Decidable Navigation Logics for Object Structures

Frank S. de Boer and Rogier M. van Eijk

Institute of Information and Computing Sciences
Utrecht University, P.O. Box 80.089
3508 TB Utrecht, The Netherlands
{frankb,rogier}@cs.uu.nl

Abstract. In this paper, we introduce decidable multimodal logics to describe and reason about navigation across object structures. The starting point of these navigation logics is the modelling of object structures as Kripke models that contain a family of deterministic accessibility relations; one for each pointer attribute. These pointer attributes are used in the logics both as first-order terms in equalities and as modal operators. To handle the ambiguities of pointer attributes the logics also cover a mechanism to bind logical variables to objects that are reachable by a pointer. The main result of this paper is a tableau construction for deciding the validity of formulas in the navigation logics.

1 Introduction

In describing structures of objects, we distinguish two main levels of abstraction. First, there is the *modelling level* as specified by the Unified Modelling Language (UML) [10]. At this level, in UML, a structure of objects is described in terms of a diagram consisting of classes that are related to each other via associations. Consider for instance the diagram depicted in Figure 1, which covers a class $Person$ that is related to a class $Book$ via the association $Author$ and a class $Company$ that is related to the class $Book$ via the association $Publisher$. As indicated in the diagram, the multiplicity of these associations $Author$ and $Book$ is one-to-many.

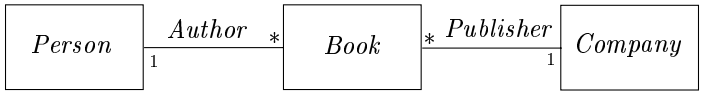

Fig. 1. A class diagram at the modelling level

Second, we distinguish the *implementation level*, which describes the execution of an object-oriented programming language, like for instance JAVA, directly in terms of the class *instances*. At this level, the associations of the high-level class diagram are implemented by means of pointer attributes and *collection objects* like sets, sequences, enumerations and bags that act as *multiplexers*. Consider for example the object structure in Figure 2, which depicts an instance p of the class $Person$, instances b_1, b_2 and

b_3 of the class $Book$, instances c_1 and c_2 of the class $Company$, and multiplexers m_1, m_2 and m_3. Persons have a pointer named $AuthorOf$ to a multiplexer, multiplexers have pointers named $item(1)$, $item(2)$, $item(3)$ etc. to books, books have a pointer $AuthoredBy$ to a person and a pointer $PublishedBy$ to a company, and companies have a pointer $PublisherOf$ to a multiplexer.

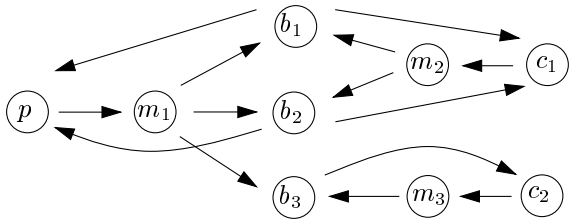

Fig. 2. An object structure at the implementation level

A central concept for the description of object structures at both the modelling and implementation level is *navigation* [7]. The main problem addressed in this paper, is a logical description of navigation at the implementation level, which denotes the operation of successively following pointers across an object structure. A crucial property of navigation is *reachability*; i.e., the question whether starting at an object it is possible to reach another object by navigation. For instance, in the object structure of Figure 2, person p is the author of a book that is published by company c_2. In terms of navigation this means that it is possible to reach c_2 from p by navigating the attributes $AuthorOf$, $item(3)$ and $PublishedBy$. Conversely, company c_2 is the publisher of a book that is authored by p. However, it is not possible to reach p from c_2 by navigation because there is no pointer from b_3 to p. This could indicate an *error* in the programming code.

Standard *first-order logic*, which allows only quantification over objects, is not expressive enough to describe reachability in a network of objects. Moreover, the validity problem for first-order logic is undecidable. Standard *modal logics* [9] are suited to navigate through Kripke models, but although they are decidable, they also lack expressive power because they do not distinguish between *bisimilar* Kripke models, such as for instance between a loop and its infinite unfolding.

Over the last decade, a new family of modal logics has come up, which combine modal operators with first-order variable-binding mechanisms. These languages are referred to as the family of *hybrid logics* [2]. In contrast to standard *first-order modal logic* [6], these logics cover mechanisms to bind variables to the *worlds* of a Kripke model. In particular, in [5], a general logical framework is presented that extends the navigation mechanism of standard modal logic with a variable-binding mechanism that allows the binding of variables to worlds that are in the domain of the current world, like for instance the worlds that are accessible.

The starting point of the navigation logics presented in this paper is the modelling of object structures at the implementation level as Kripke models that contain a family of *deterministic* accessibility relations; one for each pointer attribute. Additionally, we

introduce a variable-binding mechanism along the lines of [5] that allows the binding of logical variables to objects that are reachable by a pointer. These pointer attributes can be used in the logic both as first-order terms in equalities and as modal operators. The main result of the paper is a tableau construction for deciding the validity of formulas in the navigation logics.

The plan of the paper is as follows. In Section 2, we define the syntax of the basic navigation logic. The semantics of this logic is developed in Section 3. Additionally, in Section 4, we present a tableau construction for deciding the validity of formulas in the basic navigation logic. In the subsequent section, we discuss two decidable extensions of the basic navigation logic; viz. an extension that includes a collection of jump operators and an extension that covers navigation programs. Finally, in Section 6, we wrap up by providing several directions for future research.

2 Basic Navigation Logic

In this section, we define the basic navigation logic. Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a finite set of *pointer attributes* and $\mathcal{C} = \{nil,\ self\}$ be the set of *constants*. The constant nil is used to denote 'undefined' and the constant $self$ to denote an object itself. Additionally, let Var be a set of *variables* with typical elements x, y, z. The set $\mathcal{T} = \mathcal{A} \cup \mathcal{C} \cup Var$, with typical element t, denotes the set of *terms*. Finally, let $\mathcal{P}$ be a set of *propositional atoms*, with typical elements p, q and r.

Definition 1. *The* basic navigation language $\mathcal{L}_0$ *consists of formulas* φ *that are generated by the following BNF-grammar:*

$$\varphi ::= p \mid (t_1 = t_2) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \langle A\rangle\varphi \mid \exists_{x=t}(\varphi)$$

We have the usual abbreviations $\varphi \rightarrow \psi$ *for* $\neg\varphi \vee \psi$, $\varphi_1 \wedge \varphi_2$ *for* $\neg(\neg\varphi_1 \vee \neg\varphi_2)$, *and* $\varphi \leftrightarrow \psi$ *for* $\varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$.

In the language $\mathcal{L}_0$, pointer attributes can be used both as first-order terms in equalities and as *modal* operators for navigation, which are applied to *formulas*: A formula $\langle A\rangle\varphi$ expresses that φ holds for the object that results from following the pointer A.

Different objects, and in particular different objects from the same class, can have attributes with the same name. To handle the ambiguities of these pointer attributes, i.e., in order to be able to compare the pointer attributes of different objects, the language covers a *variable-binding* mechanism: $\exists_{x=t}(\varphi)$ denotes the formula φ in which the variable x is bound to the object denoted by the term t. For instance, the formula $\exists_{x=A}\langle B\rangle\neg(x = A)$ expresses that the pointer attribute A of the current object and the pointer attribute A of the object that is reached by following the pointer B from the current object, have different denotations. This variable-binding mechanism is further illustrated in the example below.

Example 1. Consider the structure depicted in Figure 3 in which the objects are arranged in a ring. Each object has two pointers L and R to denote its left and its right neighbour, respectively. As an example, we have that each object is the left neighbour of

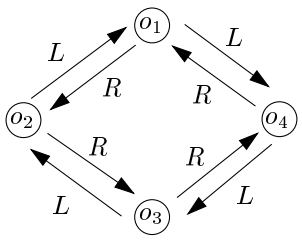

Fig. 3. Objects arranged in a ring structure

its right neighbour; formally, for each object in the ring the formula $\exists_{x=self}\langle R\rangle(L = x)$ is true. Second, each object's right neighbour is the same object as the left neighbour of the left neighbour of its left neighbour; that is, for each object the formula $\exists_{x=R}\langle L\rangle\langle L\rangle(L = x)$ holds.

The above example illustrates the difference with the variable-binding mechanism of *first-order logic*, in which this form of binding can be modelled by substitution. That is, in first-order logic, the formula $\exists_{x=t}(\varphi)$ can be modelled by $\varphi[t/x]$, which denotes the substitution of t for x in φ. If we would apply such substitution to the formula $\exists_{x=R}\langle L\rangle\langle L\rangle(L = x)$ we obtain $\langle L\rangle\langle L\rangle(L = R)$, which expresses that for the left neighbour of an object's left neighbour it holds that the left neighbour and the right neighbour are the same. This does not hold for any of the objects in the above figure.

Finally, as our primary concern is the development of a *decidable* navigation logic, we only include the datatype of pointers. We assume a *propositional encoding* of information about the state of an object, which may include other datatypes like integers and booleans; e.g., the fact $x < y$, which expresses that the value of the integer variable x is less than the value of y, can be represented by a particular proposition p.

3 Semantics

In this section, we introduce the semantics of $\mathcal{L}_0$ that is based on a formal description of object structures in terms of Kripke models that contain a deterministic accessibility relation for each pointer attribute.

Definition 2. *A* Kripke model M *is a triple* (O, K, π)*, where* O *denotes the set of objects, which form the* states *or* worlds *of the Kripke model,* K *is a total function of type*

$$K : O \to (\mathcal{A} \to O)$$

and π *is a valuation function of type*

$$\pi : O \to \wp(\mathcal{P}).$$

We assume an element $\perp \in O$ that stands for 'undefined', i.e., the value of nil. The function K defines the accessibility relations of the model; that is, for each object

o and attribute A, we have that $K(o)(A)$ denotes the object that is accessible from o by following the pointer A. Moreover, if $K(o)(A) = \bot$ then A is called a nil pointer. Additionally, the function π constitutes a valuation function that maps each world o to the set of propositions that are true in it.

Definition 3. *Given a model $M = (O, K, \pi)$, a state $o \in O$ and an assignment function $s : Var \to O$, the* interpretation $[\![t]\!]_{M,o,s}$ of a term $t \in \mathcal{T}$ *is defined by:*

$$\begin{aligned} [\![nil]\!]_{M,o,s} &= \bot \\ [\![self]\!]_{M,o,s} &= o \\ [\![A]\!]_{M,o,s} &= K(o)(A) \\ [\![x]\!]_{M,o,s} &= s(x) \end{aligned}$$

Additionally, the truth definition $M, o, s \models \varphi$ for the language $\mathcal{L}_0$ *is inductively given as follows:*

$$\begin{array}{lll} M, o, s \models (t_1 = t_2) & \Leftrightarrow & [\![t_1]\!]_{M,o,s} = [\![t_2]\!]_{M,o,s} \\ M, o, s \models p & \Leftrightarrow & p \in \pi(o) \\ M, o, s \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & M, o, s \models \varphi_1 \text{ or } M, o, s \models \varphi_2 \\ M, o, s \models \neg\varphi & \Leftrightarrow & M, o, s \not\models \varphi \\ M, o, s \models \langle A \rangle \varphi & \Leftrightarrow & M, o', s \models \varphi, \text{ where } o' = K(o)(A) \neq \bot \\ M, o, s \models \exists_{x=t}(\varphi) & \Leftrightarrow & M, o, s\{x \mapsto o'\} \models \varphi, \text{ where } o' = [\![t]\!]_{M,o,s} \end{array}$$

where $s\{x \mapsto o'\}$ denotes the function that behaves like s except for the input x for which it yields the output o'.

We define $M, o \models \varphi$ if for all assignments s it holds that $M, o, s \models \varphi$. Additionally, we have $M \models \varphi$ if $M, o \models \varphi$ holds for all objects o. Finally, $\models \varphi$ holds if $M \models \varphi$ for all Kripke models M.

In standard modal logic with deterministic Kripke models, there is only a subtle difference between the interpretation of the possibility and the necessitation operator: The possibility operator requires *exactly one* accessible world to exist, while in the case of the necessitation operator it is required that *at most one* accessible world exists. Here, we have chosen for the *possibility* interpretation of the navigation operator; the *necessitation* reading of the operator would be:

$$M, o, s \models [A]\varphi \;\; \Leftrightarrow \;\; \text{if } o' = K(o)(A) \neq \bot \text{ then } M, o', s \models \varphi.$$

Thus, the difference is that in the latter reading the formula is also true if there is no accessible state. Note that $\models [A]\varphi \leftrightarrow (\langle A \rangle \varphi \vee A = nil)$. An alternative to dealing with nil pointers is to use a three-valued logic, which besides the truth values true and false includes a value for undefinedness, but this would unnecessarily complicate the technical treatment. Note that in the evaluation of $\exists_{x=t}(\varphi)$ we do allow t to be a nil pointer.

Finally, we remark that in the evaluation of formulas, the roles of constants and variables are interchanged: The interpretation of variables remains *constant* during the evaluation of a formula, in other words, their interpretation is 'frozen', while the interpretation of constants *varies* with the current state of evaluation.

4 Decidability of Basic Navigation Logic

In this section, we show that the validity of formulas in $\mathcal{L}_0$ can be decided by means of a semantic tableau procedure.

Definition 4. *Given a set O of objects, we construct a* semantic tableau, *which is a tree-like structure with nodes of the form $o : \Gamma$, where $o \in O \setminus \{\bot\}$ and $\Gamma \subseteq \mathcal{L}_0$.*

The construction of a tableau involves three types of branch extension rules, *namely* conjunctive rules, disjunctive rules *and* navigation rules. *They are of the following general format, respectively:*

$$\frac{o : \Delta, \varphi}{o : \Delta, \Gamma} \qquad \frac{o : \Delta, \varphi}{o : \Delta, \Gamma \;\mid\; o : \Delta, \Gamma'} \qquad o : \Gamma \;\xrightarrow{A}\; o' : \Gamma'$$

The rules are as follows (we omit the context $\Delta \subseteq \mathcal{L}_0$):

$$\frac{o : \neg(\varphi_1 \vee \varphi_2)}{o : \neg\varphi_1,\ \neg\varphi_2} \qquad \frac{o : \exists_{x=t}\varphi}{o : x = t,\ \varphi} \qquad \frac{o : \neg\exists_{x=t}(\varphi)}{o :\ x = t,\ \neg\varphi} \qquad \frac{o : \neg\neg\varphi}{o : \varphi}$$

$$\frac{o : \varphi_1 \vee \varphi_2}{o : \varphi_1 \;\mid\; o : \varphi_2} \qquad \frac{o : \neg\langle A\rangle\varphi}{o : \langle A\rangle\neg\varphi \;\mid\; o : A = nil}$$

$$o : \Gamma \;\xrightarrow{A}\; o' : \Gamma_A$$

where $\Gamma_A = \{\varphi \mid \langle A\rangle\varphi \in \Gamma\}$ and o' is a fresh label.

Given an input formula $\varphi \in \mathcal{L}_0$, the construction of the tableau for φ then proceeds as follows:

(0) *Start with an initial tree consisting of the root node $o : \varphi$.*
(1) *If a conjunctive or disjunctive rule is applicable then apply it and goto* (1)*, else goto* (2)
(2) *If the navigation rule is applicable then apply it once for each attribute A that occurs in Γ such that Γ_A is non-empty and goto* (1)*, else terminate.*

The construction of the tableau *always* terminates. Formally, this can be shown by a straightforward induction on the length of the input formula φ.

Before considering the soundness and completeness of the tableau construction, let us briefly sketch how a model can be extracted from a tableau. A tableau consists of a number of branches; the branches with a consistent theory can be used in the model construction. The theory of a branch is given by the set of literals that occur on it, but as the interpretation of these literals is relative to the nodes of the branch, we need to replace them by absolute literals whose interpretation is node-independent. A model is then obtained from a consistent branch by identifying the nodes that are expressed to be equal in the theory; in other words, each world of the model represents an equivalence class of identified nodes. Although the branch is a tree structure, the corresponding model may contain cycles due to these identifications. Finally, the accessibility relation

and the valuation function of the model can be extracted from the branch in a straightforward manner.

The remainder of this section is devoted to an outline of the soundness and completeness proofs of the above semantic tableau method. From now on, we fix O, with $\bot \in O$, to be the set of objects used in the tableau construction.

First, we give a definition of terms and propositions, so-called *absolute terms* and *absolute propositions*, whose evaluation does not depend on the current object.

Definition 5. *The set $\mathcal{T}_{abs}$ of* absolute terms *and the set $\mathcal{P}_{abs}$ of* absolute propositions *are given by:*

$$\mathcal{T}_{abs} = Var \cup O \cup \{o.A \mid o \in O,\ A \in \mathcal{A}\}$$

$$\mathcal{P}_{abs} = \{o.p \mid o \in O,\ p \in \mathcal{P}\}$$

An absolute literal *is an (in)equation between terms of $\mathcal{T}_{abs}$ or (the negation of) an absolute proposition of $\mathcal{P}_{abs}$.*

Next, we define the interpretation of absolute terms and propositions.

Definition 6. *Given a Kripke model $M = (O', K, \pi)$, a corresponding assignment s, and a strict (with respect to the respective bottom elements $\bot \in O$ and $\bot' \in O'$) mapping $\theta \in O \to O'$, we define the* interpretation $[\![t]\!]_{M,s,\theta}$ of an absolute term t *as follows:*

$$\begin{array}{ll} [\![x]\!]_{M,s,\theta} & = s(x) \\ [\![o]\!]_{M,s,\theta} & = \theta(o) \\ [\![o.A]\!]_{M,s,\theta} & = K(\theta(o))(A) \end{array}$$

Additionally, for all $t_1, t_2 \in \mathcal{T}_{abs}$ and $o.p \in \mathcal{P}_{abs}$ we define:

$$\begin{array}{lll} M, s, \theta \models (t_1 = t_2) & \Leftrightarrow & [\![t_1]\!]_{M,s,\theta} = [\![t_2]\!]_{M,s,\theta} \\ M, s, \theta \models o.p & \Leftrightarrow & p \in \pi(\theta(o)) \end{array}$$

Note that the strictness of θ yields $[\![\bot]\!]_{M,s,\theta} = \bot'$.

The following definition associates with each $o \in O \setminus \{\bot\}$ an operation that transforms a term $t \in \mathcal{T}$ into a corresponding absolute term.

Definition 7. *For each term $t \in \mathcal{T}$ and object $o \in O \setminus \{\bot\}$ we define the term $t^o \in \mathcal{T}_{abs}$ by:*

$$\begin{array}{ll} nil^o & = \bot \\ self^o & = o \\ A^o & = o.A \\ x^o & = x \end{array}$$

The following lemma states some truth-preserving properties of this operation.

Lemma 1. *For every model $M = (O', K, \pi)$, corresponding variable assignment s, and strict mapping $\theta \in O \to O'$ we that have for every term $t \in \mathcal{T}$:*

$$[\![t]\!]_{M,\theta(o),s} = [\![t^o]\!]_{M,s,\theta}$$

Similarly, for every equality $(t_1 = t_2) \in \mathcal{L}_0$ *and every proposition* $p \in \mathcal{P}$ *we have:*

$$M, \theta(o), s \models (t_1 = t_2) \text{ iff } M, s, \theta \models (t_1^o = t_2^o)$$
$$M, \theta(o), s \models p \quad \text{iff } M, s, \theta \models o.p$$

Next, we define the deductive closure of a set of absolute literals.

Definition 8. *For each set* Γ *of absolute literals, its* deductive closure $Clos(\Gamma)$ *is defined to be the smallest set that contains* Γ *and that is closed under the following rules:*

- $(t = t) \in Clos(\Gamma)$*, for every term* t *occurring in* Γ
- *if* $(t_1 = t_2) \in Clos(\Gamma)$ *then* $(t_2 = t_1) \in Clos(\Gamma)$
- *if* $(t_1 = t_2),\ (t_2 = t_3) \in Clos(\Gamma)$ *then* $(t_1 = t_3) \in Clos(\Gamma)$
- *if* $t_1 = t_2,\ \varphi \in Clos(\Gamma)$ *then* $\varphi[t_2/t_1] \in Clos(\Gamma)$

It is not difficult to see that if Γ is a finite set of absolute literals then $Clos(\Gamma)$ is also finite.

A *branch* T of a semantic tableau is a substructure that contains the root node and that for each application of a conjunctive rule contains its child, for each application of a disjunctive rule contains precisely one of its children, and for each application of the navigation rule contains all its children. The theory of a tableau branch as a deductively closed set of absolute literals, is introduced in the following definition.

Definition 9. *The* theory of a branch T *is given by* $Th(T) = Clos(\Gamma)$,

$$\begin{aligned} \textit{where } \Gamma \;=\; & \{o.p \mid o : p \in T\} \cup \{\neg o.p \mid o : \neg p \in T\} \cup \\ & \{t_1^o = t_2^o \mid o : (t_1 = t_2) \in T\} \cup \{\neg(t_1^o = t_2^o) \mid o : \neg(t_1 = t_2) \in T\} \cup \\ & \{o.A = o' \mid o \xrightarrow{A} o' \in T\} \cup \\ & \{\neg(o = \bot) \mid o \in T\}. \end{aligned}$$

(When we write $o : \varphi \in T$ *we mean that there exists a node* $o : \Delta, \varphi$ *in* T*, for some set of formulas* Δ*. Additionally, by* $o \xrightarrow{A} o' \in T$ *we mean that in* T *there exists an* A*-link from a node labelled by* o *to one labelled by* o'*.)*

It is worthwhile to observe that $Th(T)$ is a *finite* set of absolute literals. Next, we define when a tableau branch is closed, that is, does not give rise to a model.

Definition 10. *A branch* T *is called* closed *if* $Th(T)$ *is inconsistent (i.e., contains both a literal and its negation). If* T *is not closed it is called* open*.*

In the following definition, we introduce the notion of a homomorphic mapping of a branch of a semantic tableau into a Kripke model.

Definition 11. *A strict mapping* $\theta : O \to O'$ *is a* homomorphic mapping *of a branch* T *of a tableau into a model* $M = (O', K, \pi)$ *with respect to a variable assignment* s *if the following hold:*

– *if* $o \xrightarrow{A} o' \in T$ *then* $K(\theta(o))(A) = \theta(o')$
– *if* $o : \varphi \in T$ *then* $M, \theta(o), s \models \varphi$

Finally, we are in a position to state and prove the soundness of the tableau method.

Theorem 1. *For all* $\varphi \in \mathcal{L}_0$*, if* φ *is satisfiable then its tableau has an open branch.*

Proof. Without loss of generality we may assume that in φ each variable x is bound at most once and does not occur both free and bound. Let $M = (O', K, \pi)$ be a model and s a variable assignment such that $M, o', s \models \varphi$, for some o'. For an inductive argument, suppose we have constructed a variable assignment s and a corresponding homomorphic mapping $\theta : O \to O'$ into M of all the nodes of a branch of the tableau of φ which occur at a depth smaller than some $n \geq 0$. We consider the following two main extensions.

– The branch with node $o : \Delta, \exists_{x=t}(\psi)$ at level n is extended with $o : \Delta, x = t, \psi$. From the existence of the homomorphism θ we derive that $M, \theta(o), s \models \exists_{x=t}(\psi)$. Using the truth definition we obtain $M, \theta(o), s\{x \mapsto o'\} \models \psi$ for $o' = [\![t]\!]_{K,\theta(o),s}$. Thus, we have $M, \theta(o), s\{x \mapsto o'\} \models (x = t) \wedge \psi$. It follows by the above assumption on φ that x does not occur (free) in any formula occurring in the tableau other than ψ. So we have that θ is also a homomorphism into M with respect to the variable assignment $s\{x \mapsto o'\}$ and which now includes also the node $o : \Delta, x = t, \psi$.
– A node $o : \Gamma \in T$ on this branch at level n is A-linked to a new node $o' : \Gamma_A$, for some attribute A. From the assumption $M, \theta(o), s \models \Gamma$ we derive $M, o'', s \models \Gamma_A$ for $o'' = K(\theta(o))(A) \neq \bot$. So, we can extend θ to a homomorphism $\theta\{o' \mapsto o''\}$ which includes the new node $o' : \Gamma_A$.

Via Lemma 1 and the construction of the homomorphism θ we derive $M, s, \theta \models Th(T)$, for some branch T. From this it follows that $Th(T)$ is consistent, i.e., that T is an open branch. □

To prove completeness we first show how to construct a model from an open branch.

Definition 12. *Given an open branch* T*, we define the Kripke model* $M_T = (O', K, \pi)$ *as follows:*

– *The underlying set* O' *of worlds consists of the equivalence classes:*

$$[t] = \{t' \mid (t = t') \in Th(T)\},$$

where $t, t' \in \mathcal{T}_{abs} \setminus Var$*. The equivalence class* $[\bot]$ *plays the role of bottom in* O'*.*
– *The accessibility function* K *is defined by:*

$$\begin{array}{l} K([o])(A) = [o.A] \\ K([t])(A) = [nil], \text{ if there does not exist an } o \in [t]. \end{array}$$

– *The valuation function* π *is defined by:*

$$\pi([t]) = \{p \mid o : p \in T \text{ for some } o \in [t]\}.$$

It is straightforward to check that M_T is indeed well-defined. Moreover, by construction of M_T, an absolute term denotes in M_T its corresponding equivalence class under the strict mapping θ which assigns to every $o \in O$ its equivalence class $[o]$. This is expressed formally by the following lemma.

Lemma 2. *Let s be an assignment that assigns to each variable an object of M_T, i.e., an equivalence class of absolute terms. Additionally, let $\theta \in O \to O'$ be a strict mapping which assigns to every $o \in O$ its corresponding equivalence class $[o]$ in M_T. For each $t \in \mathcal{T}_{abs}$ let $[t]_s$ denote the equivalence class $[t]$ itself, in case $t \notin Var$, and $[x]_s = s(x)$, otherwise. For each term $t \in \mathcal{T}_{abs}$ we then have:*

$$[\![t]\!]_{M_T,s,\theta} = [t]_s$$

Given the above, we are now in a position to prove the completeness theorem.

Theorem 2. *For all $\varphi \in \mathcal{L}_0$, if the tableau for φ has an open branch then φ is satisfiable.*

Proof. Let T be an open branch of the tableau for φ. We show that for all $o : \psi \in T$ we have $M_T, [o], s \models \psi$ for the assignment s that is defined by: $s(x) = [t]$, where $(x = t) \in Th(T)$ and $t \in \mathcal{T}_{abs} \setminus Var$. Note that the existence of such an absolute term follows from the fact that the initial formula φ is assumed not to contain free variables. The proof proceeds by induction on the length of ψ. We treat the following characteristic cases.

- **case** p. From $o : p \in T$ and $o \in [o]$ we derive $M_T, [o], s \models p$.
- **case** $\neg p$. From $o : \neg p \in T$ and the fact that T is open we derive $o' : p \notin T$ for all $o' \in [o]$. In other words, $M_T, [o], s \models \neg p$.
- **case** $t_1 = t_2$. Suppose $o : (t_1 = t_2) \in T$. By definition of $Th(T)$ we have $t_1^o = t_2^o \in Th(T)$. By construction of the model M_T and the definition of s we derive $[t_1^o]_s = [t_2^o]_s$. From Lemma 2 we subsequently infer $M_T, s, \theta \models (t_1^o = t_2^o)$, where $\theta \in O \to O'$ is a strict mapping which assigns to every $o \in O$ its corresponding equivalence class $[o]$ in M_T. Finally, from Lemma 1 we conclude $M_T, [o], s \models (t_1 = t_2)$.
- **case** $\neg(t_1 = t_2)$. Suppose $o : \neg(t_1 = t_2) \in T$. By definition of $Th(T)$ we have $\neg(t_1^o = t_2^o) \in Th(T)$. Since $Th(T)$ is consistent we have $t_1^o = t_2^o \notin Th(T)$. By construction of the model M_T and the definition of s it thus follows that $[t_1^o]_s \neq [t_2^o]_s$, from which we derive by Lemma 2 that $M_T, s, \theta \models \neg(t_1^o = t_2^o)$, where $\theta \in O \to O'$ is a strict mapping which assigns to every $o \in O$ its corresponding equivalence class $[o]$ in M_T. From Lemma 1 we conclude $M_T, [o], s \models \neg(t_1 = t_2)$.
- **case** $\langle A \rangle \varphi$. Suppose $o : \langle A \rangle \varphi \in T$. From the construction of the tableau we derive that there exists a node o' with $o \xrightarrow{A} o' \in T$ and $o' : \varphi \in T$. The induction hypothesis yields $M_T, [o'], s \models \varphi$. The construction of the model M_T then yields $M_T, [o], s \models \langle A \rangle \varphi$.
- **case** $\exists_{x=t}(\varphi)$. Suppose $o : \exists_{x=t}(\varphi) \in T$. From the construction of the tableau we derive that also $o : x = t, \varphi \in T$. Applying the induction hypothesis we obtain $M_T, [o], s \models (x = t)$ and $M_T, [o], s \models \varphi$. From this we conclude $M_T, [o], s \models \exists_{x=t}(\varphi)$. □

Note that from the construction of the model in the completeness theorem it follows that the language $\mathcal{L}_0$ satisfies the *finite model property*, i.e., every satisfiable formula is satisfiable in a finite model. Moreover, note that the constructed model may contain cycles, which indicates that the language $\mathcal{L}_0$ does not satisfy the *tree model property*, stating that every satisfiable sentence has a model that is a tree of bounded branching [12]. A counterexample is the formula $\exists_{x=self}\langle A\rangle(x = self)$, which is only satisfiable in worlds that are A-linked to themselves.

5 Extensions

In this section, we briefly discuss some interesting decidable extensions of the basic navigation logic presented above.

5.1 Jump Operators

The logic $\mathcal{L}_1$ extends the basic navigation logic $\mathcal{L}_0$ with *jump operators*; i.e., we generalise the syntax of the navigation operator to $\langle t\rangle\varphi$, where t is a term in $\mathcal{T}$. So, if the index is a variable x, the operator can be used to jump back to the state to which x is bound.

Definition 13. *The* truth definition $M, o, s \models \varphi$ for the language $\mathcal{L}_1$ *is the same as for* $\mathcal{L}_0$ *except:*

$$M, o, s \models \langle t\rangle\varphi \;\Leftrightarrow\; M, o', s \models \varphi, \text{ where } o' = [\![t]\!]_{M,o,s} \neq \bot.$$

As an example, consider the object structure M in Figure 2. At the world p, the following formula is true:

$$\langle AuthorOf\rangle\exists_{x=self}\langle item(1)\rangle\exists_{y=PublishedBy}\langle x\rangle\langle item(2)\rangle(PublishedBy = y)$$

which means that person p is the author of two books that are published by the *same* company.

The tableau construction $\mathcal{L}_1$ is similar to that of $\mathcal{L}_0$ modulo some minor modifications.

Definition 14. *The* tableau rules for $\mathcal{L}_1$ *are the same as for* $\mathcal{L}_0$ *except for the navigation rule, which is generalised as follows:*

$$o : \Gamma \;\;\xrightarrow{t}\;\; o' : \Gamma_t$$

where $\Gamma_t = \{\varphi \mid \langle t\rangle\varphi \in \Gamma\}$ *and* o' *is a fresh label.*

Moreover, it suffices to modify the theory $Th(T)$ of a branch as given in Definition 9 as follows. The support set Γ of absolute literals additionally contains:

$$\{x = o' \mid o \xrightarrow{x} o' \in T\} \cup$$
$$\{\bot = o' \mid o \xrightarrow{nil} o' \in T\} \cup$$
$$\{o = o' \mid o \xrightarrow{self} o' \in T\}.$$

The soundness and completeness proofs are similar to the proofs for $\mathcal{L}_0$.

5.2 Navigation programs

In the extension $\mathcal{L}_2$, which is inspired by the dynamic logic of [8], we include formulas of the form $\langle \Pi \rangle \varphi$, where Π is a *navigation program* that defines a particular navigation strategy.

Definition 15. *The* navigation language $\mathcal{L}_2$*, consisting of boolean conditions* b*, navigation programs* Π *and formulas* φ*, is generated by the following BNF-grammar:*

$$b \ ::= p \mid t_1 = t_2 \mid \neg b \mid b_1 \vee b_2$$

$$\Pi ::= A \mid \Pi_1; \Pi_2 \mid \mathsf{if}\ b\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2 \mid \mathsf{while}\ b\ \mathsf{do}\ \Pi$$

$$\varphi \ ::= b \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists_{x=t}\varphi \mid \langle \Pi \rangle \varphi$$

To formalise the meaning of a formula in $\mathcal{L}_2$ we first define the meaning of navigation programs.

Definition 16. *Given a model* $M = (O, K, \pi)$ *and an assignment* s*, we have the following (standard)* denotational semantics *which assigns to each program* Π *a strict mapping* $\mathcal{M}(\Pi) : O \to O$*:*

$$\begin{array}{ll}
\mathcal{M}(A)(o) & = K(o)(A) \\
\mathcal{M}(\Pi_1; \Pi_2)(o) & = \mathcal{M}(\Pi_2)(\mathcal{M}(\Pi_1)(o)) \\
\mathcal{M}(\mathsf{if}\ b\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2)(o) & = \begin{cases} \mathcal{M}(\Pi_1)(o) & \textit{if } M, o, s \models b \\ \mathcal{M}(\Pi_2)(o) & \textit{otherwise} \end{cases} \\
\mathcal{M}(\mathsf{while}\ b\ \mathsf{do}\ \Pi)(o) & = \begin{cases} \mathcal{M}(\mathsf{while}\ b\ \mathsf{do}\ \Pi)(\mathcal{M}(\Pi)(o)) & \textit{if } M, o, s \models b \\ o & \textit{otherwise} \end{cases}
\end{array}$$

The above recursive definition of $\mathcal{M}$ can be justified by means of a (standard) least fixpoint construction defined in terms of the discrete complete partial order $(O, \sqsubseteq)$ that is defined by: $\bot \sqsubseteq o$ and $o \not\sqsubseteq o'$ for all distinct objects o and o'. From this fixpoint construction it follows that $\mathcal{M}(\mathsf{while}\ b\ \mathsf{do}\ \Pi)(o) = \bot$ in case the program $\mathsf{while}\ b\ \mathsf{do}\ \Pi$ does not terminate in o.

We have the following truth definition.

Definition 17. *The* truth definition $M, o, s \models \varphi$ for the language $\mathcal{L}_2$ *is similar to that for* $\mathcal{L}_0$ *except:*

$$M, o, s \models \langle \Pi \rangle \varphi \ \Leftrightarrow \ M, o', s \models \varphi, \ \textit{where } o' = \mathcal{M}(\Pi)(o) \neq \bot$$

It is worthwhile to observe that the truth of the formula $\langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle true$ implies that the program $\mathsf{while}\ b\ \mathsf{do}\ \Pi$ terminates. In the light of the Halting Problem, however, this does not imply that we cannot decide the validity of formulas in the language $\mathcal{L}_2$, because our navigation programs are not Turing complete.

As an example of $\mathcal{L}_2$, the formula

$$\langle \mathsf{while}\ \neg(y = \mathit{self})\ \mathsf{do}\ A \rangle true$$

states that the object denoted by y is reachable from the current object by a finite chain of A-links.

Definition 18. *The* tableau rules for $\mathcal{L}_2$ *are given by the rules for* $\mathcal{L}_0$ *together with the following rules (we omit the context* Δ*):*

$$\frac{o : \langle \Pi_1; \Pi_2 \rangle \varphi}{o : \langle \Pi_1 \rangle \langle \Pi_2 \rangle \varphi}$$

$$\frac{o : \langle \mathsf{if}\ b\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2 \rangle \varphi}{o : b, \langle \Pi_1 \rangle \varphi \ \mid\ o : \neg b, \langle \Pi_2 \rangle \varphi} \qquad \frac{o : \neg \langle \mathsf{if}\ b\ \mathsf{then}\ \Pi_1\ \mathsf{else}\ \Pi_2 \rangle \varphi}{o : b, \neg \langle \Pi_1 \rangle \varphi \ \mid\ o : \neg b, \neg \langle \Pi_2 \rangle \varphi}$$

$$\frac{o : \langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \varphi}{o : \neg b, \varphi \ \mid\ o : b, \langle \Pi \rangle \langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \varphi} \qquad \frac{o : \neg \langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \varphi}{o : \neg b, \neg \varphi \ \mid\ o : b, \neg \langle \Pi \rangle \langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \varphi}$$

The resulting tableau method may give rise to *non-termination*. However, in the tableau construction, we can stop applying any further rules to a leaf $o : \Gamma$ in case there already exists an ancestor node with the same set of formulas Γ. With this additional rule the method is guaranteed to terminate, because starting from an initial formula, only a finite number of *different* formulas can be generated.

Additionally, a branch T of a tableau now also closes if one of its leafs $o : \Gamma$ contains a formula of the form $\langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \psi$. (Note that by definition there also exists an ancestor node of the form $o' : \Gamma$.) The new tableau method is sound and complete because in case its theory does not contain an inconsistency, such a branch T corresponds to a model in which the program $\mathsf{while}\ b\ \mathsf{do}\ \Pi$ 'loops', i.e., $\langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \psi$ does not hold. For the same reason, a branch T of a tableau with a consistent theory and a leaf which contains a formula of the form $\neg \langle \mathsf{while}\ b\ \mathsf{do}\ \Pi \rangle \psi$ does constitute a model.

6 Conclusions and Future Research

In this paper, we have presented multimodal logics for navigation across object structures. The starting point of these logics is the modelling of object structures at the implementation level as Kripke models that contain a family of deterministic accessibility relations, namely one relation for each pointer attribute. The logics cover a variable-binding mechanism that allows the binding of logical variables to objects that are reachable by a pointer. In this way, pointer attributes can be used in the logic both as first-order terms in equalities and as modal operators. The main result of the paper is a tableau construction for deciding the validity of formulas in these navigation logics.

In [1], it is stated that for hybrid languages, i.e., modal logics that include mechanisms for naming the worlds of a given Kripke model: "it seems unlikely that restricted forms of (label) binding will lead to decidable systems." However, in this paper, we have shown that decidable hybrid languages can be obtained by restricting to particular classes of models. This point is also illustrated in [3], where decidability of a hybrid language is obtained for the class of strict partial orders. In [11], a decision procedure based on a tableau construction is given for hybrid logics that do not involve variable-binding mechanisms. Of interest in this context would be an extension of our tableau construction to hybrid languages that do include variable-binding.

In the graphical modelling language UML, class diagrams can also be annotated with so-called *constraints*, which are formulated in the Object Constraint Language,

OCL for short [13]. The language OCL is a textual language for the description of object structures mainly at the modelling level. In contrast to our approach, in OCL, navigation is modelled as a *dereferencing operator* that is applied to first-order terms. For instance, the term $t.A$ denotes the value of the pointer attribute A of the object denoted by t. By means of a formalisation of navigation in terms of a modal logic, however, we are able to identify decidable navigation logics that are still expressive enough to express interesting properties of object structures. Future work concerns an extension of our approach to the modelling level of class diagrams and the development of tools for computer-aided-verification by means of an implementation of the corresponding tableau procedure.

At the implementation level another interesting line of future work concerns an application of our decidable navigation logics to the computer-aided-verification of the correctness of object-oriented programs. Such an application involves the definition of a *weakest precondition calculus* [4] for our navigation logics.

Acknowledgements

The authors would like to thank the anonymous referees for their comments.

References

1. C. Areces, P. Blackburn, and M. Marx. A road-map on the complexity of hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic, Proceedings of CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, pages 307–321. Springer-Verlag, Heidelberg, 1999.
2. P. Blackburn and J. Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4:251–272, 1995.
3. P. Blackburn and J. Seligman. What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic'96*. CSLI Publications, Stanford, California, 1997.
4. F.S. de Boer. A WP-calculus for OO. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS'99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 135–149, 1999.
5. R.M. van Eijk, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Modal logic with bounded quantification over worlds. *Journal of Logic and Computation*, 2001. To appear.
6. M. Fitting and R.L. Mendelsohn. *First-Order Modal Logic*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.
7. A. Hamie, J. Howse, and S. Kent. Navigation expressions in object-oriented modelling. In *Fundamental Approaches to Software Engineering, Proceedings of FASE'98*, volume 1382 of *Lecture Notes in Computer Science*, pages 123–137. Springer-Verlag, Heidelberg, 1998.
8. D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1979.
9. G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic*. Methuen and Co. Ltd, London, 1968.
10. J. Rumbauch, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1999.

11. M. Tzakova. Tableau calculi for hybrid logics. In N.V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, Proceedings of TABLEAUX'99*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 278–292. Springer-Verlag, Heidelberg, 1999.
12. M.Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. AMS, 1997.
13. J.B. Warmer and A.G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, Reading, Massachusetts, 1998.