MODEL CHECKING BIRTH AND DEATH*

Dino Distefano, Arend Rensink, Joost-Pieter Katoen
Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: {ddino, rensink, katoen}@cs.utwente.nl

Abstract This paper proposes Allocational Temporal Logic ($\mathcal{A}\ell\ell$TL) as a formalism to express properties concerning the dynamic allocation (birth) and de-allocation (death) of entities, such as the objects in an object-based system. The logic is interpreted on History-Dependent Automata, extended with a symbolic representation for certain cases of unbounded allocation. The paper also presents a simple imperative language with primitive statements for (de)allocation, with an operational semantics, to illustrate the kind of behaviour that can be modelled. The main contribution of the paper is a tableau-based model checking algorithm for $\mathcal{A}\ell\ell$TL, along the lines of Lichtenstein and Pnueli's algorithm for LTL.

1. Introduction

One of the aspects of computation that state-of-the-art model checking does not deal with very well is that of dynamic *allocation* and *deallocation* (birth and death) of entities. This is especially true if the number of entities is not known beforehand, or even unbounded. Though there are now calculi (such as the π-calculus [15]) that can express the generation of fresh names, as well as models (such as History-Dependent automata [16]) that can describe both the birth and the death of entities, what has been missing so far is a logic where these concepts are captured as primitives; a logic that should be as fundamental to reasoning about dynamic allocation as standard propositional logic is to reasoning about a fixed state space.

An attempt to formulate such a logic is presented in this paper. Called *allocational temporal logic* ($\mathcal{A}\ell\ell$TL), it has the following features: *(i)* Entity variables x, y, interpreted by a mapping to the entities existing (i.e., *alive*) in a given state. The interpretation is *partial*: a variable not mapped onto an existing entity stands for an entity that has *died*. *(ii)* Entity equations $x = y$ (where x, y are entity variables), asserting

*Appeared in R. A. Baeza-Yates and U. Montanari and N. Santoro. *Foundations of Information Technology in the Era of Networking and Mobile Computing*. IFIP 17th World Computer Congress - TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002), Montréal, Canada 2002. Kluwer Academic Publishers.

that x and y refer to the same entity. This cannot hold if either x or y has died; hence the entity equations express a *partial equivalence* of entity variables (symmetric and transitive, but not reflexive). *(iii)* Entity quantification $\exists x.\phi$, which holds in a given state if ϕ holds for some interpretation of x, provided that x is alive. *(iv)* A predicate x new to express that the entity referred to by x is *fresh*, i.e., newly born. In addition, $\mathcal{A}\ell\ell$TL has the standard LTL temporal operators.

The logic is interpreted over *high-level allocational Büchi automata* (HABA) which extend HD-automata [16] with a predicate for the *unboundedness* of (the number of entities in) a state, and with a (generalised) Büchi acceptance condition.

Together with the logic $\mathcal{A}\ell\ell$TL, the main contribution of this paper is that the model-checking problem for $\mathcal{A}\ell\ell$TL is shown to be decidable. In particular, we present a tableau-based model-checking algorithm that decides whether a given $\mathcal{A}\ell\ell$TL-formula holds for a given HABA. Our algorithm extends the tableau-based algorithm for LTL [14]. To the best of our knowledge, this yields the first approach to effectively model-check models with an unbounded number of entities. This is of particular interest to e.g. the verification of object-oriented systems in which the number of objects is typically not known in advance and may be unbounded. Furthermore, reasoning about (de)allocation of fresh names is relevant also in relation to *privacy* and *locality* as discussed in, e.g., [1, 5, 15].

Organisation of the paper. This paper introduces the logic (Section 2) and HABA (Section 3), as well as a simple imperative language, featuring statements for the allocation and deallocation of entities, with an operational semantics in terms of HABA (Section 4). The latter provides an intuition about the sort of behaviour that HABA can model. The main contribution of the paper is the proof of $\mathcal{A}\ell\ell$TL model checking property (Section 5). A discussion about related and future work completes the paper (Section 6). Proofs are reported in the long version of this paper [9].

2. Allocational temporal logic

Syntax. Let $LVar$ be a countable set of logical variables ranged over by x, y, z, and Ent be a countable set of entities ranged over by $e, e', e_1, \ldots$. Allocational Temporal Logic ($\mathcal{A}\ell\ell$TL) is an extension of propositional LTL [17] that allows existential quantification over logical variables that can denote entities, or may be undefined. For $x \in LVar$, the syntax of $\mathcal{A}\ell\ell$TL is defined by the following grammar:

$$\phi ::= x \ \mathsf{new} \mid x \ \mathsf{dead} \mid x = x \mid \exists x.\phi \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{X}\phi \mid \phi \,\mathsf{U}\, \phi$$

The operators have the following intuitive meaning. Formula x new holds if the entity denoted by x is new in the current state. Formula x dead holds if the entity denoted by x has died. Formula $x = y$ holds if variables x and y denote the same entity in the current state; $x = x$ is violated if x is undefined, i.e., if x does not denote any entity. $\exists x.\phi$ is valid in the current state if there exists an entity for which ϕ holds if assigned to x. X (next) and U (until) are the standard LTL operators. We denote $x \neq y$ for $\neg(x = y)$, x alive for $\neg(x \ \mathsf{dead})$, and x old for x alive $\wedge \neg(x \ \mathsf{new})$. The other boolean connectives and temporal operators F (eventually) and G (always) are standard.

Semantics. An *allocational sequence* σ is an infinite sequence of sets of entities $E_0 E_1 E_2 \cdots$ where $E_i \subseteq Ent$, for $i \in \mathbb{N}$. Let $\sigma^i = E_i E_{i+1} \cdots$. For given σ, E_i^σ

denotes the set of entities in the i-th state of σ. The semantics of $\mathcal{A}\ell\ell$TL-formulae is defined by a satisfaction relation $\sigma, N, \theta \models \phi$ where σ is an allocational sequence, $N \subseteq E_0^\sigma$ is the set of entities that is initially new, and $\theta : LVar \rightharpoonup Ent$ is a partial valuation of the free variables in ϕ. Let N_i^σ denote the set of new entities in state i, i.e., $N_0^\sigma = N$ and $N_{i+1}^\sigma = E_{i+1}^\sigma \backslash E_i^\sigma$, and let $\theta_i^\sigma : LVar \rightharpoonup Ent$ denote the valuation at state i, where $\theta_i^\sigma(x) = \theta(x)$ if $\theta(x) \in E_k^\sigma$ for all $k \leqslant i$, and is undefined otherwise. The condition avoids that contradictions like $\exists x.\mathsf{X}(x\ \mathsf{dead} \Rightarrow \mathsf{X}x\ \mathsf{alive})$ are fulfilled. Note that once a logical variable is mapped to an entity, then this association remains along σ unless the entity dies, i.e., is deallocated. Thereafter, although the entity may be reallocated, the logical variable remains undefined. The satisfaction relation $\models$ is defined as follows:

$$
\begin{array}{lll}
\sigma, N, \theta \models x\ \mathsf{new} & \text{iff} & x \in \mathrm{dom}(\theta) \text{ and } \theta(x) \in N \\
\sigma, N, \theta \models x\ \mathsf{dead} & \text{iff} & x \notin \mathrm{dom}(\theta) \\
\sigma, N, \theta \models x = y & \text{iff} & x, y \in \mathrm{dom}(\theta) \text{ and } \theta(x) = \theta(y) \\
\sigma, N, \theta \models \exists x.\phi & \text{iff} & \exists e \in E_0^\sigma : \sigma, N, \theta\{e/x\} \models \phi \\
\sigma, N, \theta \models \neg\phi & \text{iff} & \sigma, N, \theta \not\models \phi \\
\sigma, N, \theta \models \phi \vee \psi & \text{iff} & \text{either } \sigma, N, \theta \models \phi \text{ or } \sigma, N, \theta \models \psi \\
\sigma, N, \theta \models \mathsf{X}\phi & \text{iff} & \sigma^1, N_1^\sigma, \theta_1^\sigma \models \phi \\
\sigma, N, \theta \models \phi\,\mathsf{U}\,\psi & \text{iff} & \exists i : (\sigma^i, N_i^\sigma, \theta_i^\sigma \models \psi \text{ and } \forall j < i : \sigma^j, N_j^\sigma, \theta_j^\sigma \models \phi).
\end{array}
$$

Here, $\theta\{e/x\}$ is defined as: $\theta\{e/x\}(x) = e$ and $\theta\{e/x\}(y) = \theta(y)$ for $y \neq x$.

Example 2.1. Properties concerning dynamic allocation and de-allocation can be formalised in $\mathcal{A}\ell\ell$TL. For example, formula $\mathsf{G}(\forall x.\forall y.\forall z.(x = y \vee x = z \vee y = z))$ asserts that the number of entities that are alive never exceeds 2, while $\mathsf{G}((\mathsf{F}\exists x.x\ \mathsf{new}) \wedge \forall x.\mathsf{X}(x\ \mathsf{alive}))$ states that the number of entities that are alive grows unboundedly. As a more involved example,

$$x\ \mathsf{alive}\,\mathsf{U}\,\exists y.(y\ \mathsf{new} \wedge (x\ \mathsf{alive}\,\mathsf{U}\,\exists z.(z\ \mathsf{new} \wedge y \neq z \wedge x\ \mathsf{alive})))$$

states that before x is deallocated, two new entities will be allocated. Note that formulas like $\mathsf{G}(x\ \mathsf{dead} \Rightarrow \mathsf{X}(x\ \mathsf{dead}))$, stating that entities cannot be allocated once they are de-allocated, and $\mathsf{X}(x\ \mathsf{dead} \vee x\ \mathsf{old})$ are tautologies.

Folded allocational sequences. In $\mathcal{A}\ell\ell$TL-formulae, entities can only be addressed through logical variables and valuations of variables (i.e., entities) can only be compared in the same state. These observations allow a reallocation (re-denomination) of entities from one state to its next state, as long as this is done injectively. For $E, E' \subseteq Ent$, a *reallocation* λ from E to E' is a partial injective function $\lambda : E \rightharpoonup E'$. A *folded allocational sequence* is an infinite alternating sequence $E_0\lambda_0 E_1\lambda_1 \cdots$, where λ_i is a reallocation from E_i to E_{i+1} for $i \geqslant 0$. We write λ_i^σ for the reallocation function of σ in state i. Note that for folded allocational sequence σ, $N_0^\sigma = N$, and $N_{i+1}^\sigma = E_{i+1}^\sigma \backslash \mathrm{cod}(\lambda_i^\sigma)$. Similarly, $\theta_0^\sigma = \theta$ and $\theta_{i+1}^\sigma = \lambda_i^\sigma \circ \theta_i^\sigma$. Thus, entity e is considered to be deallocated if $e \notin \mathrm{dom}(\lambda)$. Using these adapted definitions of N and θ, a satisfaction relation for $\mathcal{A}\ell\ell$TL can be defined in terms of folded allocational sequences precisely in the same way as above. The two kinds of sequences are equivalent models for $\mathcal{A}\ell\ell$TL-formulae [9]. The use of reallocations yields a local notion of entity identity that in turn allows minimisation of models [16].

3. High-level Allocational Büchi automata

In this section, we introduce an extension of (generalised) Büchi automata. High-level Allocational Büchi automata (HABA) generate folded allocational sequences and are inspired by History-Dependent automata [16]. HABA are basically Büchi automata where to each state a set of entities is associated. These entities, in turn, serve as valuation of logical (entity) variables.

Let $\infty \notin \mathit{Ent}$ be a special, distinguished entity, called *black hole*. Its role will become clear later on. We denote $E^\infty = E \cup \{\infty\}$ for arbitrary $E \subseteq \mathit{Ent}$. Furthermore, for $E, E_1 \subseteq \mathit{Ent}$, a ∞-*reallocation* is a partial function $\lambda : E^\infty \rightharpoonup E_1^\infty$ such that $\lambda(e) = \lambda(e') \neq \infty \Rightarrow e = e'$ for all $e, e' \in E$ and $\infty \in \mathrm{dom}(\lambda) \Rightarrow \lambda(\infty) = \infty$. That is, λ is injective when mapping away from ∞ and preserves ∞.

Definition 3.1. A *High-level Allocational Büchi Automaton* (HABA) $\mathcal{H}$ is a tuple $\langle X, Q, E, \rightarrow, I, \mathcal{F} \rangle$ with

- $X \subseteq \mathit{LVar}$ a finite set of logical variables;
- Q a (possibly infinite) set of states;
- $E : Q \to 2^{\mathit{Ent}} \times \mathbb{B}$, a function that associates to each state $q \in Q$ a finite set E_q of entities and a predicate B_q which holds iff there is a bounded number of entities in q.
- $\rightarrow \subseteq Q \times (\mathit{Ent}^\infty \rightharpoonup \mathit{Ent}^\infty) \times Q$, such that for $q \rightarrow_\lambda q'$, λ is an ∞-reallocation from E_q^∞ to $E_{q'}^\infty$ with (i) $\infty \in \mathrm{dom}(\lambda)$ iff $E_q = (E, \mathsf{ff})$ and $E_{q'} = (E', \mathsf{ff})$, and (ii) $\infty \in \mathrm{cod}(\lambda) \Rightarrow E_{q'} = (E', \mathsf{ff})$.
- $I : Q \rightharpoonup 2^{\mathit{Ent}} \times (X \rightharpoonup \mathit{Ent})$ a partial function yielding for every initial state $q \in \mathrm{dom}(I)$ an *initial valuation* (N, θ), where $N \subseteq E_q$ is a finite set of entities, and $\theta : X \rightharpoonup E_q$ is a partial valuation of the variables in X;
- $\mathcal{F} \subseteq 2^Q$ a set of sets of accept states.

We write $q \rightarrow_\lambda q'$ for $(q, \lambda, q') \in \rightarrow$. We adopt the generalised Büchi acceptance condition, i.e, $\rho = q_0 \lambda_0 q_1 \lambda_1 q_2 \cdots$ is a *run* of HABA $\mathcal{H}$ if $q_i \rightarrow_\lambda q_{i+1}$ for all $i \in \mathbb{N}$ and $|\{i | q_i \in F\}| = \omega$ for all $F \in \mathcal{F}$. Predicate B_q holds in state q iff the number of entities in q is bounded (denoted $\lceil q \rceil$). An unbounded state q (denoted $\lfloor q \rfloor$), possesses the distinguished entity ∞ that represents all entities that may be added to q. High-level state q thus represents all possible (concrete) states obtained from q by adding a finite number of entities to E_q. If a transition to state q' maps entities onto the black hole ∞, these entities cannot be distinguished anymore from there on. Moreover, if $q \rightarrow_\lambda q'$, entities in the black hole are either preserved (if $\lfloor q' \rfloor$), or are destroyed (if $\lceil q' \rceil$). The black hole thus allows to abstract from the identity of entities if these are not relevant anymore. The initial valuation (N, θ) associated to an initial state facilitates the generation of models for $\mathcal{A}\ell\ell$TL-formulae. This is shown in the following definition that formalises the correspondence between runs of the HABA and folded allocational sequences.

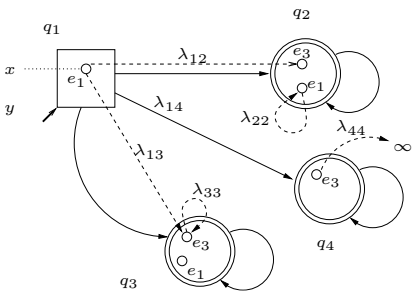
Definition 3.2. A run $\rho = q_0 \lambda_0 q_1 \lambda_1 \cdots$ of HABA $\mathcal{H} = \langle X, Q, E, \rightarrow, I, \mathcal{F} \rangle$ *generates* an allocation triple (σ, N, θ), where $\sigma = E_0 \lambda_0^\sigma E_1 \lambda_1^\sigma \cdots$ is a folded allocational

sequence, if there is a *generator*, i.e., a family of functions $\phi_i : E_i \to E_{q_i}^{\infty}$ satisfying for all $i \geqslant 0$:

1. $\forall e, e' \in E_i.\ (\phi_i(e) = \phi_i(e') \neq \infty \Rightarrow e = e')$
2. $\forall e \in E_{i+1}.\ (\phi_{i+1}(e) = \infty \Rightarrow e \in \mathrm{cod}(\lambda_i^{\sigma}))$
3. $\lceil q_i \rceil \Rightarrow (\forall e \in E_i : \phi_i(e) \neq \infty)$
4. $\lambda_i \circ \phi_i = \phi_{i+1} \circ \lambda_i^{\sigma}$
5. $E_{q_i} \subseteq \mathrm{cod}(\phi_i)$
6. $I(q_0) = (\phi_0(N), \phi_0 \circ \theta)$

In the previous definition, notice the difference between ∞-reallocations λ_i of HABA transitions and reallocations λ_i^{σ} of folded allocational sequence σ. Let $\mathsf{runs}(\mathcal{H})$ denote the set of runs of $\mathcal{H}$ and $\mathcal{L}(\mathcal{H}) = \{(\sigma, N, \theta) \mid \exists \rho \in \mathsf{runs}(\mathcal{H}) :\ \rho \text{ generates } (\sigma, N, \theta)\}$.

Example 3.3. The picture just below depicts a HABA with $X = \{x, y\}$. Squares denote bounded states, (large) circles denote unbounded states, small circles denote entities, and accept states have a double boundary. Here for simplicity we assume $|\mathcal{F}| = 1$. Dashed arrows indicate ∞-reallocations. In initial states, dotted lines represent θ, and filled circles denote new entities. In q_1, variable x denotes (old) entity e_1, while y is undefined. Entity e_3 in state q_2 represents the same entity as e_1 in q_1, while e_1 (in q_2) represents a new entity. Run $q_1\lambda_{12}(q_2\lambda_{22})^{\omega}$ generates sequences where the initial entity dies after the second state, while the new entity created in the second state will be alive forever. After the second state, at every step, a new entity is created and it will be alive only in one state. Run $q_1\lambda_{14}(q_4\lambda_{44})^{\omega}$ generates sequences where the entity in the initial state dies immediately. Once q_4 is reached, a new entity e_3 is created at every step, and in this run thus the number of entities grows unboundedly.



4. Programming allocation and deallocation

This section introduces a simple programming language $\mathcal{L}$ capturing the essence of allocation and deallocation. It is used for providing an intuition about the setup and the sort of behaviour that can be modelled by HABA. The operational semantics for $\mathcal{L}$ is defined using HABA as underlying model.

Syntax. For $PVar$ a set of program variables with $v, v_i \in PVar$ and $PVar \cap LVar = \varnothing$, the set of statements of $\mathcal{L}$ is given by:

$$
\begin{array}{lcl}
(p \in) \mathcal{L} & ::= & \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \parallel \cdots \parallel s_k) \\
(s \in) Stat & ::= & \mathsf{new}(v) \mid \mathsf{del}(v) \mid v := v \mid \mathsf{skip} \mid s; s \mid \mathsf{if}\ b\ \mathsf{then}\ s\ \mathsf{else}\ s\ \mathsf{fi} \\
 & & \mid \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od} \\
(b \in) Bexp & ::= & v = v \mid b \vee b \mid \neg b
\end{array}
$$

A program p is a parallel composition of a finite number of statements preceded by the declaration of a finite number of global variables. $\mathsf{new}(v)$ creates (i.e., allocates) a new entity that will be referred to by the program variable v. The old value of v is lost.

Thus, if v is the only variable that refers to entity e, say, then after the execution of $\mathsf{new}(v)$, e cannot be referenced anymore. In particular, e cannot be deallocated anymore. In other words, there is no automatic garbage collection. $\mathsf{del}(v)$ destroys (i.e., deallocates) the entity associated to v, and makes v undefined. The assignment $v := w$ passes the reference held by w (if any) to v. Again, the entity v was referring to might become unreferenced (for ever). Sequential composition, while, skip, and conditional statement have the standard interpretation. For the sake of simplicity, new and del create and destroy, respectively, a single entity only; generalisations in which several entities are considered simultaneously can be added in a straightforward manner.

Example 4.1. The following program, where $g(i) = (i+1) \bmod 4$, models the implementation of a naive solution to the dining philosopher problem:

$$\begin{array}{lcl} \mathit{DPhil} & \equiv & \mathsf{decl}\ v_1, v_2, v_3, v_4 : \mathit{Ph}_1 \parallel \mathit{Ph}_2 \parallel \mathit{Ph}_3 \parallel \mathit{Ph}_4) \text{ where} \\ \mathit{Ph}_i & \equiv & \mathsf{while}\ \mathsf{tt}\ \mathsf{do}\ \mathsf{if}\ (v_i\ \mathsf{alive} \wedge v_{g(i)}\ \mathsf{alive})\ \mathsf{then} \\ & & \quad \mathsf{del}(v_i);\ \mathsf{del}(v_{g(i)});\ \mathsf{new}(v_i);\ \mathsf{new}(v_{g(i)})\ \mathsf{else\ skip\ fi} \\ & & \mathsf{od} \end{array}$$

The variables v_i and $v_{g(i)}$ represent the left and the right chopstick of philosopher Ph_i, respectively. If v_i and $v_{g(i)}$ are defined[1], then the chopsticks are on the table. Taking the chopsticks from the table is represented by destroying the corresponding entities, while putting the chopsticks back on the table is modelled by creating new entities. Some properties that can possibly be satisfied by this program, stated in $\mathcal{A}\ell\ell$TL, are: $\mathsf{FG}(\forall x.\forall y.(x = y))$, expressing that eventually there is only one chopstick on the table (an inconsistency), or $\mathsf{G}(\forall x.\forall y.\forall z.(x\ \mathsf{old} \wedge y\ \mathsf{old} \wedge z\ \mathsf{old} \wedge (x = y \vee x = z \vee y = z)))$, expressing that among the philosophers there exists a greedy impostor who always eats and never thinks (an unfair computation).

Operational semantics. A (symbolic) semantics of our example language is given in terms of HABA where entities are represented by a partial partition of a subset of $PVar$; that is, the set E of entities is of the form $\{X_1, \ldots, X_n\}$ with $X_i \subseteq PVar$ and $X_i \cap X_j = \varnothing$ (for $i \neq j$). Note that we do not require $\bigcup_i X_i = PVar$ which would make it a full partitioning. Variable v is defined iff $v \in X_i$ for some i. Then, v refers to the entity represented by the set X_i. Otherwise, v is undefined. Using this approach, there is no need to represent (in a state) a mapping from the set of program variables onto the entities.

Let Par denote the compound statements, i.e., $r(\in Par) ::= s \mid r \parallel s$. The semantics of $p = \mathsf{decl}\ v_1, \ldots, v_n : (s_1 \parallel \cdots \parallel s_k)$ is the HABA $\mathcal{H}_p = \langle \varnothing, Q, E, \rightarrow, I, \mathcal{F} \rangle$ where

- $Q \subseteq Par \times 2^{2^{PVar}}$, i.e., a state $q = (r, E)$ consists of a compound statement and a set of entities; we have $\lfloor q \rfloor$ iff $\varnothing \in E$ (i.e., we represent the black hole by $\varnothing$).
- $E(r, E') = E' \backslash \{\varnothing\}$ and $I(s_1; \mathsf{skip} \parallel \cdots \parallel s_k; \mathsf{skip}, \varnothing) = (\varnothing, \varnothing)$;

[1] Notice that v dead iff $\neg(v = v)$. Again, here v alive stands for $\neg(v\ \mathsf{dead})$.

Table 1. Operational rules for the semantics of $\mathcal{L}$.

$$\frac{}{v := w, E \rightarrow_\lambda \mathsf{skip}, \{X_i \backslash \{v\} | w \notin X_i\} \cup \{X_i \cup \{v\} | w \in X_i\}} \; \lambda : X_i \mapsto \begin{cases} X_i \backslash \{v\} & \text{if } w \notin X_i \\ X_i \cup \{v\} & \text{otherwise} \end{cases}$$

$$\frac{}{\mathsf{new}(v), E \rightarrow_\lambda \mathsf{skip}, \{X_i \backslash \{v\} | X_i \in E\} \cup \{\{v\}\}} \; \lambda(X_i) = X_i \backslash \{v\}$$

$$\frac{v \in X_i}{\mathsf{del}(v), E \rightarrow_\lambda \mathsf{skip}, (E \backslash \{X_i\})} \; \lambda : X_j \mapsto \begin{cases} X_j & \text{if } j \neq i \\ \bot & \text{otherwise} \end{cases} \qquad \frac{s_1, E \rightarrow_\lambda s_1', E'}{s_1; s_2, E \rightarrow_\lambda s_1'; s_2, E'}$$

$$\frac{}{\mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}, E \rightarrow_{id} \mathsf{if}\ b\ \mathsf{then}\ s; \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, E} \qquad \frac{}{\mathsf{skip}; s_2, E \rightarrow_{id} s_2, E}$$

$$\frac{1 \leqslant j \leqslant k \ \wedge \ s_j, E \rightarrow_\lambda s_j', E'}{s_1 \parallel \cdots \parallel s_j \parallel \cdots \parallel s_k, E \rightarrow_\lambda s_1 \parallel \cdots \parallel s_j' \parallel \cdots \parallel s_k, E'} \qquad \frac{}{\parallel \mathsf{skip}, E \rightarrow_{id} \parallel \mathsf{skip}, E}$$

$$\frac{\mathcal{V}(b)(E)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, E \rightarrow_{id} s_1, E} \qquad \frac{\neg \mathcal{V}(b)(E)}{\mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, E \rightarrow_{id} s_2, E}$$

- $\rightarrow$ is the smallest relation defined by the rules in Table 1 such that for $r, E \rightarrow_\lambda r', E'$ we have $\varnothing \in E \Rightarrow \varnothing \in \mathrm{dom}(\lambda)$.
- let $\widehat{F}_i = \{(s_1' \parallel \cdots \parallel s_k', E) \in Q \mid s_i' = \mathsf{skip} \vee s_i' = \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}; s''\}$ and $\widetilde{F}_i = \{(s_1' \parallel \cdots \parallel s_k', E) \in Q \mid s_i' = \mathsf{skip} \vee s_i' = s; \mathsf{while}\ b\ \mathsf{do}\ s\ \mathsf{od}; s''\}$; then $\mathcal{F} = \{\widehat{F}_i \mid 0 < i \leqslant k\} \cup \{\widetilde{F}_i \mid 0 < i \leqslant k\}$.

A few remarks are in order. $\mathcal{H}_p$ has a single initial state $s_1; \mathsf{skip} \parallel \cdots \parallel s_k; \mathsf{skip}$, where each sequential component is terminated by a skip statement. The set of accept states for the i-th sequential component consists of all states in which the component has either terminated ($s_i = \mathsf{skip}$) or is processing a loop (which could be infinite).

The condition on $\varnothing$ in the definition of $\rightarrow$ can be seen as a kind of "preservation law" of the black hole. In fact, once a state explodes into an unbounded one, the black hole generated by this explosion will last forever. Note that in Def. 3.1 this is not always the case. The semantics of the boolean expressions is given by the function $\mathcal{V} : Bexp \times 2^{2^{PVar}} \rightarrow \mathbb{B}$ defined by $\mathcal{V}(v = w, E) = \mathsf{tt}$ if $\exists X_i \in E : v, w \in X_i$ and false otherwise, $\mathcal{V}(b_1 \vee b_2, E) = \mathcal{V}(b_1, E) \vee \mathcal{V}(b_2, E)$, and $\mathcal{V}(\neg b, E) = \neg \mathcal{V}(b, E)$. Note that $\parallel \mathsf{skip}$ is a shorthand for $\mathsf{skip} \parallel \ldots \parallel \mathsf{skip}$. Whenever entity X_i is not referenced by any program variable, the state will become unbounded. Entity X_i will then be mapped by λ onto $\varnothing$ (recall that in the special case of $\mathcal{H}_p$ we represent ∞ by $\varnothing$), which can be viewed as a "black hole" collecting every non-referenced entity. These entities share the property that they cannot be deallocated anymore, thus they will have the same future, i.e, they will be "floating" in the black hole *ad infinitum*.

Although, there may be an unbounded number of entity creations, for the semantics defined in this section we have the following result:

Theorem 4.2. For any $p \in \mathcal{L}$: $\mathcal{H}_p$ is finite state.

In [9] it is shown that $|Q_{\mathcal{H}_p}|$ is exponential in the number of sequential components of p and super-exponential in $|PVar|$.

5. Model-checking $\mathcal{A}\ell\ell$TL

In this section, we define an algorithm for model-checking $\mathcal{A}\ell\ell$TL-formulae against a HABA. The algorithm extends the tableau method for LTL [14] to $\mathcal{A}\ell\ell$TL.

We will evaluate $\mathcal{A}\ell\ell$TL-formulae on states of a HABA by mapping the free variables of the formula to entities of the state. It should be clear that, in principle, any such mapping resolves all basic propositions. In turn, the basic propositions determine the validity of arbitrary formulae. There are, however, two obstacles to this principle, the first of which is slight and the other more difficult to overcome.

- It is not always uniquely determined whether or not an entity is fresh in a state. Our model allows states in which a given entity is considered fresh when arriving by one incoming transition (since it is not in the codomain of the reallocation associated with that transition), but not when arriving by another (the entity is the image of an entity in the previous state). This obstacle is dealt with by *duplicating* the states where such an ambiguity exists.
- For variables (of the formula in question) that are mapped onto entities in the black hole, entity equations are not resolved, since it is not clear whether the variables are mapped to distinct entities that have imploded into the black hole, or to the same one. To deal with this obstacle, we introduce an intermediate layer in the evaluation of the formula on the state. This additional layer consists of a *partial partitioning* of the free variables; that is, a set of nonempty, disjoint subsets of the set of all free variables. An entity equation is then resolved by the question whether the equated variables are in the same partition. It is the partitions, rather than the individual variables, that are mapped to the entities.

Assumptions. The duplication proposed above to overcome the first of these obstacles is straightforward; we will not work it out in more detail in this paper (see [9] for details). In the remainder, we assume that the necessary duplication has been carried out already: that is, we will assume that for every state $q \in Q$ there is an associated set $N_q \subseteq E_q$ that contains the entities that are *new in* q; i.e., such that

a) $q' \rightarrow_\lambda q$ implies $E_q \setminus \mathrm{cod}(\lambda) = N_q$ b) $I(q) = (N, \theta)$ implies $N = N_q$.

Note that, because of b), we can henceforth assume that I has just θ as its image — the component N is now uniquely associated with q. Another assumption needed below is that every quantified variable actually appears free in the subformula; that is, we only consider formulae $\exists x.\phi$ for which $x \in fv(\phi)$. Note that this imposes no real restriction, since $\exists x.\phi$ is equivalent to $\exists x.(x \ \mathsf{alive} \wedge \phi)$.

Valuations. A valuation of a formula in a given state is an interpretation of the free variables of the formula as entities of the state. Such an interpretation establishes the validity of at least the *atomic propositions* within the formula, i.e., the sub-formulae of the form $x = y$ (which holds if x and y are interpreted as the same entity) and x new (which holds if x is interpreted as a fresh entity).

Definition 5.1 (valuations). Let $E \subseteq Ent^\infty$. An *E-valuation* is a triple (ϕ, Ξ, Ψ) where ϕ is an $\mathcal{A}\ell\ell$TL-formula and

- Ξ is a partial partitioning of $fv(\phi)$; that is, $\Xi = \{X_1, \ldots, X_n\}$ such that $\varnothing \subset X_i \subseteq fv(\phi)$ for $1 \leq i \leq n$ and $X_i \cap X_j = \varnothing$ for $1 \leq i < j \leq n$ (but not necessarily $\bigcup_n X_n = fv(\phi)$, which would make it a *full* partitioning).
- $\Theta\colon \Xi \to E$ is a function mapping the partitions of Ξ to E, such that Θ is injective where it maps away from ∞ — i.e., $\Theta(X_i) = \Theta(X_j) \neq \infty \Rightarrow i = j$.

This is easily lifted to the states of a HABA: (ϕ, Ξ, Θ) is a q-valuation (for some $q \in Q_{\mathcal{H}}$) if it is an E_q-valuation (if $\lceil q \rceil$) or E_q^∞-valuation (if $\lfloor q \rfloor$). We write $V_q(\phi)$, ranged over by v, to denote the set of q-valuations of ϕ, and V_q to denote the set of *all* q-valuations. We denote the components of a valuation v as $(\phi_v, \Xi_v, \Theta_v)$.

A technicality: below we will need to restrict partial partitioning Ξ and mappings Θ of a valuation (ϕ, Ξ, Θ) to subformulae of ϕ, which means restricting the underlying sets of (free) variables upon which Ξ and Θ are built to those of that subformula. For this purpose, we define $\Xi \restriction \psi = \{X \cap fv(\psi) \mid X \in \Xi, X \cap fv(\psi) \neq \varnothing\}$ and $\Theta \restriction \psi = \{(X \cap fv(\psi), \Theta(X)) \mid X \in \mathrm{dom}(\Theta), X \cap fv(\psi) \neq \varnothing\}$.

The *atomic proposition valuations* of a state q of a HABA are those q-valuations of basic propositions of $\mathcal{A}\ell\ell$TL (i.e., freshness predicates and entity equations) that make the corresponding properties true.

Definition 5.2. Let $\mathcal{H}$ be a HABA and let $q \in Q_{\mathcal{H}}$ be arbitrary. The *atomic proposition valuations* of q are defined by the set $AV_q \subseteq V_q$ of all triples (ϕ, Ξ, Θ) for which one of the following holds:

- $\phi = \mathsf{tt}$;
- $\phi = (x = y)$, and $x, y \in X$ for some $X \in \Xi$;
- $\phi = (x\ \mathsf{new})$, and $x \in X \in \Xi$ implies $\Theta(X) \in N_q$.

Closure. Along the lines of [14], we associate to each state q of a HABA sets of q-valuations, specifically aimed at establishing the validity of a given formula ϕ. For this purpose, we first collect all $\mathcal{A}\ell\ell$TL-formulae whose validity is possibly relevant to the validity of a given formula ϕ into the so-called *closure of* ϕ.

Definition 5.3. Let ϕ be an $\mathcal{A}\ell\ell$TL-formula. The *closure* of ϕ, $CL(\phi)$, is the smallest set of formulae (identifying $\neg\neg\psi$ with ψ) such that:

- $\phi, \mathsf{tt}, \mathsf{ff} \in CL(\phi)$;
- $\neg\psi \in CL(\phi)$ iff $\psi \in CL(\phi)$;
- if $\psi_1 \vee \psi_2 \in CL(\phi)$ then $\psi_1, \psi_2 \in CL(\phi)$;
- if $\exists x.\psi \in CL(\phi)$ then $\psi \in CL(\phi)$;
- if $\mathsf{X}\psi \in CL(\phi)$ then $\psi \in CL(\phi)$;
- if $\neg\mathsf{X}\psi \in CL(\phi)$ then $\mathsf{X}\neg\psi \in CL(\phi)$;
- if $\psi_1\, \mathsf{U}\, \psi_2 \in CL(\phi)$ then $\psi_1, \psi_2, \mathsf{X}(\psi_1\, \mathsf{U}\, \psi_2) \in CL(\phi)$.

Since valuations map (sets of) variables of a given formula to entities, possibly to the black hole, it is important to know how many of these variables have to be

taken into account at the most. This is obviously bounded by the number of variables occurring (free or bound) in ϕ, but in fact we can be a little more precise: the number is given by $K(\phi)$ defined as $K(\phi) = \max\{|fv(\psi)| \mid \psi \in CL(\phi)\}$.

The interesting case for the model checking construction is when one or more variables are indeed mapped to the black hole. Among other things, we will then have to make sure that sufficiently many entities of the state have imploded into the black hole to meet the demands of the valuation. For this purpose, we introduce the *black number* of a function, which is the number of entities that that function maps (implodes) into the black hole. For an arbitrary set A and (partial) mapping $\alpha\colon A \rightharpoonup Ent^\infty$ we define $\Omega(\alpha) = |\{a \in A \mid \alpha(a) = \infty\}|$.

Tableau graph. We now construct a graph that will be the basis of the model checking algorithm. The nodes of this graph, called *atoms* after [14], are built from states of a HABA, valuations of formulae from the closure, and a bound on the black number.

Definition 5.4. Given a HABA $\mathcal{H}$ and an $\mathcal{A}\ell\ell$TL-formula ϕ, an *atom* is a triple (q, D, k) where $q \in Q_\mathcal{H}$, $D \subseteq \{v \in V_q(\psi) \mid \psi \in CL(\phi), \Omega(\Theta_v) \leq k\}$ and $k \leq K(\phi)$ if $\lfloor q \rfloor$ or $k = 0$ if $\lceil q \rceil$, such that for all $v = (\psi, \Xi, \Theta) \in V_q$ with $\psi \in CL(\phi)$ and $\Omega(\Theta) \leq k$:

- if $v \in AV_q$, then $v \in D$;
- if $\psi = \neg\psi'$, then $v \in D$ iff $(\psi', \Xi, \Theta) \notin D$;
- if $\psi = \psi_1 \vee \psi_2$, then $v \in D$ iff $(\psi_i, \Xi \restriction \psi_i, \Theta \restriction \psi_i) \in D$ for $i = 1$ or $i = 2$;
- if $\psi = \exists x.\psi'$, then $v \in D$ iff there exists a $(\psi', \Xi', \Theta') \in D$ such that $\Xi = \Xi' \restriction \psi$, $\Theta = \Theta' \restriction \psi$ and $x \in \bigcup \Xi'$;
- if $\psi = \neg \mathsf{X}\psi'$, then $v \in D$ iff $(\mathsf{X}\neg\psi', \Xi, \Theta) \in D$;
- if $\psi = \psi_1 \,\mathsf{U}\, \psi_2$, then $v \in D$ iff either $(\psi_2, \Xi \restriction \psi_2, \Theta \restriction \psi_2) \in D$, or both $(\psi_1, \Xi \restriction \psi_1, \Theta \restriction \psi_1) \in D$ and $(\mathsf{X}\psi, \Xi, \Theta) \in D$.

The set of all atoms for a given formula ϕ constructed on top of $\mathcal{H}$ is denoted $A_\mathcal{H}(\phi)$, ranged over by A, B. We denote the components of an atom A by (q_A, D_A, k_A).

Definition 5.5. The *tableau graph* for a HABA $\mathcal{H}$ and an ATL-formula ϕ, denoted $G_\mathcal{H}(\phi)$, consists of vertices $A_\mathcal{H}(\phi)$ and edges $\rightarrow \subseteq A_\mathcal{H}(\phi) \times (Ent^\infty \rightharpoonup Ent^\infty) \times A_\mathcal{H}(\phi)$ determined by:

$$
\begin{aligned}
(q, D, k) \rightarrow_\lambda (q', D', k') \quad \text{iff} \quad & q \rightarrow_\lambda q', \\
& \forall \mathsf{X}\psi \in CL(\phi)\colon (\mathsf{X}\psi, \Xi, \Theta) \in D \Leftrightarrow (\psi, \Xi, \lambda \circ \Theta) \in D', \\
& k' = \begin{cases} \min(K(\phi), k + \Omega(\lambda)) & \text{if } \lfloor q' \rfloor \\ 0 & \text{if } \lceil q' \rceil. \end{cases}
\end{aligned}
$$

Note that if $\mathcal{H}$ is finite-state, then $G_\mathcal{H}(\phi)$ can be effectively constructed: the set of atoms is finite for every given state. A *path* through a tableau graph is an infinite sequence of states and transitions, starting at an initial state of the HABA and satisfying the acceptance condition of the HABA, such that all "until"-subformulae in any of the atoms are satisfied somewhere further down the sequence.

Definition 5.6. An *allocational path* in $G_{\mathcal{H}}(\phi)$ is an infinite sequence $\pi{=}(q_0, D_0, k_0)$ λ_0 (q_1, D_1, k_1) $\lambda_1 \cdots$ such that:

1 $q_0\lambda_0 q_1\lambda_1 \cdots \in \mathsf{runs}(\mathcal{H})$;

2 for all $i \geq 0$, $(q_i, D_i, k_i) \rightarrow_{\lambda_i} (q_{i+1}, D_{i+1}, k_{i+1})$;

3 for all $i \geq 0$ and all $(\psi_1 \,\mathsf{U}\, \psi_2, \Xi, \Theta) \in D_i$, there exists a $j \geq i$ such that $(\psi_2, \Xi \restriction \psi_2, \lambda_{j-1} \circ \cdots \lambda_i \circ (\Theta \restriction \psi_2)) \in D_j$.

Given an allocational path π in $G_{\mathcal{H}}(\phi)$ of this form, we say that π *fulfills* ϕ if the underlying run $\rho = q_0\lambda_0 q_1\lambda_1 \cdots$ generates an allocation triple (σ, N, θ) with a generator $(h_i)_{i\in\mathbb{N}}$ such that $k_0 = \min(K(\phi), \Omega(h_0))$ and $\sigma, N, \theta \vDash \phi$. If ϕ is clear from the context, we call π a *fulfilling path*. Furthermore, if there exists $(\sigma, N, \theta) \in \mathcal{L}(\mathcal{H})$ such that $\sigma, N, \theta \models \phi$ we say that ϕ is $\mathcal{H}$-satisfiable.

This sets the stage for the main results. We first state the correspondence between the fulfilment of a formula by a path and the presence of that formula in the initial atom of the path. For a partition interpretation Θ let $\overline{\Theta}\colon fv(\phi) \rightharpoonup Ent^{\infty}$ (*flattening* of Θ) be defined as $\overline{\Theta}\colon x \mapsto \Theta(X)$ if $x \in X \in \mathrm{dom}(\Theta)$.

Proposition 5.7. A path π in $G_{\mathcal{H}}(\phi)$ fulfills ϕ if and only if there exists $(\phi, \Xi, \Theta) \in D_0$ (for some Ξ, Θ) such that $I_{\mathcal{H}}(q_0) = \overline{\Theta}$.

Furthermore, there is a correspondence between the satisfiability of a formula in the HABA and the existence of a fulfilling path in the tableau graph.

Proposition 5.8. ϕ is $\mathcal{H}$-satisfiable iff there exists a path in $G_{\mathcal{H}}(\phi)$ that fulfills ϕ.

From now on we can (almost) rely on standard theory (see [14]). The first observation is that a tableau graph can have infinitely many different paths, therefore looking for a fulfilling path for ϕ is still not an effective method for model checking. We need the following definitions.

A subgraph $G' \subseteq G_{\mathcal{H}}(\phi)$ is *self-fulfilling* if every node A in G' has at least an outgoing edge and for every $(\psi_1 \,\mathsf{U}\, \psi_2, \Xi, \Theta) \in D_A$ there exists a node $B \in G'$ s.t.

- $A = A_0 \rightarrow_{\lambda_0} A_1 \rightarrow_{\lambda_1} \cdots \rightarrow_{\lambda_{i-2}} A_{i-1} \rightarrow_{\lambda_{i-1}} A_i = B$
- $(\psi_2, \Xi \restriction \psi_2, \lambda_{i-1} \circ \cdots \lambda_0 \circ (\Theta \restriction \psi_2)) \in D_B$.

A *prefix* in $G_{\mathcal{H}}(\phi)$ is a sequence $A_0 \rightarrow_{\lambda_0} A_1 \rightarrow_{\lambda_1} \cdots \rightarrow_{\lambda_{i-2}} A_{i-1} \rightarrow_{\lambda_{i-1}} A_i$ such that A_0 is an initial atom (i.e., $q_{A_0} \in I_{\mathcal{H}}$) and A_i is in a self-fulfilling subgraph.

Let $\mathit{Inf}(\pi)$ denote the set of nodes that appear infinitely often in the path π. $\mathit{Inf}(\pi)$ is a strongly connected subgraph (SCS). We can prove the following implications:

Proposition 5.9. π is a fulfilling path in $G_{\mathcal{H}}(\phi) \Rightarrow \mathit{Inf}(\pi)$ is a self-fulfilling SCS of $G_{\mathcal{H}}(\phi)$.

Proposition 5.10. Let $G' \subseteq G_{\mathcal{H}}(\phi)$ be self-fulfilling SCS such that

- there exists a fulfilling prefix of G' starting at an initial atom A with $(\phi, \Xi, \Theta) \in D_A$ such that $I_{\mathcal{H}}(q_A) = \overline{\Theta}$;

- for all $F \in \mathcal{F}_{\mathcal{H}} : F \cap \{q | (q, D, k) \in G'\} \neq \varnothing$;

Then there exists a path π in $G_{\mathcal{H}}(\phi)$ that fulfils ϕ and such that $Inf(\pi) = G'$.

Finally, we present the main result of the paper:

Theorem 5.11. For any HABA $\mathcal{H}$ and formula ϕ, it is decidable whether or not ϕ is $\mathcal{H}$-satisfiable.

The complexity of the algorithm is double exponential in $|\phi|$, polynomial in $|Q_{\mathcal{H}}|$ (conjecture) and in the largest number of entities (in a state). A detailed analysis can be found in [9].

6. Related and future work

History-dependent automata. History-dependent (HD) automata [16] are the main inspiration for HABAs. An HD-automaton is an automaton where states, transitions and labels are equipped with a set of local names that can be created dynamically. HD-automata represent an adequate model for history-dependent formalisms such as the π-calculus. Reallocation of entities in HABA resembles the reallocation of names in HD-automata. The novelty introduced in HABAs is the black hole abstraction. This key feature allows us to deal with a possibly unbounded number of entities.

Spatial logic. Related to $\mathcal{A}\ell\ell$TL, concerning properties of freshness, is the Spatial Logic (SL) [4, 3]. SL is defined for the Ambient Calculus and has modalities that refer to space as well as time. Freshness can be identified in SL using a special quantifier, and has a somewhat different interpretation than in $\mathcal{A}\ell\ell$TL. In SL "fresh" means distinct from any name used in the formula and in the model satisfying it. If there is a fresh name, there are infinitely many of them. In contrast, in $\mathcal{A}\ell\ell$TL, if an entity is fresh it means that the entity is used in the current state and did not appear previously. This conceptual difference has several consequences. For instance, there exist non-contradictory $\mathcal{A}\ell\ell$TL-formulae where more than one distinct fresh entity is identified in the same state. Another difference between SL and $\mathcal{A}\ell\ell$TL concerns quantification. In SL, quantification is over a fixed (countable) set of names, whereas in $\mathcal{A}\ell\ell$TL, quantification ranges over entities that are alive in the current state. This set is not fixed from state to state. Therefore, e.g., $\forall x.\mathsf{X}\phi$ is not equivalent to $\mathsf{X}\forall x.\phi$.

Tableau-based methods. There are basically two approaches to model-checking temporal logics: the automata-theoretic approach (for LTL [19] and CTL [10, 13]) and the tableau method. Tableaux are typically used for the solution of more general problems, like satisfiability. For model checking, the tableau approach was first developed for CTL [6, 2]. Our algorithm is based on the tableau method for LTL reported in [14].

Model-checking and logics for object-oriented systems. Model-checking tools for object-oriented systems are becoming more and more popular, but the property specification formalisms are not tailored towards properties over objects (such as allocation and de-allocation). Bandera [7] is a model checker for Java that uses abstract interpretation and program slicing to yield compact state spaces. Another model checker for Java is Java PathFinder [12]. JPF employs garbage collection in order to obtain a finite state space. Dynamic creation of objects is only supported to a limited extent (the

number of created objects must be bounded). The verification of (only) *safety* properties for systems with an unbounded number of objects is recently reported in [20]. Opposed to our approach which always provides correct answers, this approach may report *false negatives*. Apart from these tool-oriented approaches, several temporal logics for object-oriented systems have been defined [18, 11, 8], that, however, do not support primitives for the birth and death of objects.

Future work. In the future we plan to investigate the use of HABA-like models for the definition of the semantics of more realistic OOP languages. The first step would be the definition of automata where entities can reference each other. Another (long term) open research question that needs further investigation is satisfiability of $\mathcal{A}\ell\ell$TL. Similarly, it would be interesting to develop a proof theory for $\mathcal{A}\ell\ell$TL, as well as, to explore a possible embedding of LTL in $\mathcal{A}\ell\ell$TL.

References

[1] M. Abadi, A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. & Comp.* 148(1): 1-70, 1999.

[2] M. Ben-Ari, A. Pnueli, Z. Manna. The temporal logic of branching time. *Acta Inf.* 20(3):207–226,1983.

[3] L. Caires, L. Cardelli. A spatial logic for concurrency (part I). In *TACS'01*, LNCS 2255:1–37, Springer, 2001.

[4] L. Cardelli, A. Gordon. Logical properties of name restriction. In *TLCA'01*, LNCS 2044:46–60, Springer, 2001.

[5] L. Cardelli, A. Gordon. Mobile ambients. In *FoSSaCS'98*, LNCS 1378:140–155, Springer, 1998.

[6] E. Clarke, E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, LNCS 131:52–71, Springer, 1981.

[7] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, H. Zheng. Bandera: Extracting finite-state models from Java source code. In *ICSE'00* pp. 439–448, IEEE CS Press, 2000.

[8] D. Distefano, J.-P. Katoen, A. Rensink. On a temporal logic for object-based systems. In *FMOODS'00*, pp. 305–326, Kluwer, 2000.

[9] D. Distefano, A. Rensink J.-P. Katoen. Model checking dynamic allocation and deallocation. Technical report TR-01-40, University of Twente, 2002. Available on line at `http://fmt.cs.utwente.nl/~ddino/papers/DSRK01-report.ps.gz`

[10] E. A. Emerson. Automata, tableaux and temporal logics. In *Logic of Programs*, LNCS 193:79–88, Springer, 1985.

[11] J. Fiadeiro, T. Maibaum. Verifying for reuse: foundations of object-oriented system verification. In *Theory and Formal Methods*, pp. 235–257, 1995.

[12] K. Havelund, T. Pressburger. Model checking Java programs using Java PathFinder. *Int. J. on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

[13] O. Kupferman, M. Y. Vardi, P. Wolper. An automata-theoretic approach to branching-time model checking. *J. of the ACM*, 47(2):312–360, 2000.

[14] O. Lichtenstein, A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL'85*, pp. 97–107, ACM Press, 1985.

[15] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Inf. & Comp.* 100(1):1-77, 1992.

[16] U. Montanari, M. Pistore. An introduction to history-dependent automata. *Electr. Notes in Th. Comp. Sci.*, 10, 1998.

[17] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pp. 46–57, IEEE CS Press, 1977.

[18] A. Sernadas, C. Sernadas, J.F. Costa. Object specification logic. *J. of Logic & Computation*, 5(5):603–630, 1995.

[19] M. Y. Vardi, P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS'86*, pp. 332–344, IEEE CS Press, 1986.

[20] E. Yahav. Verifying safety properties of concurrent Java programs using 3-valued logic. In *POPL 2001*, pp. 27-40 ACM Press, 2001.