A METHOD FOR DESIGN REASONING USING LOGIC: FROM SEMANTIC TABLEAUX TO DESIGN TABLEAUX

Hendriks, Lex; Kazakci, Akin Osman

ABSTRACT
Inspired by C-K theory, the current work presents a framework demonstrating how C-K type design reasoning can be applied within logic. Building on our previous work, we extend and generalize the well-known method of Semantic Tableaux, invented by Beth for logical theorem-proving, to Design Tableaux – a general, formal procedure allowing to implement expansive reasoning within the formalism of logic. Our contribution is twofold. First, we give a formal, verifiable procedure that explicit and apply the ill-defined operators of C-K theory. Second, we contribute to the notion that design science can be useful to other fields and theories (in this case, logic) by proposing a mode of creative reasoning within a logical framework stemming directly from a theory of design

Keywords: design, logic, creativity, C-K theory, semantic tableaux

1 INTRODUCTION
While there are extensive studies on *modeling* and *theorizing* about design (see e.g. [1-10]), most engineers still perceive design as *practice* or even *art*. Considering the large variety of *design practices*, it is arguably difficult to determine a fixed universal set of core characteristics and phenomena defining *design science*. This difficulty hinders the maturation of design as a scientific field of study. Indeed, what is scientific about design? One possible answer is the specific mode of reasoning design requires. Design is a reasoning process by which new descriptions for new classes of objects are generated. Thus, the properties of this creative process constitute a core phenomenon for design research.
Yet, traditional formal theories and models of design usually treats creativity as external phenomena, for example, due to imagery (see e.g. [11, 12]). A first category of models aim at *structuring the process* (e.g. [13]), and hence, they provide models for the *organization of design activities*. A second category describe design as the *progressive refinement* of the object being designed (e.g. [14] or [7]), and hence, propose a model of the *evolution of design artifacts.* In both cases, the creative mechanisms of design reasoning are not deemed as central (i.e. the models do not seek to explain where do refinements come from or what sort of organizing allows better creativity).
One notable exception to these works is the concept-knowledge (C-K) theory of design [15-17]. C-K theory places the creative generation of new definitions and objects at the heart of design through a notion of conceptual expansion. It describes design as a reasoning process where object's identity are attempted to be *revised* by changing their definitions through the introduction of new properties. If successful, a new class of objects has been designed. According to the theory, design reasoning is triggered by conceptual expansions (introduction of new properties to design concepts), which, in turn, provoke the expansion of knowledge (learning about the new type of object). This expansive reasoning and rationality is claimed to be the central mechanism of design [16, 17], and understanding it fully is necessary for a foundation of design science [18].
C-K theory and derived methods have seen many successful industrial applications [19, 20]. The theory has also contributed to different literatures ranging including management, economy, philosophy of mathematics, design education and creativity methods [19-25]. This demonstrates authentic and well-established design theories would be useful for other fields as well.
Contrastingly, although C-K theory is a theory of reasoning, *its formal and logical aspects have yet to be explored in depth*. Despite the fact that, in its various presentations, C-K theory refers to many formal theories and approaches (like Set Theory [16], Category Theory [26], Forcing [18]), *a complete formalization of the theory is still an open issue* after more than twelve years since its introduction in [15]. In particular, although many notions of the theory have been presented using ideas from logic

and mathematics, the mechanisms of expansive reasoning is not presently described. Although C-K theory emphasize in many occasions [16-18, 26] that design as a specific reasoning requires the use of operators different than standard algorithms or choice procedures, those operations have never been defined in a formal way.

Inspired by C-K theory, the current work presents a framework demonstrating how C-K type design reasoning can be applied within logic. Building on our previous work [27, 28], we extend and generalize the well-known method of Semantic Tableaux, invented by Beth [29] for logical theorem-proving, to Design Tableaux – a general, formal procedure allowing to implement expansive reasoning for logical databases. Starting with some facts (knowledge) and formulas to be proven, a semantic tableaux is constructed step by step, by applying *logical rules of reasoning* (see Figure 2 in section 4). In a design tableau *extra-logical rules*, like adding new knowledge or extending the design concept, can be applied to find a proof (or a counter-model) for a design concept. When used within the framework of a constructive logic, such as the Intuitionist Logic we are using, the proof can be interpreted as a *method of construction* returning a witness (or, in terms of design, a *prototype*). The method has already been presented at [30]; the current paper is the first written presentation. Our contribution is twofold. First, we give a formal, verifiable procedure that explicit and apply the ill-defined operators of C-K theory. This opens up perspectives of automating, at least partially, C-K type design reasoning, along with many research questions. Second, we contribute to the notion that design science can contribute to other fields and theories (in this case, logic) by proposing a mode of creative reasoning within a logical framework stemming directly from a theory of design – and not the other way around as it is usually the case in design research.

The plan of the paper is as follows. Section 2 gives a summary of C-K theory and previous work on its formal foundations. Section 3 introduces the basic logical framework we use. *Design stages* and *design concepts* and *body* are defined and some of their properties are summarized. Section 4 introduces semantic tableaux and then extends it to Design Tableaux. Since our work is a contribution both to design and to logic, part of the text is necessarily technical. We use a quite detailed example in this section to illustrate some of the ideas being presented. Section 5 concludes with further remarks.

2 A CONCEPT-KNOWLEGDE PERSPECTIVE ON DESIGN REASONING

C-K theory describes design reasoning based on the distinction and interaction between two spaces;

- **Knowledge space** A knowledge space represents all the knowledge available to a designer (or to a group of designers) at a given time. These are propositions that the designer is capable of declaring as true or false; i.e., propositions whose logical status are known to the designer (e.g., some tires are made of rubber).
- **Concept space** A concept space represents propositions whose logical status are unknown and cannot be determined with respect to a given knowledge space. These are propositions that can be stated as neither true, nor false by the designer at the moment of their creation (e.g., some tires are made of dust).

Concepts are descriptions of an object of the form "C: there exist an object x with the properties p_1, p_2,..., p_n" such that C is *undecidable* with respect to current K.

According to C-K theory, creative design begins by a conceptual expansion that forms a concept. A novel and unusual property is added to a concept to form a new concept (e.g. tires for life). The elaboration of concept can then be continued either by further expansions (tires for life are made of silicon) or by restrictions (that is by adding usual properties of the initial concept, e.g. tires for life are round). Conceptual expansions or restrictions are called partitioning in C-K theory.

When elaborating a concept space, a designer might use his or her K space, either to partition further the concepts, or to attempt a validation of a given concept. This last type of operation is called K-validation and it corresponds to the evaluation of a design description using knowledge. The result of a K-validation is positive, if the designer acknowledges that the proposition "there exist an object x with properties p_1,p_2,...,p_n" is true. The result is negative, if the knowledge available to the designer allows him to state that the proposition is false.

Often the validation of a concept is not readily possible. In order to validate concepts, new knowledge warranting the existence conditions of such an object should be acquired. In terms of C-K theory, knowledge should be expanded. The expansion of knowledge space is called K-expansion. The central

proposition of C-K theory is thus “design is the interaction and dual expansions concepts and knowledge” [15-17, 26]

Previous work on formal aspects of C-K theory
Despite the mathematical references and metaphors used in the presentation of the theory, a full mathematical presentation of the theory has not been provided to date. Nevertheless, some steps for formalizing C-K theory have been taken in some recent work. Hatchuel and Weil [18] argues that there are significant similarities between the type of reasoning described by C-K theory and Forcing, a technique used in Set Theory for constructing alternative set theoretic models with desired properties. It is claimed that the parallel between Forcing and C-K theory is an important step for design theory in general but this issue needs more formal investigation. In a complementary approach, Kazakci *et al.* [31] shows that C-K type reasoning can be implemented with much more simple formalisms. They use propositional term logic to model the basic ideas of C-K theory. They suggest a notion of “models of K space” to emphasize that different knowledge structures (or formalism) used to model knowledge will yield different conceptive power and degrees of flexibility in reasoning. [27] presents a first-order logical formal account of C-K theory’s core notions. To emphasize the constructive aspects of a design process, he uses Intuitionist logic. He considers the *interaction of concepts and knowledge* providing the necessary definitions that allows the interaction and expansion of the corresponding spaces. Hendriks and Kazakci [28] builds on this work and complements it in that we consider the core proposition of the theory - the *dual expansion of concepts and knowledge* and investigate the logical implications of such a principle. Next section builds on that framework to provide the necessary basis for C-K type reasoning with Design tableaux.

3 A LOGICAL FRAMEWORK FOR THE EVOLUTION OF CONCEPTS AND KNOWLEDGE

The basic notions of our formal framework presented in [28] are the *body-of-knowledge*, the *design concept* and the *design stage*. The *design concept* is a description of the artifact one is seeking to construct, a kind of design brief. As all statements in our formal framework, this description is formulated in first-order logic.
The *body-of-knowledge* is a description of the knowledge we think is relevant for the realization of the design concept. In the formal framework, the body-of-knowledge is a set of statements, again in first-order logic. Let K be a body-of-knowledge and C a design concept, than we define the pair $<K; C>$ as a *design stage*. The idea is that in $<K; C>$, the K represents our current knowledge (which we hold to be relevant) and C the current design concept.
Representing a design stage as a pair of descriptions, a description of the specifications and a description of knowledge or structures is, either explicitly or implicitly, quite common in descriptions of the design process (see e.g. [5, 11, 14]). Usually, those pairs focus on some distinction such as function and form, whereas, here, all definitionary elements are represented in C and body-of-knowledge K is a kind of resource for evolving the design concept.

3.1 The logic

The language of our formal framework for design is the language of first-order logic. The basic building blocks, the atomic formulas, of this language are *predicates*, symbolic expressions to state properties and relations, like ‘x is a man’ or ‘John is the father of y’. These atomic formulas, like ‘$M(x)$’ or ‘Father of(j, y)’, are build up from predicate symbols, like M and Father of, variables, like x and y and constants, like j. More complex formulas can be built using the following set of rules:

- If A is a formula, so are $\neg A$, $\Box xA$ and $\Box xA$
- If A and B are formulas, so are $A \wedge B$, $A($$B$ and $A \rightarrow B$

These formulas enable us to encode rather complex statements like ‘For every one, being man or woman implies having a father and mother and noone is the father of all’ as: $\Box x(Man(x) \wedge Woman(x) \rightarrow \Box y$Father of$(y, x) \wedge \Box z$Mother of$(z, x)) \wedge \neg \Box x \Box y$Father of$(x, y)$. Formulas like the above, not containing any *free* variables (i.e. not bound by a quantifier) are called *sentences*.
Recall from the introduction that the logic underlying our formal framework is constructive.

To state the rules for derivability (|-) in this logic, we will use capital letters near the end of the alphabet to designate sets of formulas and those at the beginning for formulas. In stead of $T \cup \{A\}$ for the union of the set of formulas T with the singleton set containing formula A, we will often write simply T,A. Likewise we use S, T for the union of S and T.

Structural Rules

(Ax) $T,A \vdash A$

(W) $T \vdash A \Rightarrow S, T \vdash A$

(Cut) $T \vdash A$ and $T,A \vdash B \Rightarrow T \vdash B$

Propositional Rules

(∋) $T \vdash A$ and $T \vdash B$, $T \vdash A \ni B$

(() $T,A \vdash C$ and $T,B \vdash C$, $T,A(B \vdash C$

(→) $T,A \vdash B$, $T \vdash A \rightarrow B$

(⊥) $\bot \vdash A$

Quantification Rules

(□) $T \vdash A(a)$, $T \vdash \Box x A(x)$ (in the $\Rightarrow$ direction: a not occuring in T or $A(x)$)

(□) $T,A(a) \vdash B$, $T, \Box x A(x) \vdash B$ (in the $\Leftarrow$ direction: a not occuring in T, $A(x)$ or B)

As usual we define $\neg A = A \rightarrow \infty$, where ∞ is the propositional constant for contradiction (*falsum*), from which by the ∞-rule everything can be derived.

In the quantification rules we assume the usual requirement that x can be substituted in $A(x)$, without spelling out the technical details. The logic defined above is the *intuitionistic predicate logic* , the best known constructive logic. The similarities between intuitionism and C-K theory have been further investigated in [27, 32]. The advantage of using a constructive logic in our context is that proofs can be interpreted as methods of construction of a prototype [27, 32]. Adding the axiom $\vdash A(\neg A$ or the rule $T \vdash \neg \neg A \ ) \ T \vdash A$ would turn the logic into the *classical predicate logic*.

3.2 Design stages and design moves: evolving C and K with sound design steps

Our definition of design reasoning is based on the one in C-K theory, where design is defined as a *reasoning* activity, starting with a proposition of the design concept and proceeding by *operators* adding knowledge or expanding the design concept. The definition below is slightly more abstract, which would seem to allow more design stages and design steps than described in C-K theory itself.

Definition 1 *A* design stage *is a pair* $s = \langle K; C \rangle$*, where K a finite set of sentences, called the* body-of-knowledge *of s and C a sentence, called the* design concept *of s.*

A design stage $\langle K; C \rangle$ *is called* consistent *if* $K \not\vdash \neg C$.

A design stage $\langle K; C \rangle$ *is called* open *if* $K \not\vdash C$(¬*C.*

A design step *is a pair* ($s0$, $s1$) *where* $s0$ *and* $s1$ *are design stages. We will use the notation* $s0 \Rightarrow s1$ *for the design step* ($s0$, $s1$)*. Usually we will also assume that* $s0 = \langle K0; C0 \rangle$, $s1 = \langle K1; C1 \rangle$ *etc. A design step* $s0 \Rightarrow s1$ *is called* sound, *if* $s1$ *is consistent and for all* $A \in K0$ *it is true that* $K1 \vdash A$ *and* $K1,C1 \vdash C0$.

Design step $s0$ implies $s1$ *if* $\bigcap K0 \rightarrow C0 \vdash \bigcap K1 \rightarrow C1$. *Design step* $s0$ *is* equivalent *with* $s1$ *if* $s0$ *implies* $s1$ *and* $s1$ *implies* $s0$*. We will use the notation* $s0 \vdash s1$ *if* $s0$ *implies* $s1$ *and* $s0 \equiv s1$ *when they are equivalent.*

Note that in the definition above there are no constraints on the type of sentence used as the design concept. Our C does not necessarily have the form $\Box x.P0(x) \ni \ldots \ni Pn(x)$, where the Pi are the desired properties for the object that we wish to come out of the design process. It is even not required that C is an existential formula. One can imagine for example that the 'thing' we try to design is a way of transforming all x with property A into some y with relationship R between the x and the y. So $C = \Box x(A(x) \rightarrow \Box y R(x, y)$ would be a conceivable design concept. This is more general than C-K theory.

That we assume the body-of-knowledge K to be finite is not a real restriction in practice (at any moment of time each finite group of people could only be aware of a finite number of facts). We could allow for infinite K in principle, but this would slightly complicate our formulas like in the definition of $s0 \vdash s1$, where for an infinite K the conjunction $\cap K$ formally is not defined. The following facts follow directly from our definitions.

Facts 2 *Let s*0 = <*K0; C0*>, *s*1 = <*K*1; *C*1> *and s*2 = <*K*2; *C*2> *be design stages.*
*1. If s*0 *is consistent then K*0 *is consistent (i.e. K*0|/-∞*).*
*2. If s*0 ⇒ *s*1 *is sound and K*0 ⊆ *K*1 *then s*0 *is consistent.*
*3. If s*0 ≡ *s*1 *and s*1 (*or s*0) *is consistent then s*0 ⇒ *s*1 *is sound.*
*4. If s*0 ⇒ *s*1 *and s*1 ⇒ *s*2 *are sound, so is s*0 ⇒ *s*2.

Our definition of sound design steps is one way of formalizing the intuitive notion of one design stage 'implying' another, found in most descriptions of design (i.e. in [17] and [7]). The informal notion is sometimes used in a loose sense of 'having some reason to go from *s*0 to *s*1'. On other occasions the 'implication chain' is a series of stages where some form of reasoning is involved. Our notion of soundness is a weak form of such a logical connection, whereas the defined implication (*s*0 |- *s*1) is rather strong.
Two special cases of sound design steps may clarify the often-observed difference in direction of the 'implication' between 'refining the specifications' and 'refining the knowledge'. Note that if *K*0 = *K*1 and *s*0 ⇒ *s*1 is sound, then *K*0,*C*1 |- *C*0 and hence *s*1 |- *s*0. On the other hand, if *C*0 = *C*1 and *s*0 ⇒ *s*1 is sound, *K*1 |- ⋂*K*0 and hence *s*0 |- *s*1.

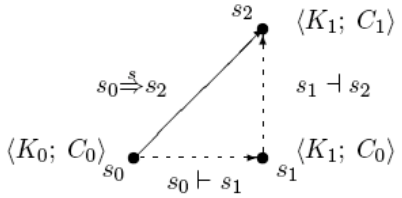


Figure 1 Splitting design steps in a K- and a C-component

As long as we confine ourselves to sound design steps that only change the design concept or only the body-of-knowledge (which theoretically we could always do by splitting up steps if necessary, see figure 1), we could use the implication of one stage by another as a basis for describing the design process. Simple examples of sound steps are:

- (Adding knowledge) <*K; C*> ⇒ <*K,A; C*>
- (Adding properties) <*K*; □*xC*(*x*)> ⇒ <*K*; □*xC*(*x*)∋*P*(*x*)>
- (Introducing a definition) <*K*; *C*∋*D*> ⇒ <*K*, *P* ↔*C*∋*D*; *P*>

An example of *s*0 |- *s*1 where *s*0 ⇒ *s*1 is not sound would be *s*0 = <*K; C*> and *s*1 = <*K*; *C(D*>. If *K*|/- *D* → *C* then *K* together with *C* (*D* does not imply *C*. As can be concluded from the examples above, our formalism does not require the design concept to be of the form □*xC*(*x*) by contrast to original C-K theory. Allowing the full generality of a first-order sentence provides us with a lot more flexibility as we will show in the sequel.
Since we are using a constructive derivability operator (i.e. each proof is a method), the following theorem shows that, when we reach a closed design stage, we have a *constructive proof of the original design concept*.

Theorem 3 *Let sn* = <*Kn*; *Cn*> *be a closed design stage reached after n sound design steps from s*0 = <*K0; C0*> (*so Kn* |- *Cn*) *then Kn* |- *C*0.
Proof. From the fact 2.3 we can conclude that *s*0 ⇒ *sn* is sound. Hence, by definition, *Kn,Cn* |- *C*0. As *Kn* |- *Cn* by the Cut-rule it follows that *Kn* |- *C*0.

Although such properties can be *intuited* from the descriptions of original C-K theory, our framework allows *proving* them.

4 REASONING WITH TABLEAUX: LOGICAL REASONING VS. DESIGN REASONING

4.1 Semantic tableaux

For the construction of a proof there is a technique in logic called *Beth tableaux* [29] (also known as *semantic tableaux* or *semantic trees*). We will first define the tableau method for the basic logic, to extend the method later on for *design tableaux*. The principle of the semantic tableau is a stepwise systematic search for a proof, say for *T* |- *A*, starting from a root node *T* • *A*. Each step takes one node into one or two new nodes (so in the process the 'branch' may split). In each step one formula (on the right or the left side of the • , is used to produce one or two formulas, which may be put on the right or the left side of the • in the new node(s). Often the rules allow us to skip the 'treated' formula, to prevent unnecessary repetitions. In general both the right side and the left side of a node in a semantical tree may contain one or more formulas and may even be empty. To understand this procedure it may be helpful to consider the formulas on the right hand side as 'true' and those on the left hand side as 'false'. So if *A*∧*B* is on the right, also *A* and *B* will be true and in a new node may be added on the right hand side. Figure 2 gives the semantic tableau rules.

(∧L)	$L, A \wedge B \bullet R$	⇒	$L, A, B \bullet R$	
(∧R)	$L \bullet A \wedge B, R$	⇒	$(L \bullet A, R \text{ and } L \bullet B, R)$	
(∨L)	$L, A \vee B \bullet R$	⇒	$(L, A \bullet R \text{ and } L, B \bullet R)$	
(∨R)	$L \bullet A \vee B, R$	⇒	$L \bullet A, B, R$	
(→L)	$L, A \rightarrow B \bullet R$	⇒	$(A \rightarrow B, L \bullet R, A \text{ and } L, B \bullet R)$	
(→R)	$L \bullet A \rightarrow B, R$	⇒	$L, A \bullet B$	
(¬L)	$L, \neg A \bullet R$	⇒	$L \bullet R, A$	
(¬R)	$L \bullet \neg A, R$	⇒	$L, A \bullet$	
(∀L	$L, \forall x A(x) \bullet R$	⇒	$\forall x A(x), L, A(d) \bullet R$	$(d \text{ exists})$
(∀R	$L \bullet \forall x A(x), R$	⇒	$L \bullet A(d)$	$(d \text{ new})$
(∃L	$L, \exists x A(x) \bullet R$	⇒	$L, A(d), R$	$(d \text{ new})$
(∃R	$L \bullet \exists x A(x) \bullet R$	⇒	$L \bullet A(d), R, \exists x A(x)$	$(d \text{ exists})$

Figure 2. The rules of semantic tableau

The way the rules are provided above suggest that formulas nearest to the • sign are treated first and formulas that need repeating are moved outward. If we add some further rules, like also moving outward atomic formulas, for which there is no rule, and decide for example to always first try to treat the formulas on the right, one can imagine the rules turn almost into a algorithm breaking down the formulas in the root node into atomic ones. The goal of the semantic tableau is to finding closed nodes.

Definition 4 *A node L • R is* closed *if L\R is not empty. A tree is called closed if all branches end in a closed node.*

If *L • R* is closed, then there is an *A* in both *L* and *R*, so *L* |- *A* and hence *L* |- ∪*R*. The rules of the semantic tableau are such that if the subtree below a node *L • R* is closed, *L* |- ∪ *R*. So if the tree below the root *T • A* is closed, than *A* is derivable from *T* (*T* |- *A* and in fact the tree shows a derivation).
The fact that there may be formulas introducing new constants that keep needing repetition makes that in the case of first-order logic, unlike in propositional logic, no such procedure as sketched above, can exist that guarantees that for each root *T • A* we can either find a closed tableau (and hence prove *T*|/- *A*) or, after a finite number of steps, decide that no further rules can be applied (in which case one can also construct a model from the 'open' tableau showing that *T* |/- ¬*A*). Such an algorithm is excluded by the undecidability of first-order logic.
The tableau rules for the basic logic already correspond to (logical) design steps, like:

- introducing a new name: *Let's call this thing we are looking for a zippahula* (□L, □R)
- splitting up design tasks (^R, →L)
- drawing conclusions from knowledge we have (^L, □R)
- making implementation decisions (∨L)
- introducing constraints or hypotheses in the body-of-knowledge (→R)

The following theorem for semantic tableaux will be useful when we later on define the *design tableaux.*

Theorem 5 *A semantic tableau with top node L • R can be closed by applying the tableau rules if and only if L |- ∪ R.*

The fact that the Beth tableau for a design stage *s, K • C(¬C* cannot be closed tells us that *s* is *open* (as in definition 1). To model the design process at a stage where all design stages are still open, we need to construct a model *M* where actually *M |= K* and *M |/= C* and *M |/= ¬C*. In fact open Beth tableaux already contain all the information to construct such a model.

4.2 Extending traditional logical reasoning: From semantic tableaux to design tableaux

In this section, we will extend the semantic tableaux introduced in section 4.1 based on sound design reasoning we introduced earlier. Observe that each tableau rule of the form *L • R ⇒ L' • R'* represents a design move of the form *<L; ∪ R>⇒ <L'; ∪ R'>*, which is sound if *<L0; ∪R0>* is consistent. For our definition of a *design tableau* we use this property to generalize the rules of the semantic tableau.

Definition 6 *A* design tableau *is a tree of* design nodes *L • R, such that:*

- *<L; ∪R> is a design stage*
- *L • R ⇒ L' • R' is a (single) design tableau rule iff, assuming <L'; ∪R'> is consistent, <L; ∪R> ⇒ <L'; ∪R '> is sound.*
- *L0 • R0 ⇒ (L1 • R1; . . . ;Ln • Rn) is a (splitting) rule for design tableau iff, assuming <Li; ∪Ri> are consistent,*
  - *Li |- ∩Lo, i=1..n*
  - *L0, ∩L1→ ∪R1, . . . , ∩Ln→ ∪Rn |- ∪R0*
- *L • R is* closed *if L ∩ R ≠ ∅;*
- *K is a* solution *for L • R if K |- ∩L and K |- ∪R.*

The design tableau rules include the rules of the semantic tableau. So all the rules defined in section 3.1 can be used also in design tableaux.
The first condition of the splitting design tableau rule may seem to be odd: why should only the last one of the bodies-of-knowledge imply the body-of-knowledge of the antecedence *s*0? In fact, it could be any of the stages in the consequence, as can be seen from the proof of theorem 8. One can easily prove that the condition for the application of the single tableau rule is a special case of the split rule (where *n* = 1).
Closing branches in a design tableau differ from semantic tableaux in one important aspect: the closure may be caused by changing information in the body-of-knowledge (the left hand side) or the design concept (the right hand side). This may cause a potential conflict in the case of the application of a splitting rule. The solutions of the design stages in the consequence could become inconsistent when put together to create a solution for the design stage in the antecedence.

Definition 7 *The solutions of the design stages s*1, . . . *sn, say K*1, . . .*Kn, of the consequence of a splitting design rule are called a* consistent resolution *if K = ∪Ki (i ∈* {1, . . . *n*} *is consistent. Such a K will be called a* consistent resolution *of the splitting rule.*

In fact in some cases the inconsistency of solutions in the consequence of a splitting rule can be resolved, i.e. by introducing a disjunction. For example, let *L,A |- C*1 and *L,¬A |- C*1 then these solutions of a splitting rule for *L*0 • *C*0 could be resolved by *L,A∨¬A*. Note that we could, afterwards have introduced an extra sound step in: *L*0 • *C*0 ⇒ *L*0*,A(¬A* • *C*0 and it would have been a simple application of the semantic tableau rule for (R.

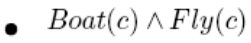
Theorem 8 *A closed design tableau with root K • C contains a solution L such that L |- C. <K; C>⇒ <L; C> will be sound if every split in the tableau has a consistent resolution.*
Proof. Observe that a consequence of the design tableau rules is that for each node *L • R* there will at least be one branch where for all nodes *L' • R'* on that branch we have *L' |- ∩R*. Also, from the

definition of the rules, we can conclude that any solution for a node $L \bullet R$ will imply both $\cap L$ and R. The proof now proceeds by induction to the height of the node in the design tableau.
A closed node $L \bullet R$ (which has heigth 0) has a solution as trivially $L \mid\!\!- \cup R$. As noted above the single step rule is in fact a special case of the splitting rule. We prove the soundness of the splitting rule if it has a consistent resolution. Without loss of generality we may assume $n = 2$. So $L0 \bullet R0$ is split into $L1 \bullet R1$ and $L2 \bullet R2$. As the tableaux below $L1 \bullet R1$ and $L2 \bullet R2$ are closed, by induction hypothesis, there are solutions for these nodes, say $L3 \mid\!\!- \cup R1$ and $L4 \mid\!\!- \cup R2$, such that $L3 \mid\!\!- \cap L1$ and $L4 \mid\!\!- \cap L2$.
From the conditions for the splitting rule it follows that

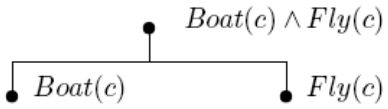
- $L0, \cup R1, \cup R2 \mid\!\!- \cup R0$
- $L1 \mid\!\!- \cap L0$.

Now use $L4 \mid\!\!- \cup L1$ and $L3,L4 \mid\!\!- \cup R1,\cup R2$ to conclude that $L3,L4 \mid\!\!- \cap L0$. Hence $L3 \cup L4$ is a solution for $L0 \bullet R0$ if $L3 \cup L4$ is sound.

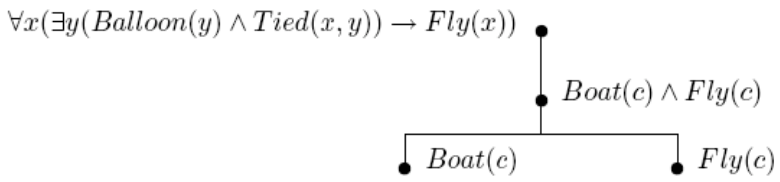
4.3. An illustration of Design tableaux by an example: the concept of a flying boat
To introduce the idea of a design tableau let us give an example. We shall use the classical example 'flying boat' used in presentations of C-K theory [15, 26, 27]. We could call our boat *c* and it will have the properties *Boat*(*c*) (it is a boat) and *Fly*(*c*), it can fly: *Boat*(*c*)∧*Fly*(*c*) Our first design tableau now simply will look like:



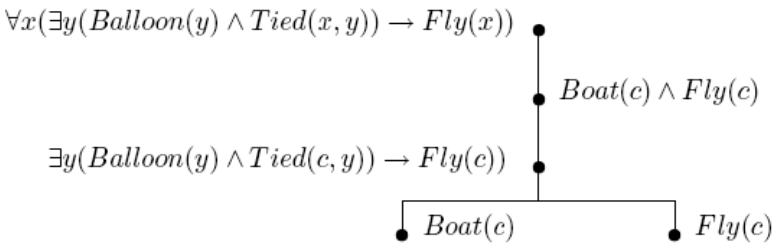
In Beth's semantic tableaux all rules for developing the tableau are based on the simplifying the formulas occuring in the tableau. If a formula of the form *A*∋*B* (*A* and *B*) is false (or unknown), then at least one of them should be false (or unknown). This would result in the following splitting of the tableau:



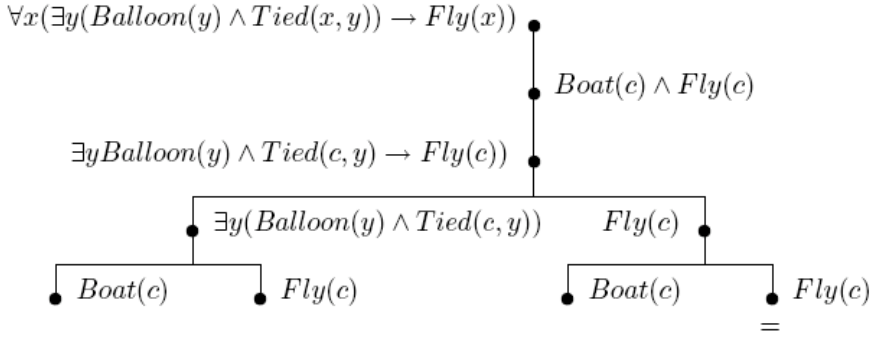
As we have seen, design tableaux differ from semantic tableaux in allowing certain *extra-logical* rules reflecting certain principles typical for design reasoning. One of these principles is that we may add knowledge (hence on the left hand side) at the top of the tableau. *Our search for relevant knowledge is guided by the formulas in the tableau expressing properties of our unknown object c.* In our example: we could add knowledge about *flying*. E.g. we know or learn that to let something fly, we can tie it to a (large) balloon: □*x*(□*y*(*Balloon*(*y*)∧*Tied*(*x*, *y*))→ *Fly*(*x*)):



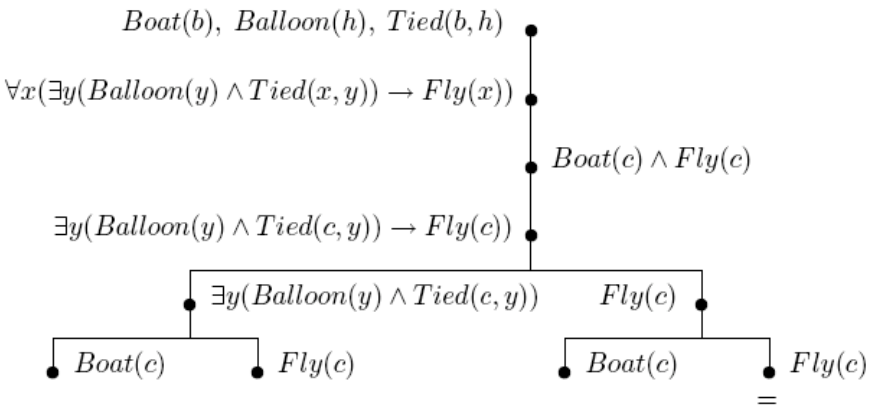
If our flying boat *c* would exist, the rule, how to make something fly, applies also to *c*.



To apply the rules of logic on the new formula, as in Beth's semantic tableaux, we consider two possibilities: the antecedent (□*yBalloon*(*y*)∧*Tied*(*c, y*)) is 'false' (or 'unproved') or the consequence (*Fly*(*c*)) is 'true' (or 'proven'). This causes a new split in our tableau:

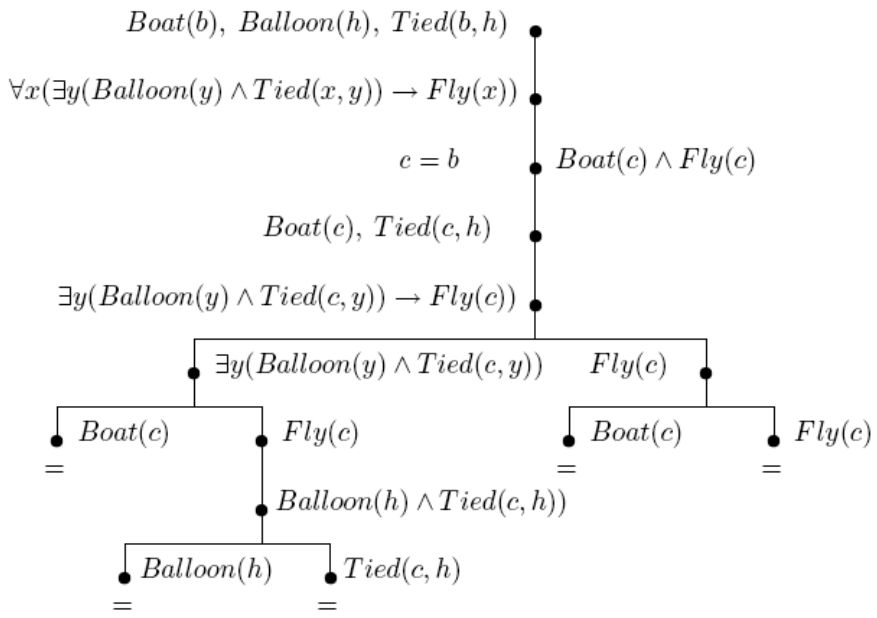


The right most branch of this tableau is *closed* because on this branch there is a formula (*Fly*(*c*)) appearing on *both sides* of the branch. If we think of the tableau method as a systematic search of a counter-model, the search along this branch failed, as a formula can not be true (known) and false (unknown) at the same time. As we will see a closed branch can be thought of as a part of a proof. To keep the example simple, assume the information in the tableau guided us to a balloon (let's call it *h*), which we could tie to some boat (called *b*). This is another K-expansion in terms of C-K.



The tableau thus far suggests that we could use our boat *b* (tied to balloon *h*) as the object *c* of our design (as this would close e.g. the left most branch).

In the final tableaux below, we have used the □-R rule to derive *Balloon*(*h*)∧*Tied*(*c, h*)) and the ^-R rule to split the subtableaux underneath it. The □-R rule says in fact that if a formula of the form □*A*(*x*) is 'false' (i.e. on the right hand side of a branch in the tableau) then for each constant *a* in the domain (here: *b, c, h*) the formula *A*(*a*) should be false (and hence can be placed on the right hand side of the branch too, underneath the node with □*A*(*x*)).



We can now read the proof by starting at the left hand side formula closest to the top node. From there we read the formulas going downwards the left most branch, turning up at the right hand side, until we meet a split, where we turn down reading the formulas on the left. This way we continue to read the formulas along the tableau tree counter clockwise. Omitting repetitions (and identifying *b* and *c*) the proof reads:

1 *Boat*(*b*) (Fact)
2 *Balloon*(*h*) (Fact)
3 *Tied*(*b, h*) (Fact)
4 □*x*(□*y*(*Balloon*(*y*)∧*Tied*(*x, y*))→ *Fly*(*x*)) (Fact)
5 □*y*(*Balloon*(*y*)∧*Tied*(*b, y*))→ *Fly*(*b*)) (4, b, □Elimination)
6 *Balloon*(*h*)∧*Tied*(*b, h*)) (2, 3, ∧Introduction)
7 □*y*(*Balloon*(*y*)∧*Tied*(*c, y*)) (6, □Introduction)
8 *Fly*(*b*) (5, 7,→ Elimination)
9 *Boat*(*b*)∧*Fly*(*b*) (1, □, ∧Introduction)

In this proof we can see that the assumptions of our design proof are facts (knowledge). From the proof we can read that we can take a boat, a balloon and tie the balloon to the boat. Now we have something that can fly (lines 4 to 8) and hence, line 9, we have a flying boat.

5 FURTHER REMARKS ON DESIGN TABLEAUX

Many important issues about Design tableaux remain outside the scope of this paper due to place restrictions. However, looking at our necessarily simple example we can already start to see some important design tableau issues.

- Design tableaux will become very large and complex if used on realistic examples. Here we will use them most of all as a theoretical device. For design support one would need a computer program for the amount of 'administration' that it will take to use design trees. Such computer support is feasible.
- Design tableaux use, for the logical rules, the same rules as semantic tableaux. This facilitates the reconstruction of an actual proof from a closed design tableau.
- A branch in a design tableau closes if a formula appears both on the left and the right of that branch.
- A design tableau closes if all the branches are closed. In this case we have accomplished a proof of the feasibility of our design concept.
- Design tableaux allow the introduction of new formulas in the process that, unlike in the case of semantic tableaux, are not necessarily subformulas of formulas already appearing in the tableaux. As in our example, rules can be set to guide the search for useful knowledge

(e.g. in other databases or by user intervention) by formulas at open-end nodes of the tableaux tree.

- Unlike semantic tableaux, design tableaux allow for expanding the tableaux at the top or somewhere in the middle. This way new knowledge can be used in all open branches without unnecessary repetition. Design tableaux start with a 'named' design concept, introducing a name for a new possibly existing object (in our example called *c*).
- Closing a design tableau will involve identifying known objects with newly introduced names. Starting with a formula of the form $\neg\forall x C(c)$ would, by the logical rules, introduce a new $C(a)$ at the right for each new domain member a.

REFERENCES

[1] Marples, D.L., The Decisions of Engineering Design. *Journal of the Institute of Engineering Designers*, 1960(December), pp1-16.

[2] Zeng, Y. and Cheng, G., On the logic of design. *Design Studies*, 1991, 12(3), pp137-141.

[3] Maher, M.L. and Gero, J.S., Theoretical requirments for creative design by analogy. In *First International Workshop on Formal Methods in Engineering Design, Manufacturing and Assembly,*. Colorado State University, Fort Collins. pp19-27

[4] Yoshikawa, H., ed. *General design theory and a CAD system*. (North Holland Publishing, 1981).

[5] Takeda, H., Veerkamp, P., Tomiyama, T. and Yoshikawa, H., Modeling design processes. *AI Magazine*, 1990, 11(4), pp37-48.

[6] Maimon, O. and Braha, D., A Mathematical Theory of Design. *International Journal of General Systems*, 1996, 27(4-5), pp275-318.

[7] Braha, D. and Reich, Y., Topological structures for modelling engineering design processes. *Research in Engineering Design*, 2003, 14, pp185-199.

[8] Shai, O. and Reich, Y., Infused design: I theory. *Research in Engineering Design*, 2004, 15(2), pp93-107.

[9] Shai, O. and Reich, Y., Infused design: II practice. *Research in Engineering Design*, 2004, 15(2), pp108-121.

[10] Suh, N.P., *The principles of design*. (Oxford University Press, 1990).

[11] Simon, H.A., *The Sciences of the artificial*. (MIT Press, Cambridge, Massachusetts, 1969).

[12] Simon, H.A., Problem forming, problem finding and problem solving in design. In Collen, A. and Gasparski, W., eds. *Design and systems: General applications of methodology*, pp245-259 (Transactions Publishers, New Jersey, 1995).

[13] Pahl, G. and Beitz, W., *Engineering Design: a systematic approach*. (The Design Council, London, 1984).

[14] Gero, J.S., Design prototypes: a knowledge representation schema for design. *AI Magazine*, 1990, 11, pp26-36.

[15] Hatchuel, A. and Weil, B., Pour une théorie unifiée de la conception, Axiomatiques et processus collectifs. In *CGS Ecole des Mines, GIS cognition-CNRS.* pp1-27

[16] Hatchuel, A. and Weil, B., A new approach of innovative design : an introduction to C-K design theory. In *ICED'03.* Stockholm, Sweden. pp14

[17] Hatchuel, A. and Weil, B., C-K design theory: an advanced formulation. *Research in Engineering Design*, 2009, 19(4), pp181-192.

[18] Hatchuel, A. and Weil, B., Design as Forcing:Deepening the foundations of Ck theory. In *16th International Conference on Engineering Design - ICED 2007, Knowledge, Innovation and Sustainability.* Paris, France.

[19] Hatchuel, A., Le Masson, P. and Weil, B., C-K theory in practice: Lessons from industrial applications. In *8th International Design Conference, Design 2004.* Dubrovnik, Croatia.

[20] Le Masson, P., Weil, B. and Hatchuel, A., *Strategic Management of Design and Innovation*. (Cambridge University Press, Cambridge, 2010).

[21] Le Masson, P., Hatchuel, A. and Weil, B., Creativity and design reasoning: how C-K theory can enhance creative design. In *International Conference on Engineering Design ICED'07.* Paris, France.

[22] Hatchuel, A., Le Masson, P. and Weil, B., Design Theory and Collective Creativity: a Theoretical Framework to Evaluate KCP Process. In *International Conference on Engineering Design, ICED'09.* 24-27 August 2009, Stanford CA.

[23] Hatchuel, A., Le Masson, P., Segrestin, B. and Weil, B., The Design Theory Program: a Paradigm for a new Theory of Management. *Convergence: Managing + Designing*Cleveland, Ohio, 18-19 June 2010, 2010).
[24] Hatchuel, A. and Masson, P.L., Innovation répétée et croissance de la firme. In *Cahiers des Recherches, Ecole des Mines de Paris - Centre de Gestion Scientifique.* pp61p.
[25] Kazakci, A. and Hatchuel, A., From pure constructivism to imaginative constructivism: An analysis of Intuitionist Mathematics from the viewpoint of design theory. *submitted*, 2011.
[26] Hatchuel, A. and Weil, B., La théorie C-K: Fondaments et usages d'une théorie unifiée de la conception. In *Colloque Sciences de la Conception.* Lyon.
[27] Kazakci, A., A formalisation of CK design theory based on Intuitionist Logic. In Chakrabarti, A., ed. *International Conference on Research into Design. ICORD09*, pp499-507 (Research Publising Services, Banglore, India, 2009).
[28] Hendriks, L. and Kazakci, A., A formal account of the dual expansion of concepts and knowledge in C-K theory. In *International Deisgn Conference - Design 2010.* Dubrovnik - Croatia, May 17 - 20, 2010.
[29] Beth, E.W., Semantic entailment and formal derivability. *Med. Kon. Ned. Akad. Wet, Amsterdam*, 1955, 18(13).
[30] Hendriks, L., Imagining Future Knowledge, Logic and Interactive Rationality Seminar, The Institute for Logic, Language and Computation, University of Amsterdam, . Amsterdam (http://www.illc.uva.nl/lgc/seminar/?p=738), 2010).
[31] Kazakci, A., Hatchuel, A. and Weil, B., A model of C-K design theory based on a term logic: a formal background for a class of design assistants. In *International Design Conference 2008.* Dubrovnik, Croatia.
[32] Kazakci, A. and Hatchuel, A., Is "creative subject" of Brouwer a designer? - an Analysis of Intuitionistic Mathematics from the Viewpoint of C-K Design Theory. In, *International Conference on Engineering Design, ICED'09,* . Stanford CA, 24-27 August 2009. .