problem, the number of beverages that philosophers can share and drink in each situation is finite but *a priori* undetermined. Moreover, the verification of (1) can only be performed in a local context by relying on the branching-time assumptions of willingness (2) and politeness (3), stated among the axioms of Phil, describing dining philosophers:

$$\mathit{eating} = \mathrm{TRUE} \rightarrow \mathbf{FE}(\mathit{chops} \downarrow) \tag{2}$$

$$\mathbf{GFE}(\mathit{chops} \downarrow) \rightarrow \mathbf{F}(\mathit{chops} \downarrow) \tag{3}$$

4 Final Remarks

We have presented a new first-order many-sorted branching-time logical system with equality devoted to support the rigorous development of open distributed systems. Many other logical systems exist with similar features and could perhaps be used for the same purpose. These are analysed below.

Among first-order formalisms, our system keeps many similarities with those developed by Lamport (TLA) [15] and by Manna and Pnueli [17]. Those systems consider the existence of external enabledness predicates (*Enabled* and *En*, respectively), analogous to our $\mathbf{E}$, which have their statement and/or proof justified by the possibility of occurrence of some computations (for TLA, a syntactic definition of *Enabled* was indeed provided recently in [1]). There are some minor differences in relation to our system, such as the transitional definitional actions adopted in TLA, the absence of the $\mathbf{X}$ (next) connective in this system and its introduction in the second one as a term as well. However, the most relevant distinction in relation to our work seems to be the characterisation of those systems to be of linear nature, as opposed to our branching-time system.

Our logical system differs substantially from other propositional branching-time formalisms. The main distinction in relation to CTL [9], for instance, is that **E** is regarded as a full branching-time modality here, in the sense that there is no restriction on using this connective in any context, whereas in CTL it can only enclose a formula surrounded by the linear time connectives **F** or **G**. Another distinction is in relation to CTL* and other Ockhamist branching-time systems (e.g. [7,23]), which have an interpretation of **E** biased by possible future events. We believe that the adopted immediate interpretation of **E** is more faithful to the intuition of software engineers in developing open distributed systems. In fact, this interpretation was originally proposed by Zanardo [25] in the context of a propositional logical system with a different language, time frames and interpretation of extra-logical symbols. Obviously, our system differs radically from Peircean propositional temporal formalisms [11], which are defined and axiomatised in a substantially different way, without a branching-time modality.

The reported work suggests many promising directions for future research. In would be interesting to attempt to introduce continuous time or intervals in our framework. In addition, proposing an alternative categorical model theory for (at least the propositional fragment of) our logic could potentially contribute to define, in a different style, a more effective proof-theory. Finally, more experimental work remains to be developed concerning the application of our logical system in the refinement of open distributed system specifications.

References

[1] Abadi, M. and S. Merz, *On TLA as a logic*, in: M. Broy, editor, *Deductive Program Design*, NATO ASI Series, Springer-Verlag, 1996 pp. 235–271.

[2] Alpern, B. and F. B. Schneider, *Defining liveness*, Information Processing Letters **21** (1985), pp. 181–185.

[3] Andréka, H. et al., *Effective temporal logics of programs*, in: L. Bolc and A. Szałas, editors, *Time and Logic: A Computational Approach*, UCL Press, 1995 pp. 51–129.

[4] Barreiro, N., J. Fiadeiro and T. Maibaum, *Politeness in object societies*, in: R. Wieringa and R. Feenstra, editors, *Information Systems: Correctness and Reusability* (1995), pp. 119–134.

[5] Barringer, H., *The use of temporal logic in the compositional specification of concurrent systems*, in: A. Galton, editor, *Temporal Logics and their applications* (1987), pp. 53–90.

[6] Chandy, K. M. and J. Misra, *The drinking philosophers problem*, ACM Transactions on Programming Languages and Systems **6** (1984), pp. 632–646.

[7] do Carmo, J. and A. Sernadas, *Branching versus linear logics yet again*, Formal Aspects of Computing **2** (1990), pp. 24–59.

[8] Duarte, C. H. C., "Proof-Theoretic Foundations for the Design of Extensible Software Systems," Ph.D. thesis, Department of Computing, Imperial College, London, UK (1998).

[9] Emerson, E. A., *Temporal and modal logic*, in: J. van Leeuwen, editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, Elsevier, 1990 pp. 996–1072.

[10] Fiadeiro, J. and T. Maibaum, *Temporal theories as modularisation units for concurrent systems specification*, Formal Aspects of Computing **4** (1992), pp. 239–272.

[11] Gabbay, D., I. Hodkinson and M. Reynolds, "Volume I," Temporal Logic: Mathematical Foundations and Computational Aspects, Oxford Science Publications, 1994.

[12] Koymans, R., *Specifying message passing systems requires extending temporal logic*, in: *6th ACM Symposium on Principles of Distributed Computing* (1987), pp. 191–204.

[13] Kröger, F., "Temporal Logic of Programs," EATCS Monographs on Theoretical Computer Science **8**, Springer-Verlag, 1987.

[14] Kröger, F., *On the interpretability of arithmetic in temporal logic*, Theoretical Computer Science **73** (1990), pp. 47–60.

[15] Lamport, L., *The temporal logic of actions*, ACM Transactions on Programming Languages and Systems **16** (1994), pp. 872–923.

[16] Maibaum, T., P. A. S. Veloso and M. Sadler, *A theory of abstract data types for program development: Bridging the gap?*, in: H. Ehrig et al., editors, *Proc. International Conference on Theory and Practice of Software Development (TAPSOFT'85), vol. II*, Lecture Notes in Computer Science **185** (1985), pp. 214–230.

[17] Manna, Z. and A. Pnueli, *How to cook a temporal proof system for your pet language*, in: *Proc. 10th Symposium on Principles of Programming Languages* (1983), pp. 141–154.

[18] Manna, Z. and A. Pnueli, *The anchored version of the temporal framework*, in: J. W. de Bakker, W.-P. de Roever and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science **354** (1989), pp. 200–284.

[19] Merz, S., *Decidability and incompleteness results for first-order temporal logics of linear time*, Journal of Applied Non-Classical Logic **2** (1992).

[20] Pnueli, A., *The temporal logic of programs*, in: *Proc. 18th Symposium of Foundations of Computer Science* (1977), pp. 45–57.

[21] Sernadas, A., C. Sernadas and J. F. Costa, *Object specification logic*, Journal of Logic and Computation **5** (1995), pp. 603–630.

[22] Sistla, A. P., E. M. Clarke, N. Francez and A. R. Meyer, *Can message buffers be axiomatised in linear temporal logic?*, Information and Computation **63** (1984), pp. 88–112.

[23] Stirling, C., *Modal and temporal logics*, in: S. Abramsky, D. Gabbay and T. Maibaum, editors, *Volume II*, Handbook of Logic in Computer Science, Oxford Science Publications, 1992 pp. 477–563.

[24] van Benthem, J., "The Logic of Time: A Model Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse," Synthese Library **156**, Kluwer Academic Publishers, 1991, 2nd edition.

[25] Zanardo, A., *A complete deductive system for since-until branching time logic*, Journal of Philosophical Logic **20** (1991), pp. 131–148.

[26] Zanardo, A., *Branching-time logic with quantification over branches: The point of view of modal logic*, Journal of Symbolic Logic **61** (1996), pp. 1–39.

Appendix: Drinking Philosophers Problem Detailed Solution

Here we present in greater detail our solution of the drinking philosophers problem. This solution is modularly organised in terms of theory presentations, which specify the local language and behaviour of each object, and presentation morphisms, which determine the configuration of the specified system.

The vocabulary of drinking philosophers (signature) is defined as part of the specification D-Phil, presented in Figure 1. We assume the existence of two sorts comprising beverages and booleans (described in separate omitted presentations). The state of each philosopher is determined by the value of the attributes *thirsty* and *drinking*, which take beverages as arguments and return boolean values representing respectively whether or not a philosopher is thirsty for or drinking a specific beverage. Philosophers can only perform one out of three actions related to each specific beverage, namely to become thirsty (*bc_thirsty*), to drink and stop drinking ($bottle \uparrow / \ bottle \downarrow$) from a specific bottle.

The behaviour of each philosopher is ruled by the axioms in specification D-Phil. Axiom (A.1.1) states that the effect of becoming thirsty is a change of value in the attribute *thirsty*. Axioms (A.1.2) and (A.1.3) define similar effects concerning the occurrence of $bottle \uparrow$ and $bottle \downarrow$. The next two axioms, (A.1.4) and (A.1.5), specify locality properties stating that attribute values change only when the appropriate philosopher actions take place. These axioms are required here because, differently from [10], we do not adopt locality as a logical property. The subsequent three axioms concern the permission of occurrence of each action: a philosopher may become thirsty for a beverage only if not already so (A.1.6); a bottle can be used from drinking only if the philosopher is thirsty for the respective beverage and is not drinking (A.1.7); and a bottle can be discarded only if the philosopher is currently using it for drinking (A.1.8). In addition to this, there is a set of axioms dealing with the local temporal behaviour of philosophers. In the beginning, philosophers are not drinking, according to (A.1.9); in each occasion, all the required bottles are used or dropped simultaneously, due to (A.1.10) and (A.1.11); and a philosopher eventually becomes thirsty for each beverage (A.1.12). Finally, we make some local assumptions concerning how these objects loosely interact with their surrounding environments. Axiom (A.1.13) states a willingness property of each philosopher of possibily disposing of his bottles eventually whenever drinking. Axiom (A.1.14) is a fairness requirement, normally called politeness [4], stating that bottles should be eventually dropped if it is always eventually possible to do so.

As previously mentioned, we adopt a dining philosophers solution as part of the implementation of our drinkers problem solution. The proposed design structure is rather standard, consisting in a presentation to represent the problem domain, D-Phil detailed above, another one to capture implementation details, Phil partially described in [10], and a third specification to determine how the problem and the implementation are related, X-Phil presented in Figure 2. In categorical terms, we define the following diagram D-Phil → X-Phil ← Phil to capture this structure.

The specification X-Phil is rather simple, reflecting its unique purpose of relating problem and implementation. Because Phil and D-Phil are imported in X-Phil without collapsing any of their symbols, we obtain as a result the desired composition of dining and drinking states in the resulting presentation. The respective actions, however, are not completely independent, due to the existence of axiom (A.2.1) obliging the simultaneous

Specification D-Phil

sorts bev(erage), bool(ean)

constants T(RUE), F(ALSE) : bool

attributes *thirsty*, *drinking* : bev → bool

actions *bc_thirsty*(bev), *bottle* ↑ (bev), *bottle* ↓ (bev)

axioms x, y : bev; v : bool

$$\mathit{bc\_thirsty}(x) \rightarrow \mathbf{X}(\mathit{thirsty}(x) = \mathrm{T}) \tag{A.1.1}$$

$$\mathit{bottle}\uparrow(x) \rightarrow \mathbf{X}(\mathit{drinking}(x) = \mathrm{T}) \tag{A.1.2}$$

$$\mathit{bottle}\downarrow(x) \rightarrow \mathbf{X}(\mathit{drinking}(x) = \mathrm{F} \wedge \mathit{thirsty}(x) = \mathrm{F}) \tag{A.1.3}$$

$$\mathit{bc\_thirsty}(x) \vee \mathit{bottle}\downarrow(x) \vee (\mathit{thirsty}(x) = v \wedge \mathbf{X}(\mathit{thirsty}(x) = v)) \tag{A.1.4}$$

$$\mathit{bottle}\uparrow(x) \vee \mathit{bottle}\downarrow(x) \vee (\mathit{drinking}(x) = v \wedge \mathbf{X}(\mathit{drinking}(x) = v)) \tag{A.1.5}$$

$$\mathit{bc\_thirsty}(x) \rightarrow \mathit{thirsty}(x) = \mathrm{F} \tag{A.1.6}$$

$$\mathit{bottle}\uparrow(x) \rightarrow \mathit{thirsty}(x) = \mathrm{T} \wedge \nexists y \cdot \mathit{drinking}(y) = \mathrm{T} \tag{A.1.7}$$

$$\mathit{bottle}\downarrow(x) \rightarrow \mathit{drinking}(x) = \mathrm{T} \tag{A.1.8}$$

$$\mathbf{beg} \rightarrow \mathit{drinking}(x) = \mathrm{F} \tag{A.1.9}$$

$$\mathit{thirsty}(x) = \mathrm{T} \wedge \mathit{thirsty}(y) = \mathrm{T} \rightarrow (\mathit{bottle}\uparrow(x) \leftrightarrow \mathit{bottle}\uparrow(y)) \tag{A.1.10}$$

$$\mathit{drinking}(x) = \mathrm{T} \wedge \mathit{drinking}(y) = \mathrm{T} \rightarrow (\mathit{bottle}\downarrow(x) \leftrightarrow \mathit{bottle}\downarrow(y)) \tag{A.1.11}$$

$$\mathit{thirsty}(x) = \mathrm{F} \rightarrow \mathbf{F}(\mathit{bc\_thirsty}(x)) \tag{A.1.12}$$

$$\mathit{drinking}(x) = \mathrm{T} \rightarrow \mathbf{FE}(\mathit{bottle}\downarrow(x)) \tag{A.1.13}$$

$$\mathbf{GFE}(\mathit{bottle}\downarrow(x)) \rightarrow \mathbf{F}(\mathit{bottle}\downarrow(x)) \tag{A.1.14}$$

End

Fig. 1. Specification of drinking philosophers.

Specification X-Phil

imports Phil $[\mathit{forks}\uparrow \mapsto \mathit{chops}\uparrow, \mathit{forks}\downarrow \mapsto \mathit{chops}\downarrow]$

imports D-Phil

axioms x : bev

$$\mathit{eating} = \mathrm{T} \wedge \mathit{thirsty}(x) = \mathrm{T} \wedge \mathit{drinking}(x) = \mathrm{F} \rightarrow (\mathit{chops}\downarrow \leftrightarrow \mathit{bottle}\uparrow(x)) \tag{A.2.1}$$

End

Fig. 2. Implementation of drinking philosophers.

occurrence of the lowering of chopsticks and the grabbing of some bottles whenever the philosopher is eating and thirsty. This is how we effectively connect the solution of the diners problem to our solution. In addition, this seems to be a minimalist set of assumptions to relate both problems in the desired way.

What remains to be described about the configuration of our system regards the disposal of philosophers and beverages around a table. We assume that each pair of

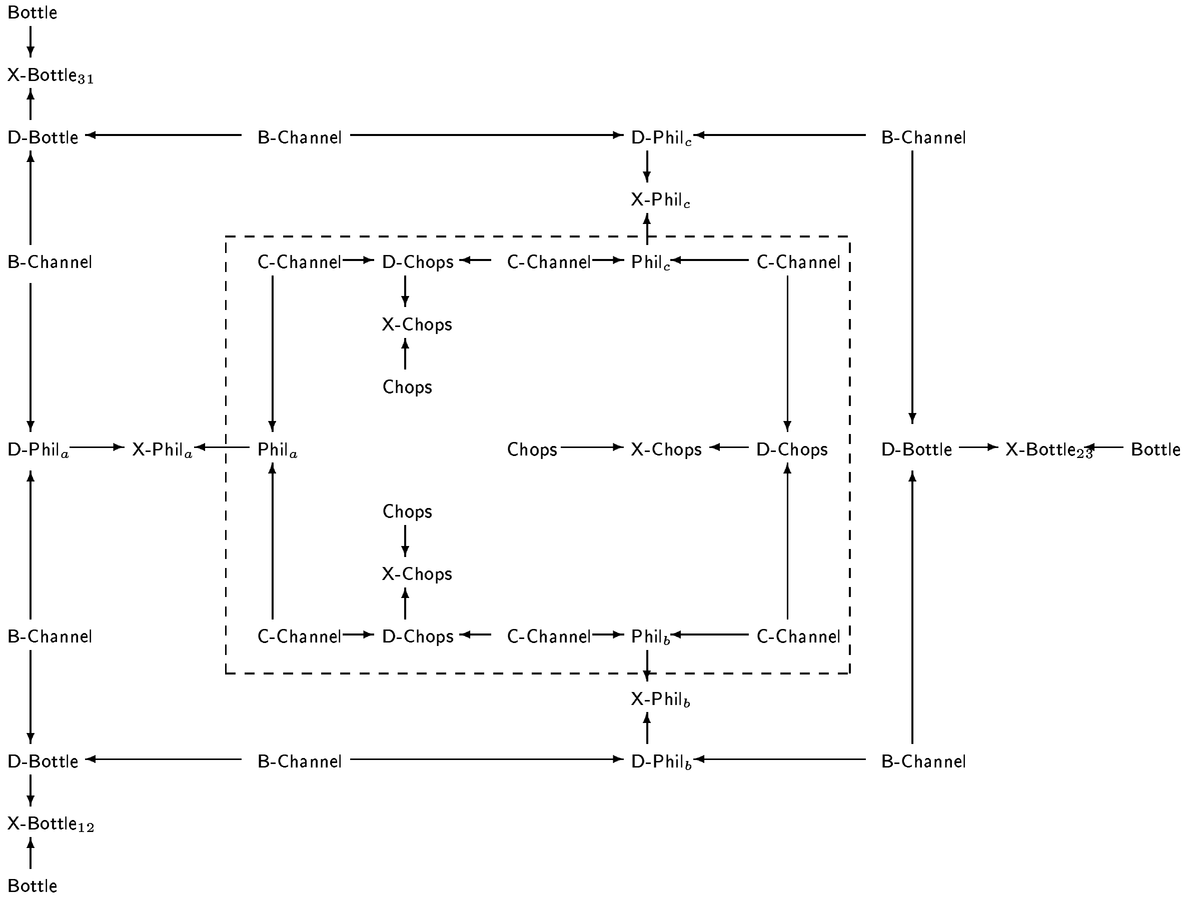

Fig. 3. Configuration diagram for three distinct philosophers and beverages.

neighbours not only shares a set of beverage bottles but also a chopstick. The definition of chopsticks (Chop) and their mutually exclusive usage through channels (C-Channel) is detailed in [10] and omitted here. The construction of sets of bottles (Bottle) and their sharing through specific channels (B-Channel) is performed in an analogous manner, which is also omitted. Specifically, these constructions follow the same design structure described above to implement drinkers in terms of diners, with resources such as chopsticks and sets of bottles playing the role of implementation details and the respective interfaces for ensuring synchronised usage describing the problem domain (D-Chop and D-Bottle respectivelly).

In order to make concrete use of the definitions above, we first assume the existence around a table of three philosophers — akira (a), shiba (b) and joshua (c) — drinking three different kinds of beverages — sake (1), shochu (2) and mirin (3). This configuration is formalised by the diagram in Figure 3. Note that we make full use of the configuration proposed in [10], particularised to three philosophers only, surrounding the respective diagram with dashed lines in the figure. Since the philosophers are represented by instances of temporal presentations, no confusion arises between them, but we have to adopt additional assumptions concerning beverages, not only to ensure their existence but also to distinguish them from each other through appropriate axioms in the respective purely first-order theory presentation. In addition, we are obliged to particularise the interfaces describing bottle sets (X-Bottle), in order to prevent the occurrence of actions concerning beverages which are not shared between philosophers. All these details are marked through indices in the diagram. Computing the co-limit of this diagram, by putting all these descriptions together in a minimalist presentation called SYSTEM in the way suggested by the morphisms (collapsing shared symbols whenever required), we obtain a specification of the whole resulting system.

Having defined our solution, we are left with the task of verifying the main correctness property for the problem, namely that in the configuration above thirsty philosophers drink the desired beverages eventually. We prove first two helpful properties suggested in [6], based on specification axioms (Ax) and theorem proofs (Th) developed in [10,4]:

A. A thinking-thirsty philosopher becomes hungry or drinking:

1.	$hungry = \mathrm{F} \rightarrow \mathbf{F}(bc\_hungry)$	Ax PHIL
2.	$hungry = \mathrm{F} \rightarrow \mathbf{F}(bc\_hungry) \vee \mathbf{F}(bottle \uparrow (x))$	**OR-R** 1
3.	$hungry = \mathrm{F} \rightarrow \mathbf{F}(bc\_hungry \vee bottle \uparrow (x))$	**DIST-ORF**, **HS** 2
4.	$hungry = \mathrm{F} \wedge thirsty(x) = \mathrm{T} \wedge drinking(x) = \mathrm{F} \rightarrow$ $\mathbf{F}(bc\_hungry \vee bottle \uparrow (x))$	**AND-L** 3

B. Philosophers (recurrently) lower their chopsticks:

1.	$\mathbf{F}(eating = \mathrm{T})$	Th PHIL
2.	$\mathbf{F}(eating = \mathrm{T}) \rightarrow \mathbf{FFE}(chops \downarrow)$	**MON-GF**, **R2-G** $Ax_{willingness}$ PHIL, **R1-MP**
3.	$\mathbf{F}(eating = \mathrm{T}) \rightarrow \mathbf{FE}(chops \downarrow)$	**IDEM-F**, **HS** 2
4.	$\mathbf{FE}(chops \downarrow)$	**R1-MP** 1, 2
5.	$\mathbf{F}(chops \downarrow)$	**R2-G** 4, **R1-MP** $Ax_{politeness}$ PHIL

At this point in the proof, we rely on three lemmas that reflect the possible state transitions performed by philosophers connected to a drinking-eating table. The following definitions characterise the source states of each of these transitions:

$$d \overset{\text{def}}{=} hungry = \mathrm{F} \wedge thirsty(x) = \mathrm{T} \wedge drinking(x) = \mathrm{F}$$

$$e \overset{\text{def}}{=} hungry = \mathrm{T} \wedge thirsty(x) = \mathrm{T} \wedge drinking(x) = \mathrm{F}$$

$$f \overset{\text{def}}{=} eating = \mathrm{T} \wedge thirsty(x) = \mathrm{T} \wedge drinking(x) = \mathrm{F}$$

The verification of the aforementioned lemmas can be performed based on a common argument which shows the actions of philosophers happening in a specific order. We develop below a derivation for the first case, which is later reused in each required proof:

C. Allowed actions in a thinking-thirsty state:

1.	$hungry = \mathrm{F} \rightarrow$ $(hungry = \mathrm{F})\mathbf{W}(hungry = \mathrm{F} \wedge (bc\_hungry \vee chops \downarrow))$	from $Ax\_locality$ PHIL
2.	$hungry = \mathrm{F} \rightarrow \neg chops \downarrow$	Ax PHIL, **CONP**, **HS**
3.	$hungry = \mathrm{F} \rightarrow (hungry = \mathrm{F})\mathbf{W}(hungry = \mathrm{F} \wedge bc\_hungry)$	1, 2 $\mid\!\sim^{MP}$
4.	$thirsty(x) = \mathrm{T} \rightarrow$ $(thirsty(x) = \mathrm{T})\mathbf{W}(thirsty(x) = \mathrm{T} \wedge (bc\_thirsty \vee bottle \downarrow (x)))$	from (A.1.4)
5.	$thirsty(x) = \mathrm{T} \rightarrow \neg bc\_thirsty$	(A.1.6), **CONP**, **HS**
6.	$thirsty(x) = \mathrm{T} \rightarrow (thirsty(x) = \mathrm{T})\mathbf{W}(thirsty(x) = \mathrm{T} \wedge bottle \downarrow (x))$	4, 5 $\mid\!\sim^{MP}$
7.	$drinking(x) = \mathrm{F} \rightarrow$ $(drinking(x) = \mathrm{F})\mathbf{W}(drinking(x) = \mathrm{F} \wedge (bottle \uparrow (x) \vee bottle \downarrow (x)))$	from (A.1.5)
8.	$drinking(x) = \mathrm{F} \rightarrow \neg bottle \downarrow (x)$	(A.1.8), **CONP**, **HS**
9.	$drinking(x) = \mathrm{F} \rightarrow (drinking(x) = \mathrm{F})\mathbf{W}(drinking(x) = \mathrm{F} \wedge bottle \uparrow (x))$	7, 8 $\mid\!\sim^{MP}$
10.	$d \rightarrow (d)\mathbf{W}(d \wedge (bc\_hungry \vee bottle \downarrow (x) \vee bottle \uparrow (x)))$	3, 6, 9 $\mid\!\sim^{MP}$

Now we develop the proof of each lemma, based on our specification axioms and previous results, where the derivation above is referred to as $\Pi_{\mathbf{C}}$:

D. A thinking-thirsty philosopher becomes hungry-thirsty or drinking:

1.	$\mathit{drinking}(x) = \mathrm{F} \rightarrow \neg \mathit{bottle} \downarrow (x)$	(A.1.8), **CONP**, **HS**
2.	$d \rightarrow (d)\mathbf{W}(d \wedge (\mathit{bc\_hungry} \vee \mathit{bottle} \uparrow (x)))$	1, $\Pi_{\mathbf{C}} \mid\!\sim^{MP}$
3.	$d \rightarrow \mathbf{F}(d \wedge (\mathit{bc\_hungry} \vee \mathit{bottle} \uparrow (x)))$	**A.**, 2 $\mid\!\sim^{MP}$
4.	$d \wedge \mathit{bc\_hungry} \wedge \neg \mathit{bottle} \uparrow (x) \rightarrow \mathbf{F}(e)$	*Ax* PHIL **RPL-XF**, **HS**, (A.1.6), (A.1.8)
5.	$\mathit{bottle} \uparrow (x) \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	(A.1.2), **RPL-XF**, **HS**
6.	$d \rightarrow \mathbf{F}(e \vee \mathit{drinking}(x) = \mathrm{T})$	**AND-I** 4, 5; **TRAN-F**, **R1-MP**, **HS** 3

E. A hungry-thirsty philosopher becomes eating-thirsty or drinking:

$\{\ \Pi^* \overset{\mathrm{def}}{=} \Pi_{\mathbf{C}}[d \mapsto e, \mathit{hungry} = \mathrm{F} \mapsto \mathit{hungry} = \mathrm{T}, \mathit{bc\_hungry} \mapsto \mathit{chops} \uparrow]\}$

1.	$\mathit{drinking}(x) = \mathrm{F} \rightarrow \neg \mathit{bottle} \downarrow (x)$	(A.1.8), **CONP**, **HS**
2.	$e \rightarrow (e)\mathbf{W}(e \wedge (\mathit{chops} \uparrow \vee \mathit{bottle} \uparrow (x)))$	1, $\Pi^* \mid\!\sim^{MP}$
3.	$\mathit{hungry} = \mathrm{T} \rightarrow \mathbf{F}(\mathit{chops} \uparrow)$	*Ax* PHIL
4.	$e \rightarrow \mathbf{F}(\mathit{chops} \uparrow \vee \mathit{bottle} \uparrow (x))$	**AND-L** 3, **OR-R**, **DIST-ORF**, **HS**
5.	$e \rightarrow \mathbf{F}(e \wedge (\mathit{chops} \uparrow \vee \mathit{bottle} \uparrow (x)))$	4, 2 $\mid\!\sim^{MP}$
6.	$e \wedge \mathit{chops} \uparrow \wedge \neg \mathit{bottle} \uparrow (x) \rightarrow \mathbf{F}(f)$	*Ax* PHIL, **RPL-XF**, **HS**, (A.1.6), (A.1.8)
7.	$\mathit{bottle} \uparrow (x) \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	(A.1.2), **RPL-XF**, **HS**
8.	$e \rightarrow \mathbf{F}(f \vee \mathit{drinking}(x) = \mathrm{T})$	**AND-I** 6, 7; **TRAN-F**, **R1-MP**, **HS** 5

F. A eating-thirsty philosopher becomes thinking-drinking:

$\{\ \Pi^{**} \overset{\mathrm{def}}{=} \Pi_{\mathbf{C}}[d \mapsto f, \mathit{hungry} = \mathrm{F} \mapsto \mathit{eating} = \mathrm{T}, \mathit{bc\_hungry} \mapsto \mathit{chops} \downarrow]\}$

1.	$\mathit{drinking}(x) = \mathrm{F} \rightarrow \neg \mathit{bottle} \downarrow (x)$	(A.1.8), **CONP**, **HS**
2.	$f \rightarrow (f)\mathbf{W}(f \wedge (\mathit{chops} \downarrow \vee \mathit{bottle} \uparrow (x)))$	1, $\Pi^{**} \mid\!\sim^{MP}$
3.	$f \rightarrow \mathbf{F}(\mathit{chops} \downarrow)$	**A1-I**, **R1-MP B.**
4.	$f \rightarrow \mathbf{F}(f \wedge (\mathit{chops} \downarrow \vee \mathit{bottle} \uparrow (x)))$	3, 2 $\mid\!\sim^{MP}$
5.	$f \rightarrow \mathbf{F}(\mathit{bottle} \uparrow (x))$	(A.2.1), 4 $\mid\!\sim^{MP}$
6.	$\mathit{bottle} \uparrow (x) \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	(A.1.2), **RPL-XF**, **HS**
7.	$f \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**AND-I** 5, 6, **TRAN-F**, **R1-MP**

Connecting the three lemmas above using a double or-elimination argument, we conclude the verification of the correctness property of our problem:

1.	$f \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**F.**
2.	$\mathbf{F}(f) \rightarrow \mathbf{FF}(\mathit{drinking}(x) = \mathrm{T})$	**R2-G** 1, **MON-GF**, **R1-MP**
3.	$\mathbf{F}(f) \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**IDEM-F**, **HS** 2
4.	$e \rightarrow \mathbf{F}(f) \vee \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**DIST-ORF**, **HS E.**
5.	$e \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**A1-I**, **OR-L** 3, **HS** 4
6.	$\mathbf{F}(e) \rightarrow \mathbf{FF}(\mathit{drinking}(x) = \mathrm{T})$	**R2-G** 5, **MON-GF**, **R1-MP**
7.	$\mathbf{F}(e) \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**IDEM-F**, **HS** 6
8.	$d \rightarrow \mathbf{F}(e) \vee \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**DIST-ORF**, **HS D.**
9.	$d \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**A1-I**, **OR-L** 7, **HS** 8
10.	$\mathit{thirsty}(x) = \mathrm{T} \wedge \mathit{drinking}(x) = \mathrm{F} \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	*Th* PHIL, *Ax* bool, **OR-E** 1, 9
11.	$\mathit{thirsty}(x) = \mathrm{T} \wedge \mathit{drinking}(x) = \mathrm{T} \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	**A1-I**, **AND-R**, **DEF-F**, **HS**
12.	$\mathit{thirsty}(x) = \mathrm{T} \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$	*Ax* bool, **OR-E** 10, 11

Therefore: $\vdash_{\text{D-PHIL}} \mathit{thirsty}(x) = \mathrm{T} \rightarrow \mathbf{F}(\mathit{drinking}(x) = \mathrm{T})$ ■