ELSEVIER

Theoretical Computer Science 286 (2002) 323–366

Theoretical Computer Science

www.elsevier.com/locate/tcs

Rasiowa–Sikorski deduction systems in computer science applications

Beata Konikowska[1]

Institute of Computer Science, Polish Academy of Sciences, Ordona 21, 01–237 Warsaw, Poland

Abstract

A Rasiowa-Sikorski system is a sequence-type formalization of logics. The system uses invertible decomposition rules which decompose a formula into sequences of simpler formulae whose validity is equivalent to validity of the original formula. There may also be expansion rules which close indecomposable sequences under certain properties of relations appearing in the formulae, like symmetry or transitivity. Proofs are finite decomposition trees with leaves having "fundamental", valid labels. The author describes a general method of applying the R-S formalism to develop complete deduction systems for various brands of C.S and A.I. logic, including a logic for reasoning about relative similarity, a three-valued software specification logic with McCarthy's connectives and Kleene quantifiers, a logic for nondeterministic specifications, many-sorted FOL with possibly empty carriers of some sorts, and a three-valued logic for reasoning about concurrency.

Keywords: Rasiowa-Sikorski; Deduction system; Decomposition rule; Decomposition tree; Completeness; Many-valued logic; Software specification logic; Concurrency

1. Introduction

The aim of this paper is to advertise a powerful and flexible, yet simple methodology of developing deduction systems for various logics based on the analysis of their semantics.The paper will present a general outline of this methodology, and show examples of its applications to various brands of computer science logics.

The said methodology is inherently connected with the semantics of the given logic, and its key concept is to obtain an adequate and complete proof mechanism for

[1] This research has been partially supported by the LoSSeD workpackage of the CRIT-2 project funded by European ESPRIT and INCO programmes.

E-mail address: beatak@ipipan.waw.pl (B. Konikowska).

0304-3975/02/$ - see front matter
PII: S0304-3975(01)00320-6

the logic in a systematic way from the said semantics. This is achieved by "mirroring" the semantics of all the logical constructs (connectives, quantifiers, modalities, etc.) through invertible rules operating on sequences of formulae of the logic. The methodology is based on the use of a simple and universal deduction mechanism developed by Polish logicians about 50 years ago: the Rasiowa–Sikorski (R–S) deduction system [24]—and hence from now on it will be called R–S methodology.

An R–S deduction system is a sequence-type formalisation of logics in a natural deduction style. To prove validity of a complex formula, we decompose it step by step into a sequence or sequences of simpler ones, the validity of which is equivalent to that of the original formula. The resulting decomposition tree has the structure of a finitely branching tree with vertices labelled by sequences of formulae; the tree is a proof if it is finite and all its leaves are labelled with "axioms" of the logic, called fundamental sequences. The R–S system is described in more detail in the next section.

From the semantic viewpoint, an R–S system is dual to the well-known tableau system of Beth [1]; in fact, the concept of developing a proof system through analysing the semantic is common to both systems. What is more, it can be argued that R–S systems and tableaux are, in fact, equivalent from the proof-theoretical viewpoint—though the difference in presentation might obscure this fact at first.

In spite of this, there is a large disparity in popularity of both these deduction mechanisms: the tableau systems have gained much popularity due to their widespread use in proof automation, whereas R–S systems have remained relatively unknown. However, over the years, the R–S methodology based on the use of such systems has—in the author's experience—proved to be a very convenient and powerful tool for developing proof mechanisms for multifarious formal systems connected with software specification and verification, logics of information systems, as well as other issues related to CS logics.

The R–S methodology has four essential advantages:

- A clear-cut method of generating proof rules from the semantics: For all the formula constructors, we just give the condition for the resulting formula to be satisfied, and the condition for it to be not satisfied.
- The resulting deduction system well suited for automated deduction purposes.
- A standard and intuitively simple way of proving completeness by constructing a counter-model for a nonprovable formula out of its "wrong" decomposition tree in an R–S system.
- An almost automatic way of transforming a complete R–S system into a complete Gentzen calculus system.

In the paper we describe the R–S methodology and show how it can be applied to formalize various types of logics encountered in computer science. Concrete examples of such applications coming from the author's past work in the area of many-valued logics, nondeterminism, similarity, indiscerniblity and complementarity, as well as many-sorted

first-order logic with empty carriers of some sorts, are given. Finally, a new result concerning formalisation of a temporal logic with a three-valued semantics for reasoning about concurrency is presented.

The framework of the paper is based on [15], with new results concerning the above-mentioned temporal logic and complementarity logics added. There is also a new chapter on the transformation of an R–S system to a Gentzen calculus of sequents subsuming earlier work and completing the presentation of the R–S methodology.

2. Rasiowa–Sikorski systems, their inherent mechanisms and auxiliary tools

In the section we shall describe the basic concept underlying the R–S methodology—the R–S deduction system mentioned in the introduction. Besides giving a formal definition of such a system, we shall also try to point out the mechanisms used within R–S systems to handle some basic logical constructs, together with some auxiliary technical tools helpful in developing such systems.

2.1. Fundamentals of Rasiowa-Sikorski deduction systems

A Rasiowa–Sikorski (R–S) deduction system belongs to the broad class of the so-called "natural deduction" systems. It consists of

- decomposition rules for sequences of formulae ("inference rules"),
- fundamental sequences ("axioms").

The system is used for constructing a decomposition tree of a formula (or finite sequence of formulae) by repeated applications of decomposition rules. The rules break a complex formula into a sequence or sequences of simpler ones, the validity of which is equivalent to that of the original formula. If the resulting tree is finite and all its leaves are labelled by fundamental sequences ("guaranteed" to be valid), then the formula (sequence) is said to be provable.

Let us now define the basic notions of the R–S system. We assume the context of an arbitrary formal language with well-defined syntax and semantics, in which the notion of a model incorporates a valuation. Satisfaction of a formula φ in a model M is denoted by $M \models \varphi$, and the set of all formulae—by $\mathscr{F}$.

Definition 1. A sequence $\Omega = \varphi_1, \varphi_2, \ldots, \varphi_n$ of formulae is *satisfied in a model M* ($M \models \Omega$) iff $M \models \varphi_i$ for some $i, 1 \leqslant i \leqslant n$, and *valid* ($\models \Omega$) iff $M \models \Omega$ for each M.

Thus, we see that the comma in sequences of formulae is equivalent to meta-disjunction on the level of satisfaction in a model (incorporating a valuation).

Fundamental sequences form a subclass of valid sequences, i.e. any fundamental sequence is satisfied in every model, and are defined separately for each concrete logic. Roughly speaking, they are chosen as sequences "guaranteed to be valid" by force of some basic law of the semantics of the logics, which gives rise to a simple criterion

for recognising such sequences. In classical first-order logic, a fundamental sequence is any sequence which contains both a formula and its negation. The underlying semantic law is, of course, the fact that either a formula or its negation have to be satisfied in a model incorporating a valuation.

A *decomposition rule* is an expression of the form

$$\frac{\Omega}{\Omega_1 \mid \Omega_2 \mid \ldots \mid \Omega_n}$$

where all the Ω's are sequences of formulae. Ω is called the *conclusion* of the rule, and $\Omega_1, \Omega_2, \ldots, \Omega_n$—its *premises*. A rule is *sound* provided its conclusion is valid iff all its premises are valid.

Note that the notion of rule soundness used traditionally in case of R–S systems is stronger than usual. Indeed, *sound decomposition rules* are equivalent, or *invertible* rules *leading from valid sequences to valid sequences in both directions—both* "*downwards*" *and* "*upwards*". Intuitively, a more complex formula or sequence of formulae Ω is replaced by some simpler sequences $\Omega_1, \ldots, \Omega_n$ whose simultaneous validity is equivalent to validity of Ω. This invertibility of rules is symbolised by a double horizontal line in notation of the rules. What is more, the branching $\mid$ in rules corresponds to meta-conjunction on the validity level.

An *indecomposable formula* is, intuitively, a formula not subject to further decomposition. Its formal definition is tailored to a concrete logic; in classical first-order logic it is just any literal. A sequence of formulae is said to be *indecomposable* if it only contains indecomposable formulae.

Any decomposition rule can be presented in the following more detailed form:

$$\frac{\Omega', \Sigma, \Omega''}{\Omega', \Sigma_1, \Omega'' \mid \Omega', \Sigma_2, \Omega'' \mid \ldots \mid \Omega', \Sigma_n, \Omega''}$$

where Ω' is an indecomposable sequence of formulae. As we see, Ω', Ω'' serve only as the context, and the actually decomposed subsequence (or just a formula) is Σ.

Hence, any decomposition rule is actually applied to the leftmost decomposable formula or subsequence of formulae, which determines the decomposition order for a given sequence of formulae. This is part of a "control mechanism" embedded in an R–S system aimed at facilitating automated deduction by providing an implicit deterministic algorithm for generating a decomposition tree of a formula (see also Sections 2.2 and 2.5). Moreover, all indecomposable formulae are inherited by the next node in the decomposition tree, and so further down the branch.

The fundamental strategy of developing decomposition rules is, roughly speaking, as follows: for each complex (i.e. decomposable) formula α, and each logical value k, we must have a decomposition rule expressing validity of the formula equivalent to the statement "the value of α is k" in terms of simultaneous validity of some

sequence(s) of formulae representing statements of the type "the value of α_i is k_i", where α_i are some simpler formulae (usually subformulae of α), and k_i are some logical values. Of course, such strategy can only be implemented if the semantic conditions quoted above are expressible through formulae of a given logic, which amounts to expressibility of all the clauses constituting the definition of semantics through formulae of the logic. Hence an R–S system can only be developed for the logics, the languages of which are expressive enough (for more discussion, see the section on R–S methodology).

As each complex formula α is of the form $C(\alpha_1,\ldots,\alpha_n)$, where C is some constructor of the logic, the simplest way of implementing the strategy described above is to provide, for each such constructor, a rule expressing the validity of the formula representing the statement "the value of $C(\alpha_1,\ldots,\alpha_n)$ is k" in terms of simultaneous validity of some sequences of the formulae representing the statements "the value of α_i is k_i".

For example, in classical logic the statement "the value of $\alpha\vee\beta$ is *true*" is expressed by the formula $\alpha\vee\beta$. The latter formula is valid iff, for any model M, either α or β are satisfied in M, i.e. if the sequence α,β is valid. This, after adding the contexts, gives rise to the rule

$$\frac{\Omega',\alpha\vee\beta,\Omega''}{\Omega',\alpha,\beta,\Omega''}.$$

On the other hand, the statement "the value of $\alpha\vee\beta$ is *false*" is valid iff the formula $\neg(\alpha\vee\beta)$ is valid. The latter holds if both $\neg\alpha$ and $\neg\beta$ are satisfied in any model, i.e. if both these formulae are valid. As a result, we get the decomposition rule

$$\frac{\Omega',\neg(\alpha\vee\beta),\Omega''}{\Omega',\neg\alpha,\Omega''\mid\Omega',\neg\beta,\Omega''}.$$

It should be noted that the above method cannot be applied directly to some unary constructors, because, e.g. in classical logic validity of $\neg\alpha$ cannot be expressed through validity of anything simpler. Yet if we go one step lower and assume that α is a composed formula, e.g. of the form $\alpha_1\vee\alpha_2$, then we see that $\neg\alpha$ can be handled using the rule for $\neg(\alpha_1\vee\alpha_2)$ (falsity of disjunction) developed above. Of course, the special case of $\alpha=\neg\beta$ needs a separate "double negation" rule

$$\frac{\Omega',\neg\neg\alpha,\Omega''}{\Omega',\alpha,\Omega''}.$$

Thus, instead of trying to define a rule for negation alone we just use a set of rules for all the compositions of the form $\neg C$, where C is any constructor of the logic (including negation itself).

As a result, in case of first-order classical logic we have a single rule for the (double) negation itself, plus two rules for each other logical connective and each quantifier: one for the resulting formula to be true, and one for it to be false. The full set of decomposition rules for first-order classical logic (quoted from [21]) is given below. *Decomposition rules for classical logic*:

$$\frac{\Omega', \neg\neg\alpha, \Omega''}{\Omega', \alpha, \Omega''} \qquad \frac{\Omega', \alpha \vee \beta, \Omega''}{\Omega', \alpha, \beta, \Omega''} \qquad \frac{\Omega', \neg(\alpha \vee \beta), \Omega''}{\Omega', \neg\alpha, \Omega'' \mid \Omega', \neg\beta, \Omega''}$$

$$\frac{\Omega', \alpha \wedge \beta, \Omega''}{\Omega', \alpha, \Omega'' \mid \Omega', \beta, \Omega''} \qquad \frac{\Omega', \neg(\alpha \wedge \beta), \Omega''}{\Omega', \neg\alpha, \neg\beta, \Omega''} \qquad \frac{\Omega', \forall x.\alpha, \Omega''}{\Omega', \alpha(z), \Omega''}$$

$$\frac{\Omega', \neg\forall x.\alpha, \Omega''}{\Omega', \neg\alpha(t/x), \Omega'', \neg\forall x.\alpha} \qquad \frac{\Omega', \exists x.\alpha, \Omega''}{\Omega', \alpha(t/x), \Omega'', \exists x.\alpha} \qquad \frac{\Omega', \neg\exists x.\alpha, \Omega''}{\Omega', \neg\alpha(z), \Omega''}$$

Here z is a variable which does not appear above the double line, and t is any term free for x in α.

As we see, basic decomposition rules of an R–S system break down any complex formula into its component parts—a sequence or sequences of simpler formulae that determine validity of the original formula. This is a fundamental concept of an R–S system that facilitates a "standardized" completeness proof by counter-model construction.

It should be noted that in addition to the decomposition rules of the type described above, called *replacement rules*, in R–S systems we encounter also a different type of rules, "closing" a sequence under some relational axioms like symmetry or transitivity, and called *expansion rules*. For more information on both types of rules, the reader is referred to Section 2.3.

2.2. Handling quantification and modalities in R–S systems

To see why the R–S systems—and hence R–S methodology—are so widely applicable, in the next two sections we discuss the way in which they handle the problem of expressing some basic constructs, like quantifiers or relational axioms,[2] in the form of either decomposition rules or fundamental sequences.

Let us start with the mechanism of dealing with quantification in R–S systems. From the rules for classical quantifiers given in the preceding section:

$$\frac{\Omega', \forall x.\alpha, \Omega''}{\Omega', \alpha(z), \Omega''} \qquad \frac{\Omega', \exists x.\alpha, \Omega''}{\Omega', \alpha(t/x), \Omega'', \exists x.\alpha}$$

[2] Here by relational axioms we mean axioms representing basic properties of relations, like reflexivity, transitivity, etc., which are specially important in case of modal logic. The best-known example of such axioms are the equality axioms.

where z is a new variable, and t is any term free for x in α, we can see that universal quantification is replaced by substitution of a new free variable in the quantified formula. Intuitively, this new variable represents an arbitrary substitution of some value for the original bound variable. In case of existential quantification, we use subsequent substitutions of arbitrary terms for the bound variable in the quantified formula, while copying the original formula. Clearly, this allows us to keep looking for a "witness" to the existential quantification (i.e. a value which substituted for the quantified variable x satisfies α) while preserving an equivalent character (invertibility) of the rule.

Note that shifting the existential quantifier to the end of the decomposed sequence assures fairness of decomposition: in this way we avoid decomposing the existential formula again and again while the ones to the right of it would wait, possibly endlessly, for their turn. This is another element of the control mechanism embedded in R–S systems to facilitate automated deduction. Evidently, this part of the said mechanism is a basic safeguard against trivial (avoidable) infinite loops.

Of course, possibility and necessity modalities—which are just bounded existential and universal quantification, respectively—can be handled in an analogous way, as we shall see in the section on similarity and complementarity logics, as well as in the section on temporal logic.

What is more, the method can be extended to other relationships involving implicit quantification—like inclusion of sets—which in turn will be seen in the section on the logic for nondeterministic specifications.

2.3. Replacement and expansion rules—handling of relational axioms

Now, let us pass to the problem of handling relation-related axioms, called shortly relational axioms. The simplest axioms involving one element only, like reflexivity or irreflexivity, are usually expressed through fundamental sequences. Thus, e.g. the fact that the equality relation is reflexive is expressed by stating that every sequence of formulae containing a formula "$\mathrm{t} = \mathrm{t}$" is fundamental.

However, properties of relations involving several objects, like symmetry, transitivity, 3-transitivity, antisymmetry, etc., are best expressed using a special type of decomposition rules.

Namely, decomposition rules of any R–S system divided into two basic types. The first—discussed up to now—are *replacement rules* "breaking down" each decomposable formula into a sequence or sequences of simpler formulae whose validity determines validity of the original formula. The R–S system for classical logic (without equality) given in the preceding section consisted of replacement rules only, and as a typical example we can quote, e.g. the disjunction rule:

$$\frac{\Omega', \alpha \vee \beta, \Omega''}{\Omega', \alpha, \beta, \Omega''}.$$

The second are *expansion rules* which add some formula(e) to the sequence to close it, e.g. under a symmetry or transitivity property of some relation—or any other "relational

axiom" of the logic. An exemplary expansion rule is

$$\frac{\Omega', t = u, \Omega''}{\Omega', t = u, u = t, \Omega''}$$

closing a sequence of formulae under symmetry of equality.

If we read the rule in the upward direction, these rules can be called *omitting rules* since one of the formulae in the premise is omitted due to the considered relational property. Sometimes the rule is expressed—for purely technical reasons—in terms of negated formulae, but the principle is always the same: an expansion rule adds to the relationships we already have those that follow from them due to certain properties of the relation involved.

When defining decomposition rules or fundamental sequences, we should also keep in mind a simple fact that validity of the sequence $\neg\alpha_1, \ldots, \neg\alpha_n, \beta$ is equivalent to validity of the (meta-)implication $\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta$. This can be very helpful in defining especially some expansion rules modelling entailments of the kind discussed above.

Typical expansion rules expressing properties of relations will be given in the subsequent sections on modal logics.

An important feature of the R–S methodology is that the two types of rules mentioned above have the following basic properties:

- Replacement rules are applicable to decomposable sequences only.
- Any R–S system is constructed in such a way that exactly one (instance of) a replacement rule is applicable to any given decomposable sequence.
- Expansion rules can also be applied to indecomposable sequences.

2.4. Use of signed formulae

In many cases, the language we are interested in does not exhibit the dichotomous property characteristic for classical logic, where, given a model incorporating a valuation, either a formula or its negation must be satisfied in this model. This is the case, e.g. when the logical calculus is many valued, the negation is a nonstandard one, or there is no negation at all. Such a situation is usually a considerable drawback in developing the R–S deduction system, because we might lack means for expressing our basic semantic conditions of the form "the value of formula α is k" for k different from *true*, i.e. the maximal logical value in a given logic.

To overcome it, we can employ the mechanism of the so-called signed formulae used already by Beth in his tableau systems.

A signed formula is obtained by preceding an ordinary formula by either the "truth" (satisfaction) operator **T**, or the "nontruth" (nonsatisfaction) operator **N**. Thus, using the **N** operator we can express the fact that a formula is not satisfied, which amounts to introducing an artificial two-valued negation on top of the existing structure of the language. The use of **T** and **N** clarifies many technical issues by providing a clear-cut dichotomy "either $\mathbf{T}(\alpha)$ or $\mathbf{N}(\alpha)$".

Definition 2. A signed formula over $\mathscr{F}$ is any element of the set

$$\mathscr{SF} = \{\mathbf{T}(\alpha)\colon\ \alpha \in \mathscr{F}\} \cup \{\mathbf{N}(\alpha)\colon\ \alpha \in \mathscr{F}\}.$$

The notion of satisfaction of a signed formula in a model M is given by $M \models \mathbf{T}(\alpha)$ *iff* $M \models \alpha$, $M \models \mathbf{N}(\alpha)$ *iff* $M \not\models \alpha$.

Some examples of rules involving signed formulae will be given in the subsequent sections.

We should note that the extension of the language with signed formulae need not be permanent. Namely, the **T**-**N** operators are not nested, and we can drop them by passing from a complete R–S system for signed formulae to an equivalent Gentzen calculus for "ordinary" formulae, as will be shown in Section 9.

Finally, in case of a general many-valued logic—like e.g. Rosser–Turquette n-valued logic—instead of "binary" signed formulae we can use formulae labelled with the individual logical values. An example of such an approach is given e.g. in [14] or [17]. In that paper, the role of labels is actually played by the J_k operators "selecting" individual logical values k—but the principle is the same.

2.5. Decomposition trees and proofs

The general idea of the R–S deduction system is to decompose the original formula with the help of replacement rules into quite elementary formulae that cannot be decomposed any further—they are just the indecomposable formulae mentioned in Section 2.1. Then, in some logics, sequences of indecomposable formulae are "closed" by means of expansion rules.

Proofs in R–S systems consist in constructing decomposition trees for formulae—or more generally, for sequences of formulae—using decomposition rules in the way described below.

Definition 3. A decomposition tree for a formula φ (a sequence Ω) is a maximal tree $DT(\varphi)(DT(\Omega))$ with vertices labelled by sequences of formulae defined inductively as follows:

(i) The root of the tree is labelled by $\varphi(\Omega)$.

(ii) Let v labelled by Σ be the leaf of a branch B of the tree constructed up to now. Then:

(a) *we terminate the branch B at vertex v* if either:

(a1) Σ is a fundamental sequence, or

(a2) Σ is indecomposable and no expansion rule is applicable to Σ;

(b) *otherwise we expand the branch B beyond v by attaching to that vertex n sons labelled by* $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ if

$$\frac{\Sigma}{\Sigma_1 | \Sigma_2 | \ldots | \Sigma_n}$$

is a decomposition rule applicable to Σ.

Thus in order to obtain a decomposition tree of φ (or Ω), we label the root by φ (resp. Ω), and then expand the tree by applying the decomposition rules in R–S. If rule R applied to the label of a vertex has n premises, then the vertex has n sons, labelled by the n premises of the rule, respectively. If in the course of expanding the branch we get a fundamental sequence, we terminate the branch—successfully. The branch is also terminated, but unsuccessfully, if we get an indecomposable sequence to which no expansion rule (and hence no rule at all) can be applied. Hence, every leaf of the decomposition tree is labelled by either a fundamental or indecomposable sequence.

Definition 4. A decomposition tree is called a proof if it is finite and all its leaves are labelled by fundamental sequences. A formula φ (sequence Ω) is said to be provable ($\vdash\varphi$, $\vdash\Omega$) iff its decomposition tree $DT(\varphi)$ ($DT(\Omega)$) is a proof.

Recall that R–S systems are constructed in such a way that exactly one replacement rule is applicable to each decomposable sequence of formulae. Hence, refining the definition of the decomposition tree (see [11]) by adding rules for a unique choice of new variables/terms introduced by the rules, as well as for a unique choice of the expansion rule to be applied, we get, in fact, an algorithm for generating a unique decomposition tree $DT(\Omega)$ ($DT(\varphi)$) for every sequence Ω (formula φ). Since the refinement is done in such a way that for a provable Ω that unique $DT(\Omega)$ is a proof of Ω, then the said algorithm can be considered as a proof search algorithm. Of course, what we call an "algorithm" can only be a real algorithm in case of decidable logic. In case of an undecidable logic, the only thing we can hope for is a partial algorithm, which will produce proofs of valid formulae (in practice, those with "short" proofs), refutations of some invalid formulae (those with finite decomposition trees), and will run forever on invalid formulae with infinite decomposition trees—unless we impose a fixed limit on the number of steps, after which the algorithm will halt returning a "failure" result.

From now on, we assume the decomposition tree is unique, and denote it as above.

Example. As an example, let us consider the proof of an obviously valid FOL formula

$$\alpha(a) \to (\beta(a) \to \exists x.(\alpha(x) \wedge \beta(x)))$$

in the R–S system for classical logic presented in Section 2.1. Since for the sake of brevity we did not incorporate implication in that system, treating it as a derived conjunctive, than we first replace the above formula by its equivalent using the equivalence $\varphi \to \psi \equiv \neg\varphi \vee \psi$, which yields

$$\neg\alpha(a) \vee (\neg\beta(a) \vee \exists x.(\alpha(x) \wedge \beta(x))).$$

The proof of the latter formula is as follows:

$$\neg\alpha(a) \vee (\neg\beta(a) \vee \exists x.(\alpha(x) \wedge \beta(x)))$$

$$| \ (\vee)$$

$$\neg\alpha(a), \neg\beta(a) \vee \exists x.(\alpha(x) \wedge \beta(x))$$

$$| \ (\vee)$$

$$\neg\alpha(a), \neg\beta(a), \exists x.(\alpha(x) \wedge \beta(x))$$

$$| \ (\exists, a)$$

$$\neg\alpha(a), \neg\beta(a), \alpha(a) \wedge \beta(a), \exists x.(\alpha(x) \wedge \beta(x))$$

$$| \ (\wedge) \qquad\qquad | \ (\wedge)$$

$$\underline{\neg\alpha(a), \neg\beta(a), \alpha(a), \exists x.(\alpha(x) \wedge \beta(x))} \quad \underline{\neg\alpha(a), \neg\beta(a), \beta(a), \exists x.(\alpha(x) \wedge \beta(x))}$$

Here the symbols labelling the vertical bars denote (the instances of) the rules used, and the underlined sequences are fundamental sequences labelling the leaves of the proof tree (recall that any sequences containing both a formula and its negation is fundamental).

As we have mentioned in the introduction, the R–S system is dual to the tableau system from the semantic viewpoint. First, in a tableau system we prove validity of a formula φ by showing that $\neg\varphi$ has a closed tableau (refutation method), whereas in an R–S system we prove directly that φ has a closed, "correct" decomposition tree. Second, splitting of branches corresponds to disjunction in a tableau, and to conjunction in an R–S decomposition tree.

There are, however, two differences connected with the issue of automated deduction. First, the R–S system has an embedded "control mechanism" we have discussed above, generating after some refinement a kind of proof search algorithm. Second, in an R–S system all information needed for further decomposition is contained in the vertex, so—in contrast to the tableaux case—no "look back" up the branch is needed, and when constructing the decomposition tree we need only to store the last vertex on each branch.

Of course, from the viewpoint of practical applications in automated deductions these differences might not be very significant. The "control mechanism" will need to be replaced by a more efficient algorithm—probably analogous (or dual) to those used in case of tableaux, and storage of decomposition tree or a tableaux will probably be effected through use of pointers to formulae anyway. However, one thing seems to be clear: R–S systems are as well suited to automated deduction as tableaux.

In fact, the most important difference between these two types of systems—in the author's opinion, constituting a pronounced advantage of the R–S systems over tableaux—is the invertibility of rules in R–S systems: the property that tableaux do not have. Indeed, invertible rules are more desirable from the automatic deduction viewpoint, since they facilitate proof search. What is more, they enforce a greater discipline in

developing the system, and make it easier to discover any possible loopholes leading to incompleteness.

2.6. Soundness and completeness

If the decomposition rules correctly represent semantics of individual logical constructs—which is exactly the basic idea behind the formalism, then it is a straightforward thing to prove that the system is sound, i.e. provability implies validity.

Theorem 1 (Soundness). *For any sequence Ω and any formula φ,*

$$\vdash \Omega \text{ implies } \models \Omega, \qquad \vdash \varphi \text{ implies } \models \varphi.$$

It is much more difficult to prove completeness, but there is a general strategy of the proof:

Theorem 2 (Completeness). *For any sequence Ω of formulae, and any formula φ, if Ω (φ) is valid, then Ω (φ) is provable, i.e.*

$$\models \Omega \text{ implies } \vdash \Omega, \qquad \models \varphi \text{ implies } \vdash \varphi.$$

Proof (*Strategy*). We argue by contradiction, showing that $\nvdash \Omega$ implies $\not\models \Omega$. There are two situations when the decomposition tree $DT(\Omega)$ is not a proof. First, certain leaf may be not labelled by a fundamental sequence (Case 1), or the tree may be infinite. By König's lemma, the latter implies existence of an infinite path (Case 2).

In Case 1, the set Δ of formulae appearing in the label of the leaf must consist of indecomposable formulae only; in Case 2, the set Δ of all indecomposable formulae appearing in the labels on the infinite branch cannot contain a fundamental sequence. In both cases Δ must be closed under all expansion rules due to maximality of the tree.

We construct a counter-model $CM = \langle C, v \rangle$ for Δ with the universe consisting of either expressions of the language, or equivalence classes of such expressions. In Case 1, Ω cannot be true in CM. Indeed, as our decomposition rules are two-way ones, the label of a node of $DT(\Omega)$ is valid iff the labels of all its sons are valid. Since the sequence labelling a leaf of $DT(\Omega)$ is not valid and the tree is finite in Case 1, this proves Ω cannot be valid. In Case 2, we prove by induction on the rank of a formula (defined as the "height" of its syntactic derivation tree) that no formula on the considered infinite branch can be true in CM. As the top node of this branch is the root labelled by Ω itself, then $CM \not\models \Omega$, i.e. Ω is not valid, which ends the proof by contradiction.

3. Outline of the R–S methodology

In this section we present the basic outline of the R–S methodology, whose applications to concrete computer science logics will be given in the subsequent sections.

According to what has been said above, the basic idea of the methodology is to develop a sound and complete R–S deduction system (see the preceding section) for a given logic through analysis of its semantics. In principle, the resulting R–S deduction system is treated as the final result returned by this methodology. However, when required it can be also further transformed to a Gentzen calculus using the translation procedure given in the paper, or—after certain refinement—employed for automatic deduction purposes. Below we shall describe the individual aspects of the methodology, recalling also those that have been already discussed in the preceding sections in connection with presentation of the R–S deduction systems.

Prerequisites of the methodology: As the R–S methodology is based on analysing the semantics of a given logic, the first prerequisite for applicability of this methodology is availability of an appropriate semantics for the logic, which is the starting point for the whole process. However, there is a second prerequisite too: the language of the logic must be expressive enough to adequately capture the semantics. Indeed: due to the strategy of developing the decomposition rules of an R–S system (see *Development of an R–S system for a logic* below), the individual clauses of the definition of the semantics must be expressible through formulae of the logics. In fact, this requirement closely parallels the one for tableaux, since both the proof methods base on mirroring semantics of individual logical constructs by proof rules.

Adapting a logical language for the use of R–S methodology: If the language of the logic is not expressible enough to capture the necessary semantic conditions, then we can try to enrich it by introduction of some additional tools, like e.g. signed formulae, nominals, etc. This makes sense in cases when what we are really interested in is providing a sound and complete logic describing a given class of semantic models, which gives us a certain liberty with respect to the choice of the actual language. Thus, for example, the R–S methodology cannot handle any propositional modal or temporal logic in its "classical" shape, i.e. with language containing propositional variables, logical conjunctives and modal operators only, since in that case validity of, e.g. $\Box\alpha$ cannot be expressed through validity of any simpler formulae. However, once we introduce nominals representing states of the underlying Kripke frame and a constant(s) for the accessibility relation(s), we can easily develop a complete R–S system for the logic (see e.g. Section 8). What is more, use of R–S methodology makes it quite easy to provide complete deduction systems for modal logics with modally inexpressible conditions on the accessibility relations, since in an R–S system such conditions can as a rule be expressed in a simple way like any other relational axioms, i.e. through either fundamental sequences or expansion rules (see Section 2.2).

Applicability: From what has been said above it is rather evident that applicability of the R–S methodology should be practically the same as applicability of the tableau one: they are based on the same general principle, they are dual from the semantic viewpoint, practically equivalent from the proof-theoretic viewpoint, and have analogous prerequisites for applicability. Of course, up to now applications of the tableaux have been much more widespread than those of the R–S systems. Yet the application scope of the latter found in the literature is also quite wide. It covers classical logic [24],

many-valued logics [11, 4], generic many-valued modal logic [13], arbitrary finitely valued propositional logics [17], various types of programming logics [2, 11, 16], relative poly-modal logic [5, 12, 13], and temporal logic (Section 8).

Finally, it should be mentioned that since the R–S methodology is particularly well suited to relational logics, then a wide class of logics to which the R–S methodology is not directly applicable can be treated in an indirect way by either providing a relational semantics for them or translating them to (some variant of) relational logic, for which an R–S system is readily available. Some examples of logic that can be handled in this way are various kinds of modal logic [22], relevant logic [21] and intuitionistic logic [7].

Development of an R–S system for a logic: Assume we already have a logic satisfying the requirements set forth above, and now want to develop a complete R–S system for it. Then the strategy is as follows:

Development of decomposition rules. As we have already mentioned in Section 2.1, for each complex (i.e. decomposable) formula α, and each logical value k, we must have a decomposition rule expressing validity of the statement "the value of α is k" in terms of simultaneous validity of some sequences of formulae representing the statements "the value of α_i is k_i", where α_i is a formula simpler than α and k_i are some logical values. Only in this way we can ensure that each complex formula is broken into elementary ones in the decomposition process—which is the cornerstone of an R–S deduction system. Of course, such strategy can only be implemented if the semantic conditions quoted above are expressible as formulae of a given logic (for more discussion, see *Adapting a logical language for the use of R–S methodology* above).

When developing the rules, we must keep in mind the fact that the comma in sequences corresponds to meta-disjunction on the level of satisfaction in a single model incorporating a valuation, whereas branching in a decomposition rule corresponds to meta-conjunction on the validity level (satisfaction in all models). A more detailed discussion on decomposition rule development can be found in Section 2.1, and the mechanisms for handling quantification, modalities and relational type of axioms through decomposition rules are presented in Section 2.2.

Fundamental sequences. The choice of fundamental sequences is governed by one basic principle: they must be valid due to some simple semantic property of the logic, which is expressible by a simple syntactic condition on a sequence Ω. E.g., in classical logic the semantic property in question is the excluded middle law, and the corresponding syntactic condition is "for some α, both α and $\neg\alpha$ are in Ω". Fundamental sequences are usually determined by the basic semantic "axioms" of the logic that guarantee validity of some formulae or meta-disjunctions of formulae. Some laws of the logic, like properties of relations, can be expressed either through fundamental sequences or by expansion rules (see the relevant discussion in Section 2.2), and the choice is often decided by technical reasons connected with completeness proof.

Ensuring completeness. There are two prerequisites for ensuring completeness of an R–S deduction system. First, we must have enough decomposition rules to break up

every complex (in our terminology, decomposable) formula into simpler ones (see *Development of decomposition rules* above). Such rules are the replacement rules discussed in Section 2.2. Second, we must have enough expansion rules (see Section 2.2) and fundamental sequences to ensure that every indecomposable sequence that is valid will be recognised as such in the decomposition process (through giving rise to a fundamental sequence), and the respective branch of the tree will get closed. Indeed: expansion rules close indecomposable sequences under, e.g., reflexivity or transitivity rules for some relations appearing in the formulae, and then the resulting sequence—if valid—should fall into some class of fundamental sequences. The heuristics to achieve these is simply to look at the class of indecomposable formulae and check what types of sequences of such formulae must be valid—which is usually quite easy to determine in view of the simplicity of those formulae. Then we check if each such type of sequence is either some type of a fundamental sequence, or can be transformed to such a sequence through "closure" by an expansion rule. If not, we add new fundamental sequences and/or expansion rules to guarantee that.

Further options: As we have said before, the basic goal of the R–S methodology is to provide a logic with a sound and complete R–S deduction system. However, there are two optional further alternatives. One of them is transformation to a Gentzen sequent calculus to provide an interface to the sequent calculus environment, or in some cases get rid of the signed formulae introduced to enable development of an R–S system and return to the original language (see Sections 2.4 and 9). This is done using the practically automatic method described in Section 9. The second, much more important option, is

Use for automatic deduction: If we want to use the resulting R–S system for automated deduction purposes, then the elementary control mechanism embedded in the system, discussed in Sections 2.1, 2.2, and 2.5 must be replaced with something much more efficient. For example, instead of the simple decomposition order "from left to right", we should decompose formulae in an order that minimizes expansion of the decomposition tree. Thus, formulae requiring the use of nonbranching rules or rules with less branching should be decomposed before those requiring rules with more branching, and likewise those generating shorter sequences of formulae should be decomposed before those giving rise to longer ones. Universally quantified formuale should be decomposed before existentially quantified ones, and when choosing the terms for substituting into the latter we should start with those already appearing in the sequence, because this increases the chance of finding a match to yield a fundamental sequence and thus close the branch of the decomposition tree.

In case of fundamental sequences themselves, for the automatic deduction purposes there should be simple algorithm for determining whether a given sequence is fundamental or not. This means that fundamental sequences should be best defined on the indecomposable formulae level, because as a rule simplicity of such formulae in terms of both their depth and length greatly facilitates development of such effective decision algorithms.

Of course, what we have said above is just a small glance at the issues connected with proof automation, and much deeper considerations are needed in case of any practical implementation, However, because of the close affinity between R–S systems and tableau systems, a lot of ideas might come from looking at the automation methods used in case of tableau systems. In fact, this should be a fruitful area for further research.

4. Three-valued and many-valued logics

As the first example of an R–S system application, let us consider McCarthy's three-valued logic with noncommutative disjunction and conjunction. This calculus is of a special interest for computer science as it supports lazy, sequential computation. In fact, McCarthy's AND and OR have been used in the programming languages Euclid, Algol-W and C (see [3, 19] for more discussion on the subject). If we denote the three logical values employed there by **tt** (truth), **ff** (falsity), and **ee** (undefinedness or error), then the truth tables of McCarthy's connectives are as follows:

NOT	
tt	**ff**
ff	**tt**
ee	**ee**

OR	**tt**	**ff**	**ee**
tt	**tt**	**tt**	**tt**
ff	**tt**	**ff**	**ee**
ee	**ee**	**ee**	**ee**

AND	**tt**	**ff**	**ee**
tt	**tt**	**ff**	**ee**
ff	**ff**	**ff**	**ff**
ee	**ee**	**ee**	**ee**

From the above truth tables we can see that McCarthy's disjunction and conjunction are not commutative. This rather unusual property can be, however, easily explained if we interpret **ee** as corresponding to undefinedness in the sense of infinite computation.

If we assume that the computation is sequential and lazy, and the value of a formula is computed from left to right, then obviously $\mathbf{ee} \vee x = \mathbf{ee} \wedge x = \mathbf{ee}$ for any logical value x. Indeed if the computation to evaluate a formula α loops, then the evaluation of $\alpha \vee \beta$ and $\alpha \wedge \beta$ loops too, and so both the formulae should be assigned the value **ee**, no matter what the value of β is—because we will never get to β anyway. So e.g. $\mathbf{ee} \wedge \mathbf{ff} = \mathbf{ff}$. On the other hand, $\mathbf{ff} \wedge \mathbf{ee} = \mathbf{ff}$, since in a lazy computation after evaluating the first argument of a disjunction to **ff** we do not look at the second argument at all and just accept **ff** as the result, because "false and anything is false". The asymmetry follows from the fact that in case of $\mathbf{ee} \wedge \mathbf{ff}$ we will never learn that **ff** is there at all!

To develop an R–S deduction system for such a logic, we use signed formulae (or, alternately, an "is-true" superpredicate **T** mapping three-valued predicates into two-valued ones, and its classical negation $\neg \mathbf{T} = \mathbf{N}$—see [11]). If we accept **tt** as the only designated value, then the semantics of signed formulae is as follows:

$$|\mathbf{T}(\alpha)| = \begin{cases} \mathbf{tt} & \text{iff } |\alpha| = \mathbf{tt}, \\ \mathbf{ff} & \text{iff } |\alpha| \in \{\mathbf{ff}, \mathbf{ee}\}, \end{cases} \qquad |\mathbf{N}(\alpha)| = \begin{cases} \mathbf{tt} & \text{iff } |\alpha| \in \{\mathbf{ff}, \mathbf{ee}\}, \\ \mathbf{ff} & \text{iff } |\alpha| = \mathbf{tt}, \end{cases}$$

where $|\cdot|$ represents the value of a formula in a model (incorporating a valuation). Hence, we can "encode" all the possible values of a three-valued formula in a two-valued world by remarking that

$$|\alpha| = \mathbf{tt} \quad \text{iff} \quad |\mathbf{T}(\alpha)| = \mathbf{tt}, \qquad |\alpha| = \mathbf{ff} \quad \text{iff} \quad |\mathbf{T}(\text{NOT}\alpha)| = \mathbf{tt},$$
$$|\alpha| = \mathbf{ee} \quad \text{iff} \quad |\mathbf{N}(\alpha)| = \mathbf{tt} \text{ and } |\mathbf{N}(\text{NOT}\alpha)| = \mathbf{tt}. \tag{1}$$

This allows us to develop an R–S deduction system for a language featuring McCarthy's connectives. From the above relationships it clearly follows that in case of a three-valued logic (not necessarily McCarthy's one) we need four decomposition rules for each formula constructor: one for the resulting formula φ to be true, one for it to be not true, one for NOT φ to be true (that is, for φ to be false), and one for NOT φ to be not true. Indeed: to develop an R–S system for the logic, we must somehow express the semantic condition $|\alpha| = k$ for each formula α and each logical value $k \in \{\mathbf{tt}, \mathbf{ff}, \mathbf{ee}\}$—and from Eq. (1) it clearly follows that for this purpose we have to consider all the four combinations of $\mathbf{T}/\mathbf{N}$ applied to ι/NOT, when ι denotes identity. Evidently, the underlying reason is that in the context of three valued, and in general many-valued logic, "not true" does not coincide with "false".

It should be noted that three-valued quantifiers (e.g. Kleene's ones) can be dealt with in much the same way (see [11]).

As an example, let us give the four rules describing McCarthy's disjunction. A complete set of decomposition rules for the logic is given in [11]:

$$\frac{\Omega', \mathbf{T}(\alpha \text{ OR } \beta), \Omega''}{\Omega', \mathbf{T}(\alpha), \mathbf{T}(\text{NOT } \alpha), \Omega'' \,|\, \Omega', \mathbf{T}(\alpha), \mathbf{T}(\beta), \Omega''}$$

$$\frac{\Omega', \mathbf{N}(\alpha \text{ OR } \beta), \Omega''}{\Omega', \mathbf{N}(\alpha), \Omega'' \,|\, \Omega', \mathbf{N}(\text{NOT } \alpha), \mathbf{N}(\beta), \Omega''}$$

$$\frac{\Omega', \mathbf{T}(\text{NOT}(\alpha \text{ OR } \beta)), \Omega''}{\Omega', \mathbf{T}(\text{NOT } \alpha), \Omega'' \,|\, \Omega', \mathbf{T}(\text{NOT } \beta), \Omega''} \qquad \frac{\Omega', \mathbf{N}(\text{NOT}(\alpha \text{ OR } \beta)), \Omega''}{\Omega', \mathbf{N}(\text{NOT } \alpha), \mathbf{N}(\text{NOT } \beta), \Omega''}$$

By way of example, let us see how rule ($\mathbf{T}$ OR) is developed. According to the truth tables, $|\alpha \text{ OR } \beta| = \mathbf{tt}$ iff either $|\alpha| = \mathbf{tt}$, or $|\alpha| = \mathbf{ff}$ and $|\beta| = \mathbf{tt}$. To express that "and", we need either AND or the branching $|$ in the premises. The first is not advisable, since α OR β should be decomposed into formulae simpler than itself. To use branching, we must express the condition C above as "C1 and C2", where C1, C2 do not contain "and". It is easy to see that C is equivalent to "($|\alpha| = \mathbf{tt}$ or $|\alpha| = \mathbf{ff}$) and ($|\alpha| = \mathbf{tt}$ or $|\beta| = \mathbf{tt}$)". Translating the individual semantic conditions into formulae with help of Equation (1), replacing "and" by branching and "or" by the comma in sequences of formulae, we get just the discussed rule.

It should be added that the R–S mechanism can be used equally well for formalizing many-valued logic in general. For example, in [14] it has been used to provide a complete deduction system for Rosser–Turquette many-valued logic, and in [17]—to provide a relational style formalization of an arbitrary many-valued propositional logic.

5. Similarity, indiscernibility and complementarity logic

The second area in which the R–S system has been successfully applied is that of logics for reasoning about certain relations between objects considered in information systems, like e.g. indiscernibility, similarity or complementarity.

Let us begin with similarity. By a similarity relation we mean a reflexive and symmetric binary relation on a set of entities (see [12, 13]). Given a set of entities and a set of properties for classifying the entities, we have a whole family of similarity relations: one for each subset of the set of properties. Formally, the semantic framework we work with is a *universe* U of the form

$$U = \langle \textsc{Ent},\ \textsc{Prop}, \{sim(P)\}_{P\subseteq \textsc{Prop}}\rangle, \tag{2}$$

where Ent is a set of *entities*, Prop—a nonempty set of *properties*, and $\{sim(P)\}_{P\subseteq \textsc{PROP}}$ is a family of *similarity relations* on Ent such that:
(C1) Each $sim(P)$ is a reflexive and symmetric binary relation on Ent;
(C2) $sim(\emptyset) = \textsc{Ent} \times \textsc{Ent}$;
(C3) $sim(P \cup Q) = sim(P) \cap sim(Q)$ for any $P, Q \subseteq \textsc{Prop}$.
Within this framework, we consider the operations of the *upper* and *lower approximation* of a set $E \subseteq \textsc{Ent}$ with respect to the relation $sim(P)$:

$$\begin{aligned} \overline{\mathbf{sim}}(P)E &= \{e \in \textsc{Ent}:\ (\exists e')((e,e') \in sim(P) \text{ and } e' \in E)\},\\ \underline{\mathbf{sim}}(P)E &= \{e \in \textsc{Ent}:\ (\forall e')((e,e') \in sim(P) \text{ implies } e' \in E)\}. \end{aligned} \tag{3}$$

Thus $\underline{\mathbf{sim}}(P)E$ consists of all the entities which are not similar with respect to the properties in P to any entity outside E, and $\overline{\mathbf{sim}}(P)E$—of all the entities which are similar to some entity in E. The operation of upper approximation contains implicit existential quantification, whereas that of lower approximation—implicit universal quantification.

The language is built over a fixed set Prop of properties. It contains terms in Term representing subsets of Prop, and formulae in Form representing subsets of Ent. The set Term of terms contains:
- $\textsc{Conp} = \{\mathbf{p}:\ p \in \textsc{Prop}\}$—constants representing individual properties in Prop,
- a constant $\mathbf{0}$ for the empty set of properties $\emptyset$,
- Varsp—variables representing subsets of Prop.

It is closed under $-, \cup, \cap$ representing set-theoretic operations on sets of properties.

The set Form of formulae contains:
- Vare—the set of variables representing individual entities,
- Varse—the set of variables representing sets of entities.

It is closed under $\neg, \vee, \wedge$, denoting set-theoretic operations on sets of entities. Finally, if $A \in \textsc{Term}$ and $F \in \textsc{Form}$, then $\underline{sim}(A)F, \overline{sim}(A)F \in \textsc{Form}$, where $\underline{sim}$ and $\overline{sim}$ denote the lower and upper approximation.

We also use a derived constructor $\rightarrow$ defined by

$$F \rightarrow G \stackrel{\text{df}}{=} \neg F \vee G.$$

The language is interpreted in models $M = \langle U, v\rangle$ consisting of a universe and a (many-sorted) valuation of variables. Of course, $\underline{sim}$ and $\overline{sim}$ are interpreted as the operations of lower and upper approximation, respectively; terms are interpreted as subsets of PROP, and formulae—as subsets of the set ENT of entities.

A formula F is said to be satisfied in a model iff it evaluates to the whole set ENT of this model.

The above logic is a polymodal logic, with relative modalities represented by $\underline{sim}(A)$ and $\overline{sim}(A)$. It should be noted that the Rasiowa–Sikorski system presented here was the first axiomatization of this logic, after the problem had been open for several years.

It is not dichotomic, since usually neither F nor $\neg F$ holds. To compensate for this, we base deduction on formulae of the type

$$x \in F \stackrel{\text{df}}{\equiv} x \to F, \tag{4}$$

where $x \in$ VARE and F is any formula. They model set-theoretic membership relation, since $x \in F$ is satisfied in $M = \langle U, v\rangle$ iff $v(x) \in |F|_M$. Such formulae have the desired dichotomic property: for any model M, either $M \models x \in F$ or $M \models x \in \neg F$. A special kind of type (4) formulae with $F \equiv \overline{sim}(A)\ y$, denoted shortly by $x\ sim(A)\ y$, express the similarity relation itself, since $x\ sim(A)\ y$ holds in $M = \langle U, v\rangle$ iff $(v(x), v(y)) \in sim(|A|_M)$.

We use the signed formulae to represent "top-level" negation. It suffices to consider signed formulae over formulae of type (4) only, since we can reduce all other formulae to the former with the following decomposition rules:

$$\frac{\Omega', \mathbf{T}(F), \Omega''}{\Omega', \mathbf{T}(y \in F), \Omega''} \qquad \frac{\Omega', \mathbf{N}(F), \Omega''}{\Omega', \mathbf{N}(x \in F), \Omega'', \mathbf{N}(F)}$$

where $x, y \in$ VARE, y does not occur above the double line, and F is not of the form $z \in G$.

The above rules show a simple, yet ingenious mechanism used to deal with implicit quantification: substitution of a new variable in case of universal quantification, and subsequent substitutions of arbitrary variables (note that we do not have any "entity" terms here) while copying the original formula in case of existential quantification. This is just a certain variant of the classic way of dealing with quantification in an R–S system mentioned before and described in [24].

Now let us give some examples of how certain basic features of the similarity framework are represented in the deduction system.

Property (C2), saying that similarity wrt the empty set of properties is the universal relation, and reflexivity of similarity stated in (C1) are represented by assuming that:

> Each sequence Ω containing either $\mathbf{T}(x\ sim(\mathbf{0})\ y)$ or $\mathbf{T}(x\ sim(A)\ x)$ is fundamental. (5)

In turn, the other part of (C1)—symmetry of similarity relations—is represented by a typical expansion rule of the form

$$\frac{\Omega', \mathbf{N}(x\ sim(A)\ y), \Omega''}{\Omega', \mathbf{N}(x\ sim(A)\ y), \mathbf{N}(y\ sim(A)\ x), \Omega''} \tag{6}$$

Finally, property (C3) is represented by the rules

$$\frac{\Omega', \mathbf{T}(x\ sim(A \cup B)\ y), \Omega''}{\Omega', \mathbf{T}(x\ sim(A)\ y), \Omega'' \mid \Omega', \mathbf{T}(x\ sim(B)\ y), \Omega''}$$

$$\frac{\Omega', \mathbf{N}(x\ sim(A \cup B)\ y), \Omega''}{\Omega', \mathbf{N}(x\ sim(A)\ y), \mathbf{N}(x\ sim(B)\ y), \Omega''} \tag{7}$$

The relative modalities $\underline{sim}(A), \overline{sim}(A)$ are dealt with by the following rules:

$$\frac{\Omega', \mathbf{T}(x \in \overline{sim}(A)F), \Omega''}{\Omega', \mathbf{T}(y \in F), \Omega'', * \mid \Omega', \mathbf{T}(x\ sim(A)\ y), \Omega'', *} \qquad \frac{\Omega', \mathbf{N}(x \in \overline{sim}(A)F), \Omega''}{\Omega', \mathbf{T}(x \in \underline{sim}(A)\neg F), \Omega''}$$

$$\frac{\Omega', \mathbf{T}(x \in \underline{sim}(A)F), \Omega''}{\Omega', \mathbf{N}(x\ sim(A)z), \mathbf{T}(z \in F), \Omega''} \qquad \frac{\Omega', \mathbf{N}(x \in \underline{sim}(A)F), \Omega''}{\Omega', \mathbf{T}(x \in \overline{sim}(A)\neg F), \Omega''}$$

where y is an arbitrary variable in VARE, $*$ denotes $\mathbf{T}(x \in \overline{sim}(A)F)$, and $z \in$ VARE does not occur above the double line. To see how the $\mathbf{T}(\overline{sim})$ rule was developed, let us note (using for simplicity a rather informal notation) that $x \in \overline{sim}(A)F$ if there exists a y such that $y \in F$ and $x\ sim(A)\ y$. As "and" corresponds to the branching $|$, then using the method for handling existential quantification known from classical logic we get just the presented rule. Justification for the $\mathbf{T}(\underline{sim})$ rule is analogous, but based on universal quantification over z. In turn, the **N**-rules just reflect the duality of the lower and upper approximations.

To cope with the set parameters A in modalities, before applying our deduction system we replace each A by a union of atomic "components" that evaluate to a disjoint cover of PROP in every model. In case of elementary formulas of the form $\mathbf{T}(x\ sim(A)\ y)$, $\mathbf{N}(x\ sim(A)\ y)$, we can break down a term A being the union of some components into individual components using rules (7).

All the mechanisms outlined above allow us to get a complete deduction system for the polymodal similarity logic (see [12, 13]). Such result obtained using the R–S methodology is especially valuable in view of the fact that a polymodal logic with interdependent relative modalities is particularly difficult to handle—and hence previous work on indiscernibility, similarity or discernibility logic was as a rule limited to considering a single, nonparametrised modality. The same way—as we will see below—can also be used for other information system logics; it should be also applicable to other polymodal logics of that type in general.

Out of the system discussed above, it is rather easy to get a complete system for reasoning about indiscernibility logic. Indeed, the only difference between similarity and

indiscernibility relations is that the latter are transitive whereas the former in general are not. Hence, it can be shown that after replacing *sim* with *ind* (meaning indiscernibility) in the R–S system for the similarity logic presented here, and augmenting the system with an expansion rule expressing transitivity of indiscernibility relations, namely

$$\frac{\Omega', \mathbf{N}(x\ ind(A)\ y), \mathbf{N}(y\ ind(A)\ z), \Omega''}{\Omega', \mathbf{N}(x\ ind(A)\ y), \mathbf{N}(y\ ind(A)\ z), \mathbf{N}(x\ ind(A)\ z), \Omega''}$$

we get a complete deduction system for indiscernibility logic. It should be noted that the rules for modalities remain unchanged.

Now let us turn to complementarity. This type of relations are defined in the context of an information system having the form

$$I = (\textsc{Obj}, \textsc{Attr}, \{V_a\}_{a \in \textsc{Attr}}),$$

where Obj is a set of *objects*, and Attr is a set of *attributes* such that each $a \in \textsc{Attr}$ is a mapping

$$a : \textsc{Obj} \to \mathscr{P}(V_a)$$

which assigns to each object a set of values associated with attribute a.

For any $A \subset \textsc{Attr}$, by the complementarity relation corresponding to A we mean the binary relation $comp(A)$ on Obj defined by

$$(x, y) \in comp(A) \text{ if and only if } a(x) = V_a \backslash a(y) \text{ for all } a \in A.$$

Formally, the complementarity framework we work with is

$$U = \langle \textsc{Obj}, \textsc{Attr}, \{comp(A)\}_{A \subset \textsc{Attr}} \rangle,$$

where $\{comp(A)\}_{A \subset \textsc{Attr}}$ is the family of relative complementarity relations on Obj characterized by the following properties:

(D1) Each $comp(A)$ is an irreflexive, symmetric and 3-transitive relation, i.e. for all $x, y, z, w \in \textsc{Obj}$,

$$(x, y), (y, z), (z, w) \in comp(A) \text{ implies } (x, w) \in comp(A).$$

(D2) $comp(\emptyset) = \emptyset$.

(D3) $comp(A \cup B) = comp(A) \cap comp(B)$ for any $A, B \subseteq \textsc{Attr}$.

As we see, some of the properties of complementarity relations are essentially different from those of similarity relations. Despite this, our deduction system DRS can be modified to yield a complete deduction system for a polymodal logic based on complementarity relations.

A purely syntactic change is to replace *sim* by *cp*, standing for complementarity. To express property (D2), saying that complementarity wrt the empty set of properties is the empty relation, and the part of (D1) saying that complementarity is irreflexive, we

replace the old assumption (5) with

> Each sequence Ω containing either $\mathbf{N}(x\ cp(\mathbf{0})y)$ or $\mathbf{N}(x\ cp(A)x)$ is fundamental.

The property of symmetry remains unchanged, and (D3) just mirrors (C3). The only remaining task is to model 3-transitivity being part of (D1), which is done by introducing the rule

$$\frac{\Omega', \mathbf{N}(x\ cp(A)y), \mathbf{N}(y\ cp(A)z), \mathbf{N}(z\ cp(A)w), \Omega''}{\Omega', \mathbf{N}(x\ cp(A)y), \mathbf{N}(y\ cp(A)z), \mathbf{N}(z\ cp(A)w), \mathbf{N}(x\ cp(A)w), \Omega''}$$

It can be proved (see [5]) that the deduction system obtained out of DRS as a result of the above modifications is complete for complementarity logics. Note that, similarly as in case of indiscernibility relations, the rules for modalities remain unchanged.

It should be noted that complementarity logic is an example of the R–S methodology allowing to develop complete systems for modal logic with modally inexpressible conditions on the accessibility relation. Indeed, since irreflexivity is not modally expressible, then in [6] a Hilbert-style axiomatisation of complementarity logics was impossible without introducing not very natural modal operators of "sufficiency".

6. A logic for nondeterministic specifications

The next example is a logic for nondeterministic specifications presented in [2, 16]. It is interpreted in multialgebras, i.e. algebras with set-valued functions. The basic logical language has only two constructs: the "**let**" construct used for limiting nondeterminism, and a "rewrite" operator $\rightarrow$ interpreted as set inclusion. Notwithstanding these strictly limited means (no logical connectives at all, not even negation), it is still possible to provide a simple and elegant complete deduction system for the language using the R–S method.

In fact, this application shows a decided advantage of the R–S methodology in the sense of its simplicity and power, since the only two complete deduction systems of the more widespread types presented earlier in [9, 25] for such nondeterministic specification logics were based on rather unnatural restrictions on the model level—whereas the R–S system presented here is complete for the class of arbitrary models.

Now let us introduce some basic notions.

Let $\Sigma = [S, F]$ be an algebraic signature, where S is the set of sorts, and F the set of function symbols, and let X be an S-sorted set of variables.

- A multialgebra over Σ is a tuple $A = [S^A, F^A]$, where $S^A = \{A_s\}_{s\in S}$ is a family of carrier sets for each sort $s \in S$ and $F^A = \{f^A\}_{f\in F}$ is a family of set-valued functions such that $f^A: A_{s_1} \times \cdots \times A_{s_n} \rightarrow P(A_s)$ for each $[f: s_1 \times \cdots \times s_n \rightarrow s] \in F$.
- An n-term over Σ is any term built over variables in X with help of the function symbols $f \in F$ and the [let $x := t'$ in t ni] construct.

- A formula is any expression of the form $t \to t'$, where t, t' are n-terms.

The set of all n-terms over Σ and X is denoted by $NT_\Sigma(X)$, and the set of all formulae—by $F_\Sigma(X)$.

The language is interpreted in models of the form $M = \langle A, v\rangle$, where A is a multi-algebra and $v : X \to |A|$ is a (multi-sorted) valuation of variables.

For any $x \in X$, $a \in |A|$, define

$$v[a/y](x) = \begin{cases} a & \text{iff } x = y, \\ v(x) & \text{otherwise.} \end{cases}$$

The interpretation of n-terms in $M = \langle A, v\rangle$ is a function $\bar{v} : NT_\Sigma(X) \to P(|A|)$ defined by

- $\bar{v}(x) = \{v(x)\}$, $\quad \bar{v}(f(t_1, \ldots, t_n)) = \bigcup_{a_i \in \bar{v}(t_i)} f^A(a_1, \ldots, a_n)$,
- $\bar{v}([\mathsf{let}\ x := t'\ \mathsf{in}\ t\ \mathsf{ni}]) = \bigcup_{a \in \bar{v}(t')} \overline{v[a/x]}(t)$.

Thus, the usual substitution of functions is interpreted by taking a set-theoretic union over all possible combinations of values, whereas the $[\mathsf{let}\, x := t'\ \mathsf{in}\ t\ \mathsf{ni}]$ construct limits nondeterminism and allows us to substitute the same value of t' for every occurrence of x in t.

Since the arrow operator is interpreted as inclusion, with the arrow pointing 'down' to the smaller set, then satisfaction of a formula $t \to t'$ in $M = \langle A, v\rangle$ is defined by

$$M \models t \to t' \quad \text{iff } \bar{v}(t) \supseteq \bar{v}(t').$$

To develop an R–S deduction system for this logic, we proceed analogously as in the case of similarity logics: we base deduction on the membership relation expressed by formulae of the type $t \to x$, where $x \in X$, and introduce signed formulae in order to have top-level negation.

Semantically, the inclusion $\mathbf{T}(t \to t')$ represents universal quantification of the form "for every x, if $x \in t'$, then $x \in t$". Analogously, $\mathbf{N}(t \to t')$ represents existential quantification of the form "there exists an x such that $x \in t'$ and $x \notin t$". This is modelled, as other instances of quantification before, by an appropriate use of variables (see Section 2.2). Thus we have the following rules:

$$\frac{\Omega', \mathbf{T}(t_1 \to t_2), \Omega''}{\Omega', \mathbf{N}(t_2 \to x), \mathbf{T}(t_1 \to x), \Omega''} \qquad \frac{\Omega', \mathbf{N}(t_1 \to t_2), \Omega''}{\Omega', \mathbf{T}(t_2 \to y), \Omega'', * \mid \Omega', \mathbf{N}(t_1 \to y), \Omega'', *}$$

where t_2 is not a variable in X, $x \in X$ is a variable which does not appear in the conclusion, y is an arbitrary variable in X, and * denotes $\mathbf{N}(t_1 \to t_2)$.

The rules corresponding to normal substitution and to the **let** construct demonstrate clearly the difference between the two:

$$\frac{\Omega', \mathbf{T}(f(\cdot, t_i, \cdot) \to x), \Omega''}{\Omega', \mathbf{T}(t_i \to y), \Omega'', * \mid \Omega', \mathbf{T}(f(\cdot, y, \cdot) \to x), \Omega'', *}$$

$$\frac{\Omega', \mathbf{N}(f(\cdot, t_i, \cdot) \to x), \Omega''}{\Omega', \mathbf{N}(t_i \to z), \mathbf{N}(f(\cdot, z, \cdot) \to x), \Omega''}$$

$$\frac{\Omega', \mathbf{T}([\text{let } x := l \text{ in } t \text{ ni}] \to x'), \Omega''}{\Omega', \mathbf{T}(l \to y), \Omega'', * \mid \Omega', \mathbf{T}(t[y/x] \to x'), \Omega'', *}$$

$$\frac{\Omega', \mathbf{N}([\text{let } x := l \text{ in } t \text{ ni}] \to x'), \Omega''}{\Omega', \mathbf{N}(l \to z), \mathbf{N}(t[z/x] \to x'), \Omega''}$$

where y is an arbitrary variable in X, $z \in X$ is a variable which does not appear in the conclusion, and $*$ denotes repetition of the decomposed formula from the conclusion in the premises.

The complete set of decomposition rules for the logic is given in [16].

7. Many-sorted first-order logic with possibly empty carriers of some sorts

It is well-known that classical deduction rules for first-order logic become unsound in presence of empty carriers of some sorts. Indeed, if we interpret universal quantification over an empty set of values as trivially true, and existential quantification over such a set—as trivially false, then $\forall x.\varphi$ implies neither $\varphi(z)$ nor $\exists x.\varphi$, which contradicts classical entailment. As empty carriers do appear as an useful tool in software specification, providing a complete deduction system for this case is an important task. The hitherto existing approaches (see [4, 23, 26]) are based on rather rich languages, involving concepts like dynamic sorts, and in general diverge quite far from the simplicity of classical first-order logic.

However, using the R–S methodology one can formalize a many-sorted FOL allowing for empty carriers in a quite simple way—which again stresses the advantage of that methodology in providing complete deduction systems based on simple means. The resulting logic is very close to classical FOL, and the main idea consists in preceding formulae with a context being a subset of the set of sorts. Such a contextual formula has a defined value only in models with nonempty carriers of the sorts contained in its context.

We assume a finite, nonempty set S of sorts (sort names), to be fixed for the sequel, and existence of a fixed sort b of logical values. Based on the above, we define a many-sorted FO language L_S.

For $s \in S$, let X_s be a countable set of individual variables of sort S, with

$$X_s \cap X_{s'} = \emptyset \quad \text{for } s \neq s', \quad X = \bigcup_{s \in S} X_s.$$

The unique sort of a variable $x \in X$ is denoted by $s.x$.

As before, by S^* we denote the set of all finite sequences of the elements of S; the empty sequence is denoted by 0.

Assume we are given sets $F_{w,s}$ $(w \in S^*, s \in S)$ of function symbols of arity $f : w \to s$, and sets P_w $(w \in S^*)$ of predicate symbols of arity $w \to b$.

The set $\mathscr{T}_S = \bigcup_{s \in S} \mathscr{T}_s$ of terms of the language $\mathscr{L}_S$, where $\mathscr{T}_s$ is the set of terms of sort s, is defined in the standard way over the sets of variables and function symbols

given above. To designate a term as a term of sort s, we use the sort symbol as a superscript.

The minimal context of a term t is defined by

$$C_{\min}(t) = \{s.x\colon\ x \in X \text{ and } x \text{ occurs in } t\}.$$

For simplicity, contexts are identified with their denotations, which involves writing things like "$C \subseteq S$" or "$s \in C$".

The set $\mathscr{P}\mathscr{F}_S$ of pre-formulae of the language $\mathscr{L}_S$ is the set of all first-order formulae defined in the usual way over the appropriate signature. The minimal context of a pre-formula φ is the context

$$C_{\min}(\varphi) = \{s.x\colon\ x \in \mathit{Free}(\varphi)\},$$

where $\mathit{Free}(\varphi)$ is the set of all variables having at least one free occurrence in the formula φ.

By a formula of the language $\mathscr{L}_S$ we mean every expression of the form $C.\varphi$, where $C \subseteq S$ is a context, $\varphi \in \mathscr{P}\mathscr{F}_S$ is a pre-formula, and $C \supseteq C_{\min}(\varphi)$.

The set of all formulae of $\mathscr{L}_S$ is denoted by $\mathscr{F}_S$.

Thus each formula is of the form $C.\varphi$, where $C \subseteq S$ is a context, and φ is a "normal" (many-sorted) first-order formula. Such a formula is interpreted in presence of a valuation defined for all variables of the sorts in C, and it has no value in a model where the carriers of some of the above sorts are empty.

The language is interpreted in models based on S-sorted structures of the form

$$\mathscr{A} = \langle \{A_s\}_{s\in S}, \mathbf{F}, \mathbf{P}\rangle$$

consisting of a carrier A_s (possibly empty) for each sort together with interpretations $\mathbf{F}, \mathbf{P}$ of function and predicate symbols consistent with their arities. We assume that the fixed sort b of Boolean values has a carrier B of the form $B = \{\mathbf{tt}, \mathbf{ff}\}$, where $\mathbf{tt}$ denotes truth, and $\mathbf{ff}$ falsity.

We also assume that all the function and predicate symbols are interpreted as total functions and predicates. Because of this, the minimal contexts defined above are indeed sufficient for the respective terms/formulae to have defined values.

Moreover, the above assumption imposes certain limitations on the pattern of empty carriers in the structures for our language. Namely, if the language contains a constant $f\colon 0 \to s$ of sort s, then the carrier A_s of sort s must be nonempty in each model M. Further, if the language contains a function symbol $f\colon s \to s'$ and the carrier $A_{s'}$ is empty in a model M, then the carrier A_s must also be empty in the model M—for otherwise the function symbol f could not be interpreted as a total function on the carrier of s. These facts will find their reflection in the deduction system.

The language is interpreted in models of the form $M = \langle \mathscr{A}, v\rangle$, where $\mathscr{A}$ is an S-sorted structure, and v is an S-sorted valuation of variables in $\mathscr{A}$, defined as a partial mapping $v\colon X \to \bigcup_{s\in S} A_s$, with

$$\mathit{Dom}(v) = \bigcup\{X_s\colon\ A_s \neq \emptyset\}, \quad v(x^s) \in A_s \quad \text{for any } x^s \in \mathit{Dom}(v).$$

Due to possible emptiness of carriers and introduction of contexts, both the interpretation of terms and interpretation of formulae are partial functions.

The interpretation of terms in a model $M=\langle\mathscr{A},v\rangle$ is a partial function $\mathbf{T}_M:\mathscr{T}_S\to\bigcup_s A_s$ such that

$$Dom(\mathbf{T}_M)=\{t\in\mathscr{T}:\ (\forall s\in C_{\min}(t))A_s\neq\emptyset\},\quad \mathbf{T}_M(\mathscr{T}_s)\subset A_s\quad \text{for } s\in S,$$

defined in the standard way by extending the interpretation of variables given by v to all terms through interpreting the function symbols according to their interpretation in the structure $\mathscr{A}$.

If $M=\langle\mathscr{A},v\rangle$ is a model, $x\in X_s$ and $a\in A_s$, then by $M[a/x]$ we denote the model $M[a/x]=\langle\mathscr{A},v[a/x]\rangle$ with $v[a/x]$ defined as before.

The interpretation of formulae in a model $M=\langle\mathscr{A},v\rangle$ is a partial function $\mathbf{F}_M:\mathscr{F}_S\to\{\mathbf{tt},\mathbf{ff}\}$ such that $Dom(\mathbf{F}_M)=\{C.\varphi\in\mathscr{F}_S:\ (\forall s\in C)A_s\neq\emptyset\}$ defined inductively as follows:

1. for formulae without quantifiers, $\mathbf{F}_M(C_{.\varphi})$ is defined on the basis of the interpretation $\mathbf{T}_M$ of terms using the standard interpretation of logical connectives;
2. $$\mathbf{F}_M(C.(\forall x.\varphi))=\begin{cases}\mathbf{tt} & \text{if either } A_{s.x}=\emptyset \text{ or}\\ & A_{s.x}\neq\emptyset \text{ and } \mathbf{F}_{M[a/x]}.(C\cup\{s.x\}).\varphi=\mathbf{tt}\\ & \text{for every } a\in A_{s.x},\\ \mathbf{ff} & \text{otherwise,}\end{cases}$$
3. $$\mathbf{F}_M(C.(\exists x.\varphi))=\begin{cases}\mathbf{tt} & \text{if } A_{s.x}\neq\emptyset \text{ and } \mathbf{F}_{M[a/x]}.(C\cup\{s.x\}).\varphi=\mathbf{tt}\\ & \text{for some } a\in A_{s.x},\\ \mathbf{ff} & \text{otherwise.}\end{cases}$$

$\mathbf{F}_M$ is extended to other formulae with quantifiers by interpreting the logical connectives in the standard way.

To develop an R–S system for $\mathscr{L}_S$, we assume that our language contains a logical constant $c:0\to b$, and define special 'emptiness' and 'nonemptiness' formulae' E_s, N_s for each sort $s\in S$ by

$$E_s\stackrel{\text{df}}{=}\emptyset.\forall x^s.c\wedge\neg c,\quad N_s\stackrel{\text{df}}{=}\emptyset.\exists x^s.c\vee\neg c$$

where x^s is any variable in X_s, and $\emptyset$ denotes the empty context.

Then, for every model $M=\langle\mathscr{A},v\rangle$, we have

$$M\models E_s\quad\text{iff } A_s=\emptyset,\quad M\models N_s\quad\text{iff } A_s\neq\emptyset,$$

where A_s is the carrier of sort s in structure $\mathscr{A}$.

The above formulae play a key role in the deduction system, allowing us to deal with quantification in presence of empty carriers.

Each sequence $\Omega=\alpha_1,\alpha_2,\ldots,\alpha_n$ containing either

1. both the formulae E_s,N_s for some $s\in S$, or

2. each of the formulae $E_{s_1},\ldots,E_{s_n},C.\varphi,C.\neg\varphi$, where $C=\{s_1,s_2,\ldots,s_n\}$ and $\varphi\in\mathscr{P}\mathscr{F}_S$, is considered fundamental. Since E_s is in fact a (meta-)negation of N_s, the first type of a fundamental sequence can be seen as a special instance of that for classical logic (defined as containing both φ and $\neg\varphi$). The second type of fundamental sequence is a general analogue of the latter for our logic, reflecting the fact that here "either $C.\varphi$ or $C.\neg.\varphi$" holds only if the carriers of all sorts in C are nonempty.

The rules for connectives also mirror the classical ones, with only the context thrown in. For quantifiers, we have new, double rules: one, quite classical, for quantification over a sort with nonempty carrier, and another, nonclassical, for quantification over a sort with possibly empty carrier. The latter ones reflect the fact that universal quantification over an empty carrier is trivially true, whereas the existential one—trivially false.

$$(\forall 1)\ \frac{\Omega',C.\forall x.\varphi,\Omega''}{\Omega',C.\varphi(z),\Omega''}\quad \text{where } s.x\in C \qquad (\forall 2)\ \frac{\Omega',C.\forall x.\varphi,\Omega''}{\Omega',E_{s.x},(C\cup\{s.x\}).\varphi(z),\Omega''}\quad \text{where } s.x\notin C$$

and z is a variable which does not appear above the double line

$$(\exists 1)\ \frac{\Omega',C.\exists x.\varphi,\Omega''}{\Omega',C.\varphi(t/x),\Omega'',C.\exists x.\varphi}\quad \text{where } s.x\in C \qquad (\exists 2)\ \frac{\Omega',C.\exists x.\varphi,\Omega''}{\Omega',N_{s.x},\Omega''\mid\Omega',(C\cup\{s.x\}).\varphi(t/x),\Omega'',*}\quad \text{where } s.x\notin C$$

where $*$ stands for $C.\exists x.\varphi$ and t is any term free for x in φ.

We also have a rule for context manipulation, and a rule reflecting the limitations on the pattern of empty carriers imposed by the language:

$$(Cr)\ \frac{\Omega',(C\cup\{s\}).\varphi,\Omega''}{\Omega',N_s,\Omega''|\Omega',C.\varphi,\Omega''}\qquad (Etr)\ \frac{\Omega',E_{s_1},\ldots,E_{s_k},\Omega''}{\Omega',E_{s_1},\ldots,E_{s_k},E_s,\Omega''}$$

where $C\supset C_{\min}(\varphi),\varphi$ is a literal, and $s_1,\ldots,s_k,\ s\in S$ are such that $F_{s_1,\ldots,s_k,s}\neq\emptyset$. The full set of rules, as well as the soundness and completeness proofs, are given in [15].

As a final remark, let us note that since the decomposition rules of our logic diverge from the classical ones in Section 2.1 only in a simple modification of the quantifier rules, and an additional context-handling rule, the proof search/generation mechanism used for classical many-sorted logics should be easily adaptable for the logic considered here, thus providing a good proof search basis for software specification/validation logics.

8. Three-valued temporal logic for reasoning about concurrency

Following the above overview of earlier applications of R–S systems in SC logic, let us now pass to a quite new application of this mechanism—namely, its use to formalize a three-valued approach to temporal logic for reasoning about concurrency.

This application is doubly novel in the sense that the hitherto existing temporal logics for reasoning about concurrency have been as a rule based on a two-valued approach to semantics. However, there seems to be a clear motivation for introducing a third logical value **uu** interpreted as "unknown". Indeed if we consider semantics of concurrent systems based on local states, then it is quite natural that in state s we can assign definite logical values—**tt** (*true*) or **ff** (*false*)—only to those variables that are local to the state s. For example, in a multi-agent system an agent usually does not have up-to-date knowledge about the value of variables controlled by other agents, because the time he last accessed them they might have been changed by these agents. Hence, it is natural to assign in the state s the "unknown" value **uu** to such nonlocal variables. This provides a natural motivation for developing a logic for concurrency featuring three logical values: **tt**, **ff** and **uu**—which we shall now proceed to do.

8.1. Syntax and semantics

The temporal logic L_{T} we are going to develop is a propositional logic based on partial order structures with the classical temporal operators (see e.g. [8]) $\Box$ (always), $\Diamond$ (eventually) and **U** (until) referring to the future. Since we do not assume existence of the next step operator $\circ$, the logic can be used for models based on continuous time (though it is also applicable to discrete time). The language of L_{T} is built over the set PROP of *propositional atoms* with help of the propositional connectives $\neg, \vee, \wedge$ and the above temporal operators. Thus the set $\mathrm{FORM}_{\mathrm{T}}$ of *formulae* is the least set FO such that:

1. $\mathrm{PROP} \subset FO$,
2. if $\alpha, \beta \in FO$, then $\neg\alpha, \alpha \vee \beta, \alpha \wedge \beta \in FO$,
3. if $\alpha \in FO$, then $\Diamond\alpha, \Box\alpha \in FO$,
4. if $\alpha, \beta \in FO$, then $\alpha \mathbf{U} \beta \in FO$.

The language is interpreted in models based on *temporal frames* of the form

$$F = \langle S, \leqslant \rangle,$$

where S is a nonempty set of *states*, and $\leqslant$ is a partial order on S constituting the *accessibility relation* of $\mathscr{F}$ (i.e., $\leqslant$ is reflexive, weakly antisymmetric and transitive). Denote by

$$B = \{\mathbf{tt}, \mathbf{ff}, \mathbf{uu}\},$$

the range of logical values we consider here. As the three-valued calculus underlying our temporal logic we choose the well-known Kleene's calculus. A model for L_{T} is a tuple

$$M_{\mathrm{T}} = \langle F, I \rangle,$$

where $I : S \times \text{PROP} \to B$ is the *interpretation of propositional atoms*. Then I is extended to the *interpretation* I^* *of all formulae* in FORM_T by taking, for any $s \in S$:

1. $I^*(s, p) = I(s, p)$ for $p \in \text{PROP}$,

2. $$I^*(s, \neg\alpha) = \begin{cases} \mathbf{tt} & \text{iff } I^*(s,\alpha) = \mathbf{ff}, \\ \mathbf{ff} & \text{iff } I^*(s,\alpha) = \mathbf{tt}, \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

3. $$I^*(s, \alpha \vee \beta) = \begin{cases} \mathbf{tt} & \text{iff either } I^*(s,\alpha) = \mathbf{tt}, \text{ or } I^*(s,\beta) = \mathbf{tt}, \\ \mathbf{ff} & \text{iff } I^*(s,\alpha) = I^*(s,\beta) = \mathbf{ff}, \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

4. $$I^*(s, \alpha \wedge \beta) = \begin{cases} \mathbf{tt} & \text{iff } I^*(s,\alpha) = I^*(s,\beta) = \mathbf{tt}, \\ \mathbf{ff} & \text{iff either } I^*(s,\alpha) = \mathbf{ff}, \text{ or } I^*(s,\beta) = \mathbf{ff}, \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

5. $$I^*(s, \Diamond\alpha) = \begin{cases} \mathbf{tt} & \text{iff } (\exists s' \in S)(s \leqslant s' \text{ and } I^*(s',\alpha) = \mathbf{tt}), \\ \mathbf{ff} & \text{iff } (\forall s' \in S)(s \leqslant s' \Rightarrow I^*(s',\alpha) = \mathbf{ff}), \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

6. $$I^*(s, \Box\alpha) = \begin{cases} \mathbf{tt} & \text{iff } (\forall s' \in S)(s \leqslant s' \Rightarrow I^*(s',\alpha) = \mathbf{tt}), \\ \mathbf{ff} & \text{iff } (\exists s' \in S)(s \leqslant s' \text{ and } I^*(s',\alpha) = \mathbf{ff}), \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

7. $$I^*(s, \alpha\mathbf{U}\beta) = \begin{cases} \mathbf{tt} & \text{iff } (\exists s' \in S)(s \leqslant s' \text{ and } I^*(s',\beta) = \mathbf{tt} \\ & \quad \text{and } (\forall s'' \in S)(s \leqslant s'' < s' \Rightarrow I^*(s'',\alpha) = \mathbf{tt})) \\ \mathbf{ff} & \text{iff } (\forall s' \in S)[(s \leqslant s' \text{ and } I^*(s',\beta) \neq \mathbf{ff}) \\ & \quad \Rightarrow (\exists s'' \in S)(s \leqslant s'' < s' \text{ and } I^*(s'',\alpha) = \mathbf{ff})] \\ \mathbf{uu} & \text{otherwise}, \end{cases}$$

where

$$s_1 < s_2 \stackrel{\text{df}}{\equiv} s_1 \leqslant s_2, \quad s_1 \neq s_2.$$

Note that in view of the reflexivity of $\leqslant$ we have $I^*(s, \alpha\mathbf{U}\beta) = \mathbf{tt}$ if $I^*(s,\beta) = \mathbf{tt}$. Moreover, the "until" operator we consider here is a strong one, since $I^*(s, \alpha\mathbf{U}\beta) = \mathbf{tt}$ only if there exists an $s' \in S$ with $s \leqslant s'$ and $I^*(s,\beta) = \mathbf{tt}$. Finally, since universal quantification over an empty set is trivially true, $I^*(s',\beta) = \mathbf{ff}$ for every s' with $s \leqslant s'$ implies $I^*(s, \alpha\mathbf{U}\beta) = \mathbf{ff}$.

Further, in view of the rather complicated character of the **U** operator there are several possible ways of defining the condition for $I^*(s,\alpha\mathbf{U}\beta)=\mathbf{ff}$. The clause we chose is advocated by its simplicity. Unfortunately, any attempt to separate **ff** from **uu** in this definition in a really precise way would yield a very complicated clause, resulting in an even more complicated deduction rule.

We assume a strong notion of satisfaction/validity in the sense that the only designated logical value is **tt**. Thus a formula α is

- *satisfied in a state s of a model M_T* ($M_T, s \models \alpha$) if $I^*(s,\alpha)=\mathbf{tt}$,
- *satisfied in a model M_T* ($M_T \models \alpha$) if $M_T, s \models \alpha$ for each state s of M_T,
- *valid* ($\models_T \alpha$) if $M_T \models \alpha$ for every model M.

8.2. R–S deduction system

To develop an R–S deduction system for our logic L_T, we have to extend the original language to a broader *deduction language* L_D, since in order to formulate R–S decomposition rules for temporal formulae we need—exactly as in case of tableau systems for such logics (see [27])—some means for representing the fact that a formula α holds in a state s, and that states s, s' are in the accessibility relation, i.e. that $s \leqslant s'$.

Thus let VARS be a countable set of *state variables*, and let $\prec$ be the constant representing the accessibility relation. We define the set of *state formulae* as

$$\text{FORM}_S = \{x.\alpha\colon\ \alpha \in \text{FORM}_T, x \in \text{VARS}\}.$$

Intuitively, $x.\alpha$ means that the temporal formula α is satisfied in the state x. The set of *accessibility formulae* expressing the accessibility relation on states is

$$\text{FORM}_A = \{x \prec y\colon\ x, y \in \text{VARS}\}.$$

Finally, the set of *formulae of the deduction language* L_D is defined by

$$\text{FORM}_D = \text{FORM}_S \cup \text{FORM}_A.$$

The underlying temporal frames are the same, but the notion of a model changes to

$$M_D = \langle F, I, v\rangle,$$

where, as before, $F=\langle S, \leqslant\rangle$ is a temporal frame, $I : S \times \text{PROP} \to B$ is the interpretation of propositional atoms, and, in addition, $v : \text{VARS} \to S$ is a *valuation of state variables.* The interpretation $I^+ : \text{FORM}_D \to B$ of the deduction language formulae is defined by

$$I^+(x.\alpha) = I^*(v(x),\alpha), \quad I^+(x \prec y) = \begin{cases} \mathbf{tt} & \text{if } v(x) \leqslant v(y), \\ \mathbf{ff} & \text{otherwise,} \end{cases}$$

where I^* is defined like in the preceding subsection.

A formula $\varphi \in \text{FORM}_D$ is said to be

- satisfied in a model M, in symbols $M \models \varphi$, iff $I^+(\varphi) = \mathbf{tt}$,
- valid, in symbols $\models \varphi$. if $M \models \varphi$ for each model M.

Fact 1. *A formula* $\alpha \in \text{FORM}_T$ *is valid if and only if the formula* $x.\alpha \in \text{FORM}_D$ *is valid, where* x *is an arbitrary variable in* VARS.

Evidently, this means that proving validity in L_T can be reduced to proving validity in L_D—which justifies the usefulness of our deduction language.

Since the logic we consider is a three-valued one, the deduction system is developed with help of signed formulae (see Section 3) belonging to the set

$$\text{SFORM} = \{\mathbf{N}\varphi\colon\ \varphi \in \text{FORM}_D\} \cup \{\mathbf{T}\varphi\colon\ \varphi \in \text{FORM}_D\}$$

and interpreted according to the rules

$$M_D \models \mathbf{T}(\varphi) \quad \text{iff } M_D \models \varphi, \quad M_D \models \mathbf{N}(\varphi) \quad \text{iff } M_D \not\models \varphi.$$

Thus our deduction system, denoted by DR_T, will operate on sequences Ω of signed formulae; recall that Ω is considered valid if, for any model M_D, some element of Ω is satisfied in M_D.

Indecomposable formulae are those having one of the following forms:

$$\mathbf{T}(x.p), \mathbf{N}(x.p), \mathbf{T}(x.\neg p), \mathbf{N}(x.\neg p), \mathbf{T}(x \prec y), \mathbf{N}(x \prec y) \tag{8}$$

where $x, y \in \text{VARS}$, $p \in \text{PROP}$.

Now we can present our R–S deduction system for three-valued temporal logic. In all the rules below, the Ω's are sequences of signed formulae, with Ω' being an indecomposable sequence. Furthermore, * always denotes repetition in the premises of the principal formula under decomposition from the conclusion, $\alpha, \beta \in \text{FORM}_T, x, y, z, v \in \text{VARS}$. Finally, in all rules save $(tr \prec)$ the variables y, z, w are mutually different, z, w do not appear above the double line, and y is an arbitrary variable.

8.2.1. DR_T*—an* R–S *deduction system for the three-valued temporal logic* L_T

Fundamental sequences: The *fundamental sequences* of DR_T are the sequences of formulae in SFORM that contain either formula (i) or pairs (ii), (iii) given below:

$$\textbf{(i)}\ \mathbf{T}(x \prec x), \quad \textbf{(ii)}\ \mathbf{N}(x.\alpha), \mathbf{N}(x.\neg\alpha), \quad \textbf{(iii)}\ \mathbf{T}(\varphi), \mathbf{N}(\varphi), \tag{9}$$

where $x \in \text{VARS}$, $\alpha \in \text{FORM}_T$, $\varphi \in \text{FORM}_D$.

Decomposition rules:

$$(\mathbf{T}\neg\neg)\ \frac{\Omega', \mathbf{T}(x.\neg\neg\alpha), \Omega''}{\Omega', (\mathbf{T}(x.\alpha), \Omega''} \qquad (\mathbf{N}\neg\neg)\ \frac{\Omega', \mathbf{N}(x.\neg\neg\alpha), \Omega''}{\Omega', \mathbf{N}(x.\alpha), \Omega''}$$

$$(\mathbf{T}\wedge)\ \frac{\Omega', \mathbf{T}(x.\alpha \wedge \beta), \Omega''}{\Omega', \mathbf{T}(x.\alpha), \Omega'' \mid \Omega', \mathbf{T}(x.\beta), \Omega''} \qquad (\mathbf{N}\wedge)\ \frac{\Omega', \mathbf{N}(x.\alpha \wedge \beta), \Omega''}{\Omega', \mathbf{N}(x.\alpha), \mathbf{N}(x.\beta), \Omega''}$$

$$(\mathbf{T}\neg\wedge)\ \frac{\Omega', \mathbf{T}(x.\neg(\alpha \wedge \beta)), \Omega''}{\Omega', \mathbf{T}(x.\neg\alpha), \mathbf{T}(x.\neg\beta), \Omega''} \qquad (\mathbf{N}\neg\wedge)\ \frac{\Omega', \mathbf{N}(x.\neg(\alpha \wedge \beta)), \Omega''}{\Omega', \mathbf{N}(x.\neg\alpha), \Omega'' \mid \Omega', \mathbf{N}(x.\neg\beta)\Omega''}$$

$$(\mathbf{T}\vee)\ \frac{\Omega', \mathbf{T}(x.\alpha \vee \beta), \Omega''}{\Omega', \mathbf{T}(x.\alpha), \mathbf{T}(x.\beta), \Omega''} \qquad (\mathbf{N}\vee)\ \frac{\Omega', \mathbf{N}(x.\alpha \vee \beta), \Omega''}{\Omega', \mathbf{N}(x.\alpha), \Omega'' \mid \Omega', \mathbf{N}(x.\beta), \Omega''}$$

$$(\mathbf{T}\neg\vee)\ \frac{\Omega', \mathbf{T}(x.\neg(\alpha \vee \beta)), \Omega''}{\Omega', \mathbf{T}(x.\neg\alpha), \Omega'' \mid \Omega', \mathbf{T}(x.\neg\beta), \Omega''} \qquad (\mathbf{N}\neg\vee)\ \frac{\Omega', \mathbf{N}(x.\neg(\alpha \vee \beta)), \Omega''}{\Omega', \mathbf{N}(x.\neg\alpha), \mathbf{N}(x.\neg\beta), \Omega''}$$

$$(\mathbf{T}\Box)\ \frac{\Omega', \mathbf{T}(x.\Box\alpha), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\alpha), \Omega''} \qquad (\mathbf{N}\Box)\ \frac{\Omega', \mathbf{N}(x.\Box\alpha), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{N}(y.\alpha), \Omega'', *}$$

$$(\mathbf{T}\neg\Box)\ \frac{\Omega', \mathbf{T}(x.\neg\Box\alpha), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{T}(y.\neg\alpha), \Omega'', *}$$

$$(\mathbf{N}\neg\Box)\ \frac{\Omega', \mathbf{N}(x.\neg\Box\alpha), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\neg\alpha), \Omega''}\ ß$$

$$(\mathbf{T}\Diamond)\ \frac{\Omega', \mathbf{T}(x.\Diamond\alpha), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{T}(y.\alpha), \Omega'', *} \qquad (\mathbf{N}\Diamond)\ \frac{\Omega', \mathbf{N}(x.\Diamond\alpha), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\alpha), \Omega''}$$

$$(\mathbf{T}\neg\Diamond)\ \frac{\Omega', \mathbf{T}(x.\neg\Diamond\alpha), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\neg\alpha), \Omega''}$$

$$(\mathbf{N}\neg\Diamond)\ \frac{\Omega', \mathbf{N}(x.\neg\Diamond\alpha), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{N}(y.\neg\alpha), \Omega'', *}$$

(**TU**)

$$\frac{\Omega', \mathbf{T}(x.\alpha\mathbf{U}\beta), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{T}(y.\beta), \Omega'', * \mid \Omega', \mathbf{N}(x \prec z), \mathbf{N}(z \prec y), \mathbf{T}(y \prec z), \mathbf{T}(z.\alpha), \Omega'', *}$$

(**NU**)

$$\frac{\Omega', \mathbf{N}(x.\alpha\mathbf{U}\beta), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\beta), \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\beta), \mathbf{T}(y \prec z), \Omega'', *}$$
$$\frac{}{\mid \Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\beta), \mathbf{N}(z \prec y), \Omega'', * \mid \Omega', \mathbf{N}(x \prec z), \mathbf{N}(z.\beta), \mathbf{N}(y.\alpha), \Omega'', *}$$

(**T**¬**U**)

$$\frac{\Omega', \mathbf{T}(x.\neg(\alpha\mathbf{U}\beta)), \Omega''}{\Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\neg\beta), \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\neg\beta), \mathbf{T}(y \prec z), \Omega'', *}$$
$$\frac{}{\mid \Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\neg\beta), \mathbf{N}(z \prec y), \Omega'', * \mid \Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\neg\beta), \mathbf{T}(y.\neg\alpha), \Omega'', *}$$

(**N**¬**U**)

$$\frac{\Omega', \mathbf{N}(x.\neg(\alpha\mathbf{U}\beta)), \Omega''}{\Omega', \mathbf{T}(x \prec y), \Omega'', * \mid \Omega', \mathbf{N}(y.\neg\beta), \Omega'', * \mid \Omega', \mathbf{N}(x \prec w), \mathbf{N}(w \prec y), \mathbf{T}(y \prec w), \mathbf{N}(w.\neg\alpha), \Omega'', *}$$

$$(tr \prec)\ \frac{\Omega', \mathbf{N}(x \prec y), \mathbf{N}(y \prec z), \Omega''}{\Omega', \mathbf{N}(x \prec y), \mathbf{N}(y \prec z), \mathbf{N}(x \prec z), \Omega''}$$

$$(as \prec)\ \frac{\Omega', \mathbf{N}(x \prec y), \mathbf{N}(y \prec x), \mathbf{N}(y.\pi), \Omega''}{\Omega', \mathbf{N}(x \prec y), \mathbf{N}(y \prec x), \mathbf{N}(y.\pi), \mathbf{N}(x.\pi), \Omega''}$$

where π is either p or $\neg p$ for some $p \in$ PROP.

By way of an explanation, let us first note that in Eq. (9), the fundamental sequence in (i) expresses reflexivity of the partial order relation, (ii) the basic fact that a formula and its negation cannot be both true, and (iii) is just an exact analogue of the condition on a fundamental sequence for classical logic.

As we see, the rules for connectives are three-valued analogues of the corresponding rules for classical logic—which is a consequence of the fact that Kleene's three-valued connectives have all the basic properties of the classical connectives except the excluded middle law. The rules for $\Box$ and $\Diamond$ deal with limited quantification inherent to these operators in a way quite analogous to the rules for $\underline{sim}$ and $\overline{sim}$ in similarity logic (see Section 2.2). They again exhibit the typical pattern of handling quantification in R–S systems discussed before. The same is true for the more involved rules concerning the operator **U**.

The reader can check himself that in order to get any of the modality rules it suffices to express the corresponding semantic clause as a quantified meta-formula, and than transform this formula into a decomposition rule, remembering that: universal quantification is replaced by substitution of a new variable for the quantified one in the quantified formula, existential quantification—by substituting any term for the quantified variable in the quantified formula and repeating the whole existential formula again, disjunction is represented by a comma in a sequence of formulae, and conjunction—by branching in a decomposition rule.

The only expansion rules (see Section 2) of our logic are $(tr \prec)$ and $(as \prec)$. Rule $(tr \prec)$ expresses transitivity of the accessibility relation, and rule $(as \prec)$—antisymmetry of the latter relation, whereby $x \prec y$ and $y \prec x$ implies that x, y denote the same state, so x can be substituted for y in any state formula (but the rule is limited to atomic formulae only in order to delay branching in the decomposition tree). Note that a corresponding substitution rule for accessibility formulae is not needed, since it would be just a special case of rule $(tr \prec)$.

Finally, let us note that notation of expansion rules is a somewhat simplified. In fact, the formulae in the conclusion can appear in an arbitrary order, and can be separated by arbitrary sequences of formulae; of course, both this order and contexts are preserved in the premise too. With such rules, their application order will be determined by a more detailed control mechanisms mentioned in Section 2.5; basically speaking, the first (instance of) rule to be applied is that applicable to the leftmost sequence of formulae.

Lemma 1. *The deduction system DR_{T} is sound.*

Proof. From reflexivity of the accessibility relation, the fact that $x.\alpha$ and $x.\neg\alpha$ cannot simultaneously take the value **tt** and from the properties of signed formulae it immediately follows that the fundamental sequences given above are indeed valid.

According to the remarks preceding the lemma, the rules for connectives are sound by the properties of Kleene's logic, and expansion rules—by the properties of the accessibility relation. By way of illustration, let us prove soundness of the basic rules for modal operators. The notation of models and their elements follows that introduced when defining the semantics.

Note that for greater clarity in case of a model $M_{\mathrm{D}} = \langle F, V, I\rangle$ for the deduction language L_{D} we use the semantic function I^* of the underlying model $M_{\mathrm{T}} = \langle F, I\rangle$ for L_{T} rather than the semantic function I^+ of M_{D} (recall that $I^+(x.\alpha) = I^*(v(x),\alpha)$). This facilitates reasoning in case of changing the valuation v for just one variable, since it makes v explicit, whereas it is only implicit in I^+.

Rule ($\mathbf{T}\Box$): We have to prove that the conclusion (the sequence over the double line) is valid if the premise (the sequence under the double line) is valid.

Let us start with downward implication. We argue by contradiction: assume that $\Omega', \mathbf{T}(x.\Box\alpha), \Omega''$ is valid, and suppose $M = \langle F, I, v\rangle$ is a model such that $M \not\models \Omega'$, $\mathbf{N}(x \prec z), \mathbf{T}(z.\alpha), \Omega''$. Then $M \not\models \Omega', \Omega''$, $v(x) \leqslant v(z)$ and $I^*(v(z),\alpha) \neq \mathbf{tt}$, whence $M \not\models \mathbf{T}(x.\Box\alpha)$ by the semantic clause for $\Box$—which is a contradiction. Thus the premise must be valid too.

Now let us pass to the upward implication, assuming that the premise is valid. Suppose $\Omega', \mathbf{T}(x.\Box\alpha), \Omega''$ is not valid, and let a model $M = \langle F, I, v\rangle$ be such that $M \not\models \Omega'$, $\mathbf{T}(x.\Box\alpha), \Omega''$. Then there exists an $s \in S$ such that $v(x) \leqslant s$ and $I^*(s,\alpha) \neq \mathbf{tt}$. Now let $M' = \langle F, I, v'\rangle$, where $v' = v[s/z]$ (see Section 6 for the definition). Since z is a new variable, then $M \not\models \Omega', \Omega''$ implies $M' \not\models \Omega', \Omega''$. Moreover, as $x \neq z$, then we have $v'(x) = v(x) \leqslant s = v'(z)$ and $I^*(v(z),\alpha) = I^*(s,\alpha) \neq \mathbf{tt}$. Thus $M \not\models \Omega', \mathbf{N}(x \prec z), \mathbf{T}(z.\alpha), \Omega''$, which contradicts our assumption on the validity of the premise. Hence the conclusions has to be valid too, and the rule is sound.

Rule ($\mathbf{T}\Diamond$): Now we have to prove that the conclusion is valid iff both the premises are valid.

The downward implication is trivial, since the conclusion is contained in the premises (where $\mathbf{T}(x.\Diamond\alpha)$ is marked by *). To prove the upward implication, assume both $\Omega', \mathbf{T}(x \prec y), \Omega'', *$ and $\Omega', \mathbf{T}(y.\alpha), \Omega'', *$ are valid. Let $M = \langle F, I, v\rangle$ be an arbitrary model, and suppose $M \not\models \Omega', \mathbf{T}(x.\Diamond\alpha), \Omega''$. Then from the validity of both the premises it follows that $M \models \mathbf{T}(x \prec y)$ and $M \models \mathbf{T}(y.\alpha)$. Thus $v(x) \leqslant v(y)$ and $I^*(v(y),\alpha) = \mathbf{tt}$, whence $M \models \mathbf{T}(x.\Diamond\alpha)$—which is a contradiction. In consequence, the conclusion has to be valid, and the rule is sound.

Since the "until" ($\mathbf{U}$) modality is especially complicated, in its case we will show validity of two rules: the standard ($\mathbf{T}\neg\mathbf{U}$) rule, and the negation rule ($\mathbf{TU}$). Remember that * in the premises of the rule denotes $\mathbf{TU}$.

Rule the conclusion: The downward implication is again trivial. To prove the upward implication, assume that all the premises are valid, and suppose that $\Omega', \mathbf{T}(x.\alpha\mathbf{U}\beta), \Omega''$ is not valid, i.e. $M \not\models \Omega', \mathbf{T}(x.\alpha\mathbf{U}\beta), \Omega''$ for some model $M = \langle F, I, v\rangle$.

Since the premises are valid and $M \not\models \Omega', \mathbf{T}(x.\alpha\mathbf{U}\beta), \Omega''$, we have in particular $M \models \mathbf{T}(x \prec y), M \models \mathbf{T}(y.\beta)$, whence $v(x) \leqslant v(y)$ and $I^*(v(y), \beta) = \mathbf{tt}$. As $M \not\models \mathbf{T}(x.\alpha\mathbf{U}\beta)$, then by the semantic clause for $\mathbf{U}$ there must be an $s' \in S$ with $v(x) \leqslant s < v(y)$ and $I^*(s, \alpha) \neq \mathbf{tt}$. Now let $M' = \langle F, I, v' \rangle$, where $v' = v[s/z]$. Then, in view of $z \neq x, y$, we have $v'(x) = v(x), v'(y) = v(y)$ and $v'(z) = s$, whence $v'(x) \leqslant v'(z) < v'(y)$ and $I^*(v'(z), \alpha) \neq \mathbf{tt}$. In consequence, $M' \models \mathbf{T}(x \prec z)$, $M' \models \mathbf{T}(z \prec y)$, $M' \models \mathbf{N}(y \prec z)$ and $M' \models \mathbf{N}(z.\alpha)$. Furthermore, as $M \not\models \Omega'$, $\mathbf{T}(x.\alpha\mathbf{U}\beta), \Omega''$ and z is new, then the same is true for M'. Hence $M' \not\models \Omega'$, $\mathbf{N}(x \prec z)$, $\mathbf{N}(z \prec y)$, $\mathbf{T}(y \prec z)$, $\mathbf{T}(z.\alpha), \Omega''$, $\mathbf{T}(x.\alpha\mathbf{U}\beta)$, which contradicts validity of the third premise. Thus the upward implication must hold too.

Rule ($\mathbf{T}\neg\mathbf{U}$): As before, the downward implication is trivial. To prove the upward implication, assume that the premises are valid, and suppose the conclusion is not, i.e. $M \not\models \Omega'$, $\mathbf{T}(x.\neg(\alpha\mathbf{U}\beta)), \Omega''$ for some model $M = \langle F, I, v \rangle$. Then there is an $s \in S$ such that $v(x) \leqslant s, I^*(s, \beta) \neq \mathbf{ff}$, and there does not exist an $s' \in S$ with $v(x) \leqslant s' < s$ and $I^*(s', \alpha) = \mathbf{ff}$.

Let $M' = \langle F, I, v' \rangle$, where $v' = v[s/z]$. As z is new, then $M' \not\models \Omega', \mathbf{T}(x.\neg(\alpha\mathbf{U}\beta)), \Omega''$. Moreover, $M' \models \mathbf{T}(x \prec z)$ and $M' \models \mathbf{N}(z.\neg\beta)$. Since the premises are valid, this implies $M' \models \mathbf{T}(x \prec y)$, $M' \models \mathbf{T}(y \prec z)$, $M' \models \mathbf{N}(z \prec y)$, $M' \models \mathbf{T}(y.\neg\alpha)$. Thus $v'(x) \leqslant v'(y)$, $v'(y) < v(z)$ and $I^*(v'(y), \alpha) = \mathbf{ff}$. Considering the definition of v', in view of $z \neq y$ we get $v(x) \leqslant v(y) < s$ and $I^*(v(y), \alpha) = \mathbf{ff}$, which is a contradiction. Thus the upward implication must also hold. □

Now it remains to prove completeness. Let us define the notions of a decomposition tree, proof and provability as in Section 2. Then we have the following completeness result:

Theorem 3. *Each valid sequence Ω of formulae in* $\textsc{Sform}_{\mathrm{D}}$ *is provable in* DR_{T}.

Proof. The proof follows the universal pattern outlined in Section 2: we assume that Ω is not provable, and find a counter-model for it.

Let us begin with its most crucial part—construction of a counter-model M_{C} for a set Δ of indecomposable formulae which does not contain any fundamental sequence and is closed under the expansion rules $(tr \prec), (as \prec)$. Let $\sim$ be a relation on $\textsc{Vars}$ defined by

$$x \sim y \text{ iff either } x = y \text{ or } \mathbf{N}(x \prec y) \in \Delta \text{ and } \mathbf{N}(y \prec x) \in \Delta.$$

Then $\sim$ is reflexive and symmetric. Moreover, it is also transitive in view of the fact that Δ is closed under rule $(tr \prec)$. Indeed: if $x \sim y$ and $y \sim z$, then either $x = y$ or $y = z$, whence obviously $x \sim z$, or else (1) $\mathbf{N}(x \prec y)$, (2) $\mathbf{N}(y \prec x)$, (3) $\mathbf{N}(y \prec z)$, (4) $\mathbf{N}(z \prec y) \in \Delta$, whence by applying rule $(tr \prec)$ first to (1), (3) and then to (4), (2) we get $\mathbf{N}(x \prec z), \mathbf{N}(z \prec x) \in \Delta$. Thus $\sim$ is an equivalence relation. As the set of states of our counter-model we take

$$S_{\mathrm{C}} = \textsc{Vars}/\sim = \{[x]_\sim : x \in \textsc{Vars}\}.$$

For simplicity, we shall write $[x]$ instead of $[x]_\sim$. As the temporal frame underlying the counter-model we take $F_C = \langle S_C, \leqslant \rangle$, where the accessibility relation $\leqslant$ is defined by

$$[x] \leqslant [y] \text{ iff either } [x] = [y] \text{ or } \mathbf{N}(x \prec y) \in \Delta.$$

It is easy to see that $\leqslant$ is well-defined. Indeed, reasoning in the way exactly analogous to the proof of transitivity of $\sim$ given above, we can show that, for any x, x', y, y' such that $x \neq x', y \neq y'$ and $x \sim x', y \sim y'$, we have $\mathbf{N}(x \prec y) \in \Delta$ iff $\mathbf{N}(x' \prec y') \in \Delta$, whence the definition of $[x] \leqslant [y]$ is independent of the choice of the representatives of classes $[x], [y]$. Obviously, $\leqslant$ is reflexive. It is also weakly antisymmetric, since $[x] \leqslant [y]$ and $[y] \leqslant [x]$ implies either $[x] = [y]$ or $\mathbf{N}(x \prec y), \mathbf{N}(y \prec x) \in \Delta$, whence $x \sim y$ and again $[x] = [y]$. Finally, it is also transitive. Indeed, if $[x] \leqslant [y]$ and $[y] \leqslant [z]$, then either $[x] = [y]$ or $[y] = [z]$, whence obviously $[x] \leqslant [z]$, or else $\mathbf{N}(x \prec y), \mathbf{N}(y \prec z) \in \Delta$. As Δ is closed under rule $(tr \prec)$, this yields $\mathbf{N}(x \prec z) \in \Delta$, whence $[x] \leqslant [z]$. Thus $\leqslant$ is a partial order and F_C is a correct temporal frame.

We put $M_C = \langle F, I, v \rangle$, where

$$v(x) = [x] \quad \text{for } x \in \text{VARS}, \quad I([x], p) = \begin{cases} \mathbf{tt} & \text{iff } \mathbf{N}(x.p) \in \Delta, \\ \mathbf{ff} & \text{iff } \mathbf{N}(x.\neg p) \in \Delta, \\ \mathbf{uu} & \text{otherwise.} \end{cases}$$

Again it is easy to see that I is correctly defined. Indeed if $x \neq x'$, $x \sim x'$ and $\mathbf{N}(x'.\pi) \in \Delta$, where π is either p or $\neg p$ for some $p \in \text{PROP}$, then $\mathbf{N}(x \prec x')$, $\mathbf{N}(x' \prec x)$, $\mathbf{N}(x'.\pi) \in \Delta$. As Δ is closed under rule $(as \prec)$, this implies $\mathbf{N}(x.\pi) \in \Delta$, so the definition of $I([x], p)$ does not depend on the choice of the representative of the class $[x]$. Moreover, as Δ does not contain any fundamental sequence, then by (9) the formulae $\mathbf{N}(x.p)$ and $\mathbf{N}(x.\neg p)$ cannot both belong to Δ, whence there is no clash between the **tt** and **ff** clauses of the definition.

It remains to check that M_C is indeed a counter-model for Δ. Our considerations will be based on the definitions of the interpretation I^+ on the basis of I and of satisfaction of signed formulae given in the preceding section. Let φ be an arbitrary formula in Δ. Then φ is indecomposable, so it must have one of the forms given in (8):

$$\mathbf{T}(x.p), \mathbf{N}(x.p), \mathbf{T}(x.\neg p), \mathbf{N}(x.\neg p), \mathbf{T}(x \prec y), \mathbf{N}(x \prec y).$$

Since we have shown above that neither the value of $I([x], p)$ nor satisfaction of the relation $[x] \leqslant [y]$ depend on the choice of the representatives of classes $[x], [y]$, in all the cases below we can assume the representative of a class $[z]$ is just z. In other words, in order to evaluate $I^+(x.\alpha)$ or $I^+(x \prec y)$ we do not have to consider whether Δ contains $\mathbf{N}(x'.\alpha)$ or $\mathbf{N}(x' \prec y')$ for some $x' \sim x, y' \sim y$—but only whether it contains $\mathbf{N}(x.\alpha)$ or $\mathbf{N}(x \prec y)$.

Case 1: $\varphi = \mathbf{T}(x.p)$. Then $\mathbf{N}(x.p) \notin \Delta$, since otherwise Δ would contain a fundamental sequence (see (9)). Hence $I^+(x.p) = I([x], p) = \mathbf{ff}$ if $\mathbf{N}(x.\neg p) \in \Delta$, and otherwise $I^+(x.p) = \mathbf{uu}$. In both cases $M_C \not\models \mathbf{T}(x.p)$.

Case 2: $\varphi = \mathbf{N}(x.p)$. Then $I^+(x.p) = \mathbf{tt}$ and again $M_C \not\models \mathbf{N}(x.p)$.

Case 3: $\varphi = \mathbf{T}(x.\neg p)$. Then $\mathbf{N}(x.\neg p) \notin \Delta$, since otherwise Δ would contain a fundamental sequence. Hence $I([x], p) = \mathbf{tt}$ if $\mathbf{N}(x.p) \in \Delta$, and otherwise $I([x], p) = \mathbf{uu}$, whence $I^+(x.\neg p) \in \{\mathbf{ff}, \mathbf{uu}\}$. In both cases, $M_C \not\models \mathbf{T}(x.\neg p)$.

Case 4: $\varphi = \mathbf{N}(x.\neg p)$. Then $I([x], p) = \mathbf{ff}$, whence $I^+(x.\neg p) = \mathbf{tt}$ and $M_C \not\models \mathbf{N}(x.\neg p)$.

Case 5: $\varphi = \mathbf{N}(x \prec y)$. Then $[x] \leqslant [y]$, whence $I^+(x \prec y) = \mathbf{tt}$ and $M_C \not\models \mathbf{N}(x \prec y)$.

Case 6: $\varphi = \mathbf{T}(x \prec y)$. Then neither $x = y$ nor $\mathbf{N}(x \prec y) \notin \Delta$, for otherwise Δ would contain a fundamental sequence. Thus non $[x] \leqslant [y]$ and $I^+(x \prec y) = \mathbf{ff}$, whence $M_C \not\models \mathbf{T}(x \prec y)$.

Thus M_C is indeed a counter-model for Δ.

Now let us present in more detail the inductive part of the theorem outlined briefly in Section 2. If a sequence Ω is not provable, then its decomposition tree $DT(\Omega)$ is not a proof. Hence either the tree is finite and contains a leaf l labelled by a sequence of indecomposable formulae which is not fundamental, or it has an infinite branch B by Koenig's lemma. Let us denote by Δ the set of formulae in the label of leaf l in the first case, and the set of all indecomposable formulae on the branch B in the second case. Then Δ is closed under expansion rules and does not contain a fundamental sequence, whence by what we have proved above it has a counter-model M_C.

We shall prove now that Ω is not valid. If the tree is finite, then the thesis follows immediately from invertibility of the rules and the fact that the sequence labelling the leaf l of the tree is not valid. Now consider the case when the tree has an infinite branch B. We shall show that M_C—the counter-model for the set of all indecomposable formulae of B—is a counter-model for Ω. Let us define the rank r of a formula in FORM_T by

1. $r(p) = r(\neg p) = 0$ for $p \in \text{PROP}$;
2. $r(\neg\alpha) = 1 + r(\alpha)$ for $\alpha \notin \text{PROP}$;
3. $r(D\alpha) = 1 + r(\alpha)$ for $D \in \{\Diamond, \Box\}$;
4. $r(\alpha C \beta) = 1 + \max(r(\alpha), r(\beta))$ for $C \in \{\vee, \wedge, \mathbf{U}\}$.

Further, define the rank R of formulae in SFORM_D as

1. $R(\mathbf{T}(x \prec y)) = R(\mathbf{N}(x \prec y) = 0$;
2. $R(\mathbf{T}(x.\alpha)) = R(\mathbf{N}(x.\alpha)) = r(\alpha)$.

We shall prove that $M_C \not\models \Omega$ by induction on the rank of signed formulae on the branch B. We know that $M_C \not\models \varphi$ for any signed formula φ on B with $R(\varphi) = 0$, since M_C is a counter-model for all indecomposable formulae on B—and indecomposable formulae are exactly formulae of rank 0.

Suppose some formulae on B are satisfied in M_C, and let φ be such a formula with the least rank. Then $R(\varphi) > 0$, so φ is of the form either $\mathbf{T}(x.\alpha)$ or $\mathbf{T}(x.\alpha)$. Now we shall argue on the structure of α to show that our assumption leads to a contradiction. We shall make use of the fact that if the decomposed formula contained in the conclusion of some decomposition rule is on branch B, then one of the sequence of formulae resulting from its decomposition and contained in the premises of this rule must be on branch B too.

1. α cannot be of the form $\neg\neg\beta, \beta_1 \vee \beta_2, \beta_1 \wedge \beta_2, \neg(\beta_1 \vee \beta_2), \neg(\beta_1 \wedge \beta_2)$. Indeed if e.g. $\varphi = \mathbf{T}(x.\neg\neg\beta)$, then $\mathbf{T}(M_C \models \mathbf{T}(x.\neg\neg\beta)$ would imply $M_C \models \mathbf{T}(x.\beta)$, which is a

contradiction since $\mathbf{T}(x.\beta)$ would also be on B by rule $(\mathbf{T}\neg\neg)$ and $R(\mathbf{T}(x.\beta)) < R(\varphi)$. The situation with $\mathbf{N}$, as well as with other combinations of connectives corresponding to single-premise rules is analogous. In case of double-premise rules, branch B would have to contain one of the premises—and hence again a formula satisfied in M_C with a rank lower than that of φ (e.g., if $\varphi = \mathbf{T}(x.\alpha \wedge \beta)$, then B must contain either $\mathbf{T}(x.\alpha)$ or $\mathbf{T}(x.\beta)$—both satisfied in M_C.

2. α cannot be of the form $\Diamond\beta$, $\neg\Diamond\beta$. Suppose first that $\varphi = \mathbf{T}(x.\Diamond\beta)$. Then $M_C \models \varphi$ implies that there exists an $y \in \text{VARS}$ such that $[x] \leqslant [y]$ and $I^*([y], \beta) = \mathbf{tt}$. Hence for such an y we would have $M_C \models \mathbf{T}(x \prec y)$ and $M_C \models \mathbf{T}(y.\beta)$. By rule $(\mathbf{T}\Diamond)$, one of them has to be on B. Since both have smaller ranks than φ, this is a contradiction.

   Now suppose that $\varphi = \mathbf{N}(x.\Diamond\beta)$. Then for every $z \in \text{VARS}$ such that $[x] \leqslant [z]$ we have $I^*([z], \beta) \neq \mathbf{tt}$ in M_C. In consequence, $M_C \models \mathbf{N}(x \prec z), \mathbf{N}(z.\beta)$. By rule $(\mathbf{N}\Diamond)$, both these formulae have to be on branch B. Since one of them must be satisfied in M_C and both have smaller ranks than φ, we get a contradiction again.

   The proof for each of the remaining cases is analogous to one of the above.

3. α cannot be of the form $\Box\beta, \neg\Box\beta$. The proofs are analogous to the preceding case by duality between $\Box$ and $\Diamond$.

4. α cannot be of the form $\beta_1 \mathbf{U} \beta, \neg(\beta_1 \mathbf{U} \beta_2)$. Suppose $\varphi = \mathbf{T}(x.\beta_1 \mathbf{U} \beta_2)$. Then $M_C \models \varphi$ implies that there exists $y \in \text{VARSE}$ such that $[x] \leqslant [y], I^*([y], \beta_2) = \mathbf{tt}$, and for every $z \in \text{VARSE}$ with $[x] \leqslant [z] < [y]$ we have $I^*([z], \beta_1) = \mathbf{tt}$. In consequence, $M_C \models \mathbf{T}(x \prec y)$, $M_C \models \mathbf{T}(y.\beta_2)$, and $M_C \models \mathbf{N}(x \prec z), \mathbf{N}(z \prec y), \mathbf{T}(y \prec z), \mathbf{T}(z.\alpha)$. Moreover, by rule $(\mathbf{TU})$ one of these three sequences of formulae have to be on B, and all of the formulae they include have ranks smaller than φ, Thus we arrive at a contradiction again.

The proofs for the other cases are quite analogous.

Since all the possible forms of φ have been excluded, no formula on B can hold on M_C—whence Ω, which is on B as the label of the root of the tree, is not satisfied in M_C, and hence not valid. □

From the completeness theorem, Fact 1 and the definition of a proof in an R–S system given in Section 2 it follows that in order to prove the validity of a temporal formula $\alpha \in \text{FORM}_T$ we have to construct a decomposition tree of the formula $\mathbf{T}(x.\alpha)$, when x is any variable in VARS. If the tree is finite and its leaves are labeled with fundamental sequences only, then α is valid—otherwise it is not.

It should be noted that the signed formula operators $\mathbf{T}, \mathbf{N}$ introduced on top of the formulae in FORM_D can be eliminated by passing from the DR_T system to an equivalent Gentzen sequent calculus in the way described in the next section.

9. Transformation of an R–S system into a Gentzen calculus system

As we have already said, the basic goal—and result—of applying the R–S methodology is a complete R–S system for a given logic. However, in order to propagate or

use this result in the more extensive environment of Gentzen sequent calculus users, we can transform the latter system almost automatically into a complete Gentzen sequent calculus. The interest of such a transformation is twofold. First, it allows us to pass from a complete deduction system for a richer language of signed formulae to an equally complete deduction system for the original language in a quite automatic way. Second, the Gentzen formalism is much more widely known than the Rasiowa–Sikorski one, so such a translation could generate a more widespread interest in, or use of the deduction system obtained primarily using Rasiowa–Sikorski method.

In this final section we shall present a method for transforming an R–S calculus for signed formulae over some logic into a Gentzen sequent calculus for the original logic. The method used is similar to that presented in [10] in case of Beth's tableaux and follows from a close connection between validity of sequents and validity of sequences of signed formulae. We omit the more detailed proofs, which are simple generalizations of those given, e.g. in [2, 13] for systems with rules having up to two premises only (here we allow an arbitrary finite number of premises).

The presentation of transformation given here is tailored to an R–S system using signed formulae, to show how we can get rid of this temporary extension to the language by passing to sequent calculus. The above method applies of course (with minor adaptations) also to an R–S system for a logic with "standard" negation, which does not use signed formulae. However, this involves certain additional technical details—which was another reason for our choice of presentation.

Let us start with introducing the basic notions and notation. The notation not defined below is taken from Sections 1–3.

Definition 5. By a *sequent* we mean a pair (Γ, Δ) of finite sets Γ and Δ of formulae in $\mathscr{F}$, written in the form $\Gamma \vdash \Delta$.

A sequent $\Gamma \vdash \Delta$ is said to be *valid* iff, for any model M satisfying all the formulae in Γ, at least one formula in Δ is satisfied in M.

Assume that there is some "vocabulary order" on formulae, and that notation of the type $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ used for finite sets of formulae represents just this order. For the above set Γ, let $\mathbf{T}(\Gamma), \mathbf{N}(\Gamma)$ be sequences of signed formulae defined by

$$\mathbf{T}(\Gamma) = \mathbf{T}(\alpha_1), \mathbf{T}(\alpha_2), \ldots, \mathbf{T}(\alpha_n), \quad \mathbf{N}(\Gamma) = \mathbf{N}(\alpha_1), \mathbf{N}(\alpha_2), \ldots, \mathbf{N}(\alpha_n).$$

Then the simple connection between validity of sequents and validity of signed formulae is as follows:

Lemma 2. *A sequent $\Gamma \vdash \Delta$ is valid whenever the sequence $\mathbf{N}(\Gamma), \mathbf{T}(\Delta)$ of signed formulae is valid.*

Proof. Recall that a sequence of signed formulae is valid iff, for any model M, some formula in this sequence is satisfied in M. By definition, a sequent $\Gamma \vdash \Delta$ is valid iff, for every model M such that every $\alpha \in \Gamma$ is satisfied in M, some $\beta \in \Delta$ is satisfied in M. In other words, $\Gamma \vdash \Delta$ is valid iff, for every model M, either some $\alpha \in \Gamma$ is not

satisfied in M or some $\beta \in \Delta$ is satisfied in M—or, equivalently, for every M, either $\mathbf{N}(\alpha)$ is satisfied in M for some $\alpha \in \Gamma$ or $\mathbf{T}(\beta)$ is satisfied in M for some $\beta \in \Delta$. The latter condition is obviously tantamount to validity of the sequence $\mathbf{N}(\Gamma), \mathbf{T}(\Delta)$. □

Assume now that we already have a complete R–S deduction system DRS for the language of signed formulae over our original language. Then from the completeness theorem for DRS (see Section 2) it follows that

Fact 2. *A sequent $\Gamma \vdash \Delta$ is valid whenever the sequence $\mathbf{N}(\Gamma), \mathbf{T}(\Delta)$ of signed formulae has a proof in the R–S system, i.e. whenever it has a finite decomposition tree with all terminal sequences fundamental.*

The above fact suggests a simple method of developing a sequent calculus for our original formal language out of the system DRS for sequences of signed formulae; it suffices to derive from the fundamental sequences and decomposition rules of DRS the corresponding axioms and inference rules of the sequent calculus, basing on the above Fact.

The sequent calculus SC we have in mind will consist of *axioms*, having the form of single (valid) sequents, and *inference rules*, leading from valid sequents to valid sequents. An inference rule is of the form

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \Gamma_2 \vdash \Delta_2 \ldots \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}.$$

The sequent $\Gamma \vdash \Delta$ will be called *the conclusion* of the rule, and the sequent(s) $\Gamma_1 \vdash \Delta_1, \Gamma_2 \vdash \Delta_2, \ldots \Gamma_n \vdash \Delta_n$—its *premises*. A rule is said to be *sound* iff its conclusion is valid whenever all its premises are valid.

Thus, we should bear in mind that there are two basic differences between decomposition rules for signed formulae and inference rules of the sequent calculus:

- decomposition rules for signed formulae are two-way rules saying that the conclusion is valid iff all the premises are valid, whereas sequent calculus rules are, in principle, one-way rules saying that the conclusion is valid whenever all the premises are valid;
- in proof construction, decomposition rules were used "top down" to construct a decomposition tree of a sequence of formulae, whereas sequent calculus rules will be used "bottom up" to deduce a sequent from the axioms.

For any sequence Ω of signed formulae, let us denote

$$\Omega^+ = \{\alpha \in \mathscr{F}\colon\ \mathbf{T}(\alpha) \in \Omega\}, \qquad \Omega^- = \{\alpha \in \mathscr{F}\colon\ \mathbf{N}(\alpha) \in \Omega\}.$$

Then our basic Lemma 1 can be rephrased as follows: for any finite sequence Ω of signed formulae, the sequent $\Omega^- \vdash \Omega^+$ is valid iff Ω has a proof in the system DRS.

This immediately gives rise to:

Fact 3. *If Ω is a fundamental sequence in DRS, then $\Omega^- \vdash \Omega^+$ is a sound axiom of the sequent calculus SC.*

Now suppose that

$$\frac{\Omega', \Sigma, \Omega''}{\Omega', \Sigma_1, \Omega'' \mid \Omega', \Sigma_2, \Omega'' \mid \ldots \mid \Omega' \Sigma_n, \Omega'',}$$

is a decomposition rule in DRS. Then, evidently, for any finite sets $\Gamma, \Delta \subset \mathscr{F}$, the sequence $\mathbf{N}(\Gamma), \Sigma, \mathbf{T}(\Delta)$ has a proof in DRS whenever all the sequences $\mathbf{N}(\Gamma), \Sigma_i, \mathbf{T}(\Delta)$, $i = 1, 2, \ldots, n$, do. In consequence, considering Lemma 1, the sequent $\Gamma, \Sigma^- \vdash \Delta, \Sigma^+$ is valid whenever all the sequents $\Gamma, \Sigma_i^- \vdash \Delta, \Sigma_i^+$, $i = 1, 2, \ldots, n$, are valid. Thus we have:

Lemma 3. *For every rule*

$$\frac{\Omega', \Sigma, \Omega''}{\Omega', \Sigma_1, \Omega'' \mid \Omega', \Sigma_2, \Omega'' \mid \ldots \mid \Omega', \Sigma_n, \Omega'',}$$

in the R–S deduction system DRS,

$$\frac{\Gamma, \Sigma_1^- \vdash \Delta, \Sigma_1^+ \quad \Gamma, \Sigma_2^- \vdash \Delta, \Sigma_2^+ \quad \ldots \quad \Gamma, \Sigma_n^- \vdash \Delta, \Sigma_n^+}{\Gamma, \Sigma^- \vdash \Delta, \Sigma^+}$$

is a sound inference rule of the sequent calculus SC.

The above lemma is exemplified by the following table which gives sequent calculus rules induced by some typical schemata of decomposition rules encountered in R–S systems.

Decomposition rule	Inference rule for sequents
$\dfrac{\Omega', \mathbf{T}(\alpha), \Omega''}{\Omega', \mathbf{T}(\alpha_1), \Omega''}$	$\dfrac{\Gamma \vdash \Delta, \alpha_1}{\Gamma \vdash \Delta, \alpha}$
$\dfrac{\Omega', \mathbf{N}(\alpha), \Omega''}{\Omega', \mathbf{N}(\alpha_1), \Omega''}$	$\dfrac{\Gamma, \alpha_1 \vdash \Delta}{\Gamma, \alpha \vdash \Delta}$
$\dfrac{\Omega', \mathbf{T}(\alpha), \Omega''}{\Omega', \mathbf{T}(\alpha_1), \mathbf{T}(\alpha_2), \Omega''}$	$\dfrac{\Gamma \vdash \Delta, \alpha_1, \alpha_2}{\Gamma \vdash \Delta, \alpha}$
$\dfrac{\Omega', \mathbf{N}(\alpha), \Omega''}{\Omega', \mathbf{N}(\alpha_1), \mathbf{N}(\alpha_2), \Omega''}$	$\dfrac{\Gamma, \alpha_1, \alpha_2 \vdash \Delta}{\Gamma, \alpha \vdash \Delta}$
$\dfrac{\Omega', \mathbf{T}(\alpha_1), \Omega''}{\Omega', \mathbf{T}(\alpha_1), \Omega'' \mid \Omega', \mathbf{T}(\alpha_2), \Omega''}$	$\dfrac{\Gamma \vdash \Delta, \alpha_1, \quad \Gamma \vdash \Delta, \alpha_2}{\Gamma \vdash \Delta, \alpha}$
$\dfrac{\Omega', \mathbf{N}(\alpha), \Omega''}{\Omega', \mathbf{N}(\alpha_1), \Omega'' \mid \Omega', \mathbf{N}(\alpha_2), \Omega''}$	$\dfrac{\Gamma, \alpha_1 \vdash \Delta \quad \Gamma, \alpha_2 \vdash \Delta}{\Gamma, \alpha \vdash \Delta}$

(Here all the α's and α_i's are 'ordinary' formulae in $\mathscr{F}$.)

There is a general schema of proving that the sequent calculus SC resulting from a sound complete R–S deduction system is sound and complete:

Theorem 4 (Soundness and completeness). *Any sequent is provable in SC iff it is valid.*

Proof. Soundness follows from Fact 2 and Lemma 2. To prove completeness, assume a sequent $\Gamma \vdash \Delta$ is valid. Then the sequence $\mathbf{N}(\Gamma), \mathbf{T}(\Delta)$ is also valid, so by completeness of DRS it has a finite decomposition tree T with fundamental sequences at all its leaves. Out of that tree, we can easily construct a proof of the original sequent in the SC calculus. Namely, we prove by induction that the sequent $\Omega^- \vdash \Omega^+$ corresponding to any sequence Ω of formulas labelling a vertex of T is provable. Indeed, the leaves are labelled by axiomatic sequences, so the corresponding sequents are axioms of SC. Going upwards in T, we can replace each downward application of an R–S rule by an upward application of the corresponding sequent calculus rule. We finish our induction at the root, which is labelled by the sequence $\mathbf{N}(\Gamma), \mathbf{T}(\Delta)$ corresponding to the original sequent. □.

It should be noted that the sequent calculus SC obtained in this way is usually extended by adding the weakening rules which allow us to simplify some special rules of SC, e.g. those dealing with existential quantification, by eliminating the repetition of the formula from the conclusion in the premises.

Finally, a natural question arises whether a converse translation—from a Gentzen calculus to an R–S system—is possible and/or desirable. This seems doubtful. First, no straightforward translation is possible, because R–S rules are invertible, whereas in general sequent calculus ones are not. Second, it is not quite clear whether such a translation is really advisable. The best reason would be proof search/generation, to which an R–S system seems better adapted, but for this purpose a translation to a tableau system, the rules of which need not be invertible, seems more sensible.

10. Conclusions

The examples presented in the paper prove that the R–S methodology of developing a complete deduction system for a logic through analysing its semantics is indeed a powerful and universal tool that allows us to develop complete deduction systems for a broad class of logics encountered in CS applications in an intuitively clear way, according to a uniform procedure.

As we have seen, the prerequisite for applying the methodology to a logic is sufficient expressiveness of its language, needed for representing the semantics of its operators through syntactic means—because this is what we need to build the rules of an R–S deduction system. If a given language is not expressive enough, we can sometimes remedy that by introducing certain auxiliary means, like signed formulae,

nominals, etc. Sometimes a good method is to either give the logic a relational semantics, or translate it to relational logic. Indeed, through the use of variables representing components of binary relation couples, we can easily model all standard operations on relations, as well as many nonstandard ones, by decomposition rules, and thus obtain a complete R–S system for relational logics. In such a way, many logics, like e.g. modal and substructural ones, have been formalised in an indirect method using the R–S methodology (see especially the work of Orłowska, McCaull and others quoted below).

Moreover, as we have argued e.g. in Section 2.5, Rasiowa–Sikorski systems can lend themselves very well to automation—at least as well as tableau systems. An indirect proof of this fact is the duality of R–S systems to the tableau systems and the identity of the general principle underlying both the formalism: direct modelling of the semantics through proof rules. A clear proof are the properties of an R–S system itself. After a certain refinement discussed in Section 2.5, the inductive definition of a decomposition tree can be transformed into a ready-made algorithm for generating a decomposition tree of a formula, which in case of it being finite and having leaves labelled by fundamental sequences only constitutes the proof of that formula. Of course, a practical use of such an algorithm would require further modifications to make it more effective, maybe analogous to those performed in case of tableau systems.

On the other hand, the advantage of R–S systems over tableaux is invertibility of their decomposition rules, which makes them a more powerful and precise tool from proof-theoretical viewpoint. The need for the validity of the premises to be equivalent to the validity of the conclusion enforces a greater discipline in the deduction system development and facilitates elimination of possible loopholes. This feature is reinforced by the principles of R–S methodology outlined above, which give a concrete prescription for developing a complete deductions system in a systematic way. Another advantage is that invertible rules are better suited to proof search which facilitates automated deduction.

In view of all these advantages, Rasiowa–Sikorski deduction mechanism certainly merits a greater popularity and interest as a very useful tool in many applications. It is the author's hope that this paper might contribute towards achieving this goal.

Acknowledgements

The author wishes to express her gratitude to the anonymous referees for their very thorough refereeing work and many constructive suggestions, which helped her prepare an improved final version of the paper.

References

[1] E.W. Beth, The Foundations of Mathematics, North-Holland, Amsterdam, 1959.

[2] M. Bialasik, B. Konikowska, A logic for non-deterministic specifications, in: E. Orłowska (Ed.), Logic at Work: Essays Dedicated to the memory of H. Rasiowa, Studies in Fuzziness and Soft Computing, Springer, Berlin, 1999, pp. 286–311.
[3] A. Blikle, Three-valued predicates for software specification and validation, Fund. Inform. 14 (4) (1991) 387–410.
[4] A.G. Cohn, A many sorted logic with possibly empty sorts, Proc. CADE-11, Lecture Notes in Artificial Intelligence, Vol. 607, 1992, pp. 633–647.
[5] I. Düntsch, B. Konikowska, A multimodal logic for reasoning about complementarity, J. Appl. Nonclassical Logic 10 (2000) 273–301.
[6] I. Düntsch, E. Orłowska, Logics of complementarity in information systems, Math. Logic Quart., 2000, to appear.
[7] M. Frias, E. Orłowska, A proof system for fork algebras and its applications to reasoning in logics based on intuitionism, Logique et Anal. 150–151–152 (1995) 239–284.
[8] D.H. Gabbay, H.J. Ohlbach (Eds.), Temporal Logic (ICTL'94), Lecture Notes in Computer Science, Vol. 827, 1994.
[9] H. Hussmann, Nondeterministic algebraic specifications and non-confluent term rewriting, J. Logic Programming 12 (1992) 237–255.
[10] G. Koletsos, Sequent calculus and partial logic, M.Sc. Thesis, University of Manchester, 1978.
[11] B. Konikowska, Two over three: a two-valued logic for software specification and validation over a three-valued predicate calculus, J. Appl. Nonclassical Logic 3 (1) (1993) 39–71.
[12] B. Konikowska, A logic for reasoning about relative similarity, in: H. Rasiowa, E. Orłowska (Eds.), Special Issue of Studia Logica, Reasoning with Incomplete Information, Studia Logica 58 (1997) 185–226.
[13] B. Konikowska, A logic for reasoning about similarity, in: E. Orłowska (Ed.), Incomplete Information: Rough Set Analysis, Physica-Verlag, New York, 1998, pp. 462–491.
[14] B. Konikowska, Natural deduction systems for Rosser–Turquette many-valued logic, J. Multiple-Valued Logic 4 (3) (1999) 181–205.
[15] B. Konikowska, Rasiowa–Sikorski deduction system: a handy tool for computer science logic, Proceedings WADT'98, Springer Lecture Notes in Computer Science, Vol. 1589, Springer, Berlin, 1999, pp. 183–197.
[16] B. Konikowska, M. Bialasik, Reasoning with first-order nondeterministic specifications, Acta Inform. 36 (5) (1999) 375–403.
[17] B. Konikowska, C. Morgan, E. Orłowska, A relational formalisation of arbitrary finitely many valued logic (special issue), Logic J. IGPL 5 (6) (1997) 755–774.
[18] B. Konikowska, E. Orłowska, A relational formalisation of a generic many-valued modal logic, in: E. Orłowska, A. Szałas (Eds.), Relational Methods for CS applications, Physica-Verlag, New York, 2001, pp. 183–201.
[19] B. Konikowska, A. Tarlecki, J. Blikle, A three-valued logic for software specification and validation, Fund. Inform. 14 (4) (1991) 411–453.
[20] W. MacCaull, Relational proof system for linear and other substructural logics, Logic J. IGPL 5 (5) (1997) 673–697.
[21] E. Orłowska, Relational proof system for relevant logics, J. Symbolic Logic 57 (1992) 1425–1440.
[22] E. Orłowska, Relational proof systems for modal logics, in: H. Wansing (Ed.), Proof Theory of Modal Logics, Kluwer, Dordrecht, 1996, pp. 55–77.
[23] H.J. Ohlbach, C. Weidenbach, A resolution calculus with dynamic sort structures and partial functions, in: L.C. Aiello (Ed.), Proc. ECAI, Pitman, London, 1990.
[24] H. Rasiowa, R. Sikorski, The Mathematics of Metamathematics, Warsaw, PWN (Polish Scientific Publishers), 1963.
[25] M. Walicki, S. Meldal, A complete calculus for multialgebraic and functional semantics of nondeterminism, ACM TOPLAS 17 (2) (1995).
[26] C. Weidenbach, Unification in sort theories and its applications, Ann. Math. Artif. Intell. 18 (1996) 261–293.
[27] P. Wolper, The tableau method for temporal logic: an overview, Logique et Anal. 28 (1985) 119–152.