Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

X Ciclo – 1998

Efficient Approximate Deduction and an Application to Computer Security

Fabio Massacci

Università degli Studi di Roma "La Sapienza"

Dottorato di Ricerca in Ingegneria Informatica

X Ciclo - 1998

Fabio Massacci

Efficient Approximate Deduction and an Application to Computer Security

Thesis Committee

Prof.ssa Luigia Carlucci Aiello (Advisor)
Prof. Marco Schaerf
Dott.ssa Fiora Pirri

AUTHOR'S ADDRESS:
Fabio Massacci
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
E-MAIL: massacci@dis.uniroma1.it
WWW: http://http://www.dis.uniroma1.it/∼massacci/

Abstract of the Dissertation

The use of logics and formal methods for specification and verification requires effective deduction methods which can be efficiently implemented.

To this aim, the first part of this dissertation is devoted to the design and validation (both theoretical and experimental) of a number of efficient tableau inference systems, which constitute a significant advance with respect to other state of the art deductive systems. This parts starts by enhancing the classical propositional tableau calculus with an operation for constraints propagation (called simplification). This technique subsume a number of other approaches presented in the literature in a uniform framework. For instance, it is shown that Davis-Putnam and `KSAT` procedures are variant of tableau calculi with simplification. This framework is lifted to propositional multi modal logic by introducing the concept of prefixed tableau and incorporating the notion of modal constraints propagation. The extension to propositional dynamic logic with converse is carried forward by generalizing a combination of techniques used in modal theorem proving, model checking for temporal logics and automata theoretic approaches. This is the first tableau for converse PDL, thus solving an open problem by Pratt.

The computational characteristics of the various techniques are evaluated, either by extensive experimentation on relevant benchmarks (eg IFIP benchmark on hardware verification) or by careful complexity analysis. As modal and dynamic logic theorem proving is complex, we carefully analyze computational properties such as modularity, query combination, the design of decision procedures and the complexity of search strategies, possibly generalizing our results to other deduction methods such as those based on first-order translations and resolution.

Efficient procedures are not enough and, to tame the complexity of logical inference, a second step is taken: the transformation of efficient exact deduction methods into anytime approximate deduction systems which can be interrupted at any time, and whose precision improves with time. A general theory, which capitalizes in essential ways on the deductive machinery designed in the first part, is introduced for propositional and modal approximation. Only sound, only complete and multi-directional approximation of

the classical notion of proof are presented. These approximations are shown to converge to the classical notion of proof and we discuss the guarantees on the quality of the approximation provided by this anytime deduction mechanism.

In the last part of the dissertation – to show the potentials of our approach – we show an example of the application of logic as specification and verification language to a problem of computer security (namely access control). Among the various possibilities we single out the logic for access control developed at the Digital System Research Center by Abadi, Lampson et al. and design a tableau based deduction method for the logic based on these techniques.

The overall claim of this dissertation is that tableaux can be used as efficient, effective and approximate tools for automated reasoning and formal verification in a wide range of practical cases without losing their natural ease of use.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, my deepest debt of gratitude goes to Luigia Carlucci Aiello, my advisor. Gigina has been a great advisor who constantly inspired and encouraged me. Her advice was invaluable and she practically taught me how to do research. Her special style of 'long distance supervision' was the right thing for me at the right time, leaving me enough elbow room to follow side routes, but also making sure that I didn't forget the main route: writing this dissertation.

I would like to thank all members of the Dipartimento di Informatica e Sistemistica of the University of Rome "La Sapienza", and in particular the members of the Artificial Intelligence group, a working environment which has always been stimulating. In particular I would like to thank Fiora Pirri, Marco Cadoli and Marco Schaerf and my co-authors Francesco Donini, Giuseppe De Giacomo, Daniele Nardi, Riccardo Rosati. My warmest thanks go to Francesco and Fiora for many enlightening conversations.

I am also indebted with Larry Paulson, who gave me his hospitality for one year at the Computer Laboratory at the University of Cambridge. He has pointed to me the importance of implementation and the need for experimental and applicative validation of my research. His suggestions on many research topics have been precious.

The Computer Laboratory was much more than just a friendly and pleasant working environment. I learned more during the meetings of the Automated Reasonign Group and the Computer Security Group, the project meeting on authentication logic with Larry and Roger Needham, than I could have done with many books. I find their suggestions stimulating and challenging.

There are a number of people with whom I have discussed many research topics and who have commented my works. Without them, my research work would have been more difficult and less interesting. In particular, I would like to thank Martin Abadi, Rajeev Goré, Andreas Herzig, Fausto Giunchiglia, and Roberto Sebastiani.

I express my gratitude to the members of the Ph.D. Committee of the Dipartimento di Informatica e Sistemistica, who helped me in the accomplishment of this work. To Larry Paulson and Mukesh Dalal for acting as external referees and to Marco Schaerf and Fiora Pirri for being internal referees, and to Giorgio Ausiello as president, for his patience and understanding when I struggled with this dissertation.

I would also like to thank the University of Roma "La Sapienza" for its PhD fellowship which makes this work possible. I gratefully acknowledge the support of other sponsors who financed my journeys around the world: the American Association for Artificial Intelligence, CADE Inc. for the Woody Bledsoe Award, the Italian Research Council (CNR), the Italian Space Agency (ASI) and the Ministry of University and Scientific and Technological research (MURST).

Last but not least, special thanks to my family and my beloved Beatrice, they encouraged me and gave me strength in difficult moments.

Roma, Aprile 1998,

Fabio Massacci

Chapter 1

Reasoning for Formal Verification

In this chapter we clarify the premises, the scope and the results of the dissertation.

We review the successful use of logics for the formal analysis and reasoning, the development of automated reasoning systems for inference in those logics, and set down a number of objectives for efficient deduction. Even with efficient systems, logical reasoning is computationally hard and we discuss the intuition behind the possibility of transforming exact deduction methods into deductive systems which can be interrupted at any time and yield an approximate answer which improves with time.

Finally we give an overview of the dissertation, illustrate the achievement of the objectives of efficient and approximate deduction and the application to a problem of computer security.

1.1 Logics for Formal Specification and Verification

The use of both classical and non-classical logics as specification and verification languages is widespread in Computer Science and its uses range from artificial intelligence to hardware verification, from the analysis of distributed systems to computer security.

The basic *propositional logic* has always been used as reference point for theoretical computer science and the analysis of computational complexity. It is not only a theoretical tool but a concrete specification language for many (hard) subproblems in hardware verification [19, 27, 149]. Traditional AI planning problems can also be efficiently embedded into propositional logic, as advocated by Kautz & Selman [94].

More expressive logics, such as *modal logics*, have traditionally been used in artificial intelligence and philosophical logic for modeling knowledge and belief (see the monograph by Fagin et al. [57]) or legal reasoning and obligations, a use which dates back to Kanger [93]. New applications have been found in distributed computing by Fagin & Halpern [80] and protocol verification by Dwork & Moses [52] where new results where proved using modal logics.

Modal logics have also been successfully used for computer security. After the seminal paper by Burrows, Abadi & Needham [20], they are now one of the formal tool employed in the verification of authentication and cryptographic protocols. See also the overview by Meadows [127] or the work of Syverson [168, 169]. Another important application has been the verification of access control proposed by Abadi et al. [1], Cuppens & Demolombe [33], and Glasgow et al. [75]. We will review this possible application in Chapter 8.

In the large family of modal logics, *temporal logics* and *dynamic logics* have also found plenty of applications. One may see the work of Manna & Pnueli [107] or the handbook by Moller & Birtwistle [130] for the verification of concurrent systems,the review of McFarland [126] for hardware verification the original work of Fisher & Landner [60] for reasoning about programs (see also the recent overview by Kozen & Tiuryn [95]). Dynamic logics has been used for computational linguistics by Blackburn & Spaan [18], for conditional logic by Friedman & halpern [65] and for knowledge representation languages by Schild [156]. For instance, the research in knowledge representation points out a tight correspondence between dynamic logics and description logics, a family of expressive class-based knowledge representation formalisms[182].

First and higher order logics have also been widely used in hardware verification with the system `HOL` [5, 27, 78] and in the analysis of security protocols by Marick [111] and Paulson [142]. A general overview can also be found in the paper of Rushby [149].

The unwinding of new applications areas and the progressive attention towards more complicated (real) problems require automated reasoning tools, for proving properties and reasoning about the systems under analysis.

1.2 Calculi for Automated Deduction

The development and evaluation of (more or less automatic) proof systems for classical and non-classical logics have almost been parallel to the use of those logics. The relative efficiency of proof systems for propositional logic has been extensively studied since the seminal paper by Cook [30] and a comprehensive survey has been given by Urquhart in [171].

Proof systems for propositional (and first order) logic have been 'on the market' for long time and one may choose between a number of generic methods such as traditional Gentzen calculi [70], possibly using the uniform system of Smullyan [164]. Tableau methods are available from Smullyan [163], Fitting

[63], or D'Agostino & Mondadori [35]; possibly optimized for clausal normal form by Letz et al [103] or Baumgarten et al. [8]. One can also choose the inverse method of Maslow [112] or the resolution rule by Robinson [148] — see also the monographs of Bibel [17] and Fitting [63] — or the matrix method of Bibel [17]. Ad hoc procedures can also be used such as the Davis-Putnam procedure [41, 42], algorithms based on Binary Decision Diagrams proposed by Bryant [19, 170], or the hyper-graphs techniques of Gallo & Pretolani [67].

Modal theorem proving has also been the subject of a long research: one may find Gentzen-type systems in the 1959 work by Ohnishi & Matsumoto [138] or in the comprehensive monograph by Fitting [62]. Tableaux based methods can also be found there [62] or in the work of Kripke [96], possibly enhanced with labelled formulae following Gabbay [66], Fitting [62] or Massacci [113]. Resolution methods, proposed by Abadi & Manna [2] or Enjabert & Fariñas del Cerro [54], are also available. New systems have been developed based on translations into set theory by D'Agostino et al. [34] or into first order logic using theory resolution as proposed by Gent [69], the special unification algorithms for resolution by Ohlbach [135], or the matrix method by Wallen [178].

Deduction for temporal and dynamic logic can be performed by using either tableaux as done by Manna & Pnueli [107], Pratt [144], and Wolper [181], or automata on infinite trees following the works of Vardi & Wolper [174, 175] and Street & Emerson [167].

Besides fully automatic systems, the last years have seen the development of *interactive theorem provers* where the construction of the proof is guided by the user with more or less automatic proof tools and tactics. Systems like `HOL` [5, 78], `Isabelle` [141] or `PVS` [149] have been successfully used for tackling a number of problems in hardware and software verification. Their proofs are often based on variants of the sequent (Gentzen) calculus in the attempt to combine effective proof search with human-oriented proof presentation.

Among the various methods (fully automatic and interactive) tableaux and sequent calculi are undoubtedly the most flexible ones. They do not require normal forms, can be easily adapted from one logic to another one, are reasonably efficient and, last but not least, offer human-readable proofs.

However, when it comes to decision procedures, it is a widespread belief that methods based on the sequent calculus (such as tableaux) are hopeless for "real life" satisfiability and validity search. Even for decidable problems with a natural appeal for tableaux, such as modal or propositional logics, experimental results have shown that other algorithms outperform them by orders of magnitude. For instance we may see the work of Crawford and Auton [31], Selman et al [162] and Zhang & Stickel [184] for propositional logic, and Giunchiglia & Sebastiani [72] or Hustadt & Schmidt [90] for modal logic.

Such a huge gap leads to think that this is not a matter of efficient implementations but rather of theoretical nature. The key question is whether such a gap is inherent to tableau methods or something is simply missing. This answer is not only of theoretical relevance but has an extreme importance for the future development of hardware and software verification tools.

If tableau-like methods are "hopeless by nature", then using ad-hoc tactics (in the general calculus) for decidable sub-theories would be impractical for hw/sw verification. Rather, non sequent-based systems (outside the prover) should be consulted as oracles. For instance the use of particularly efficient algorithms for propositional logics as one-off "derived rules" for HOL has been proposed by Harrison [83, 84]. The concept of proof (and verification) would then be shaded by many black-box steps.

1.3 A First Step: Efficient Deductive Systems

The design of flexible tableau calculi (with related provers) for a wide range classical and non-classical logics, faces a number of problems in the attempt to obtain something which is simple, effective and efficient.

The basic requirement is, of course, the *soundness and completeness* of the calculus with respect to the underlying logic, or at least large fragment of it.

The next step is the evaluation of its *computational complexity*. However, other computational properties are also equally important but are sometimes neglected in the analysis of proof systems and theorem proving. This usually happens because the traditional focus has been on the analysis of propositional logics where proof complexity is really the only difference between two systems. As soon as we lift from propositional logic to modal logics we face a number of different problems.

At first there are many modal logics, each representing a different introspection capability of an agent [57, 82]. So an objective may be *modularity and cumulativity*, in the same fashion as Hilbert style axiomatizations:

Question 1.1 *Can we switch from logic to logic by simply adding or deleting axioms or inference rules in a modular way?*

It is also useful to devise a general methodology for completeness and soundness proofs so that we do not start from scratch each time we change logic.

With so many modal logics around, we often want to know whether "A is a theorem for logics L_1, $\ldots \mathsf{L}_n$". For instance, when modeling knowledge and belief, each L_i corresponds to different introspection capabilities, and we may want to check which capabilities are sufficient to derive A.

Question 1.2 *Can we design a calculus which capitalizes on the computational efforts? A calculus which proves that A is a theorem for the logics L_1,*

...L_n, by spending most (possibly exponential) time on the first logic and then use this information to speed up deduction in the remaining logics?

Intuitively, we are asking whether we can *combine validity queries* from one modal logic to another modal logic. This may seem very simple for some deduction methods and some authors have also (informally) claimed it possible for their system. We will show that, unfortunately, this is impossible.

Once a logic L is fixed, and soundness and completeness are proved, we must search for a proof and a calculus may have many rules to choose from. We expect that different search strategies will have different time or space complexities but we would like to avoid more substantial losses:

Question 1.3 *Does the order in which we select rules or formulae matter for completeness?*

The property we are looking for is *proof confluence.* Loosely speaking, confluence means that the order in which we select the rules does not substantially matter: by choosing a proof strategy instead of another one, we can waste computational resources but we cannot lose the possibility of finding a proof. Proof confluence is an immediate property of any (complete) propositional proof systems and one of the advantages of resolution in first order theorem proving. We shall see that not all modal deduction methods satisfy it.

The selection of different rules may not matter for completeness but it can still make a difference for proving the *termination* of a search strategy. This is not an issue in theorem proving with propositional logic (it is trivial) nor for full first order logic (it is impossible), but it is an important topic for modal and dynamic logics. These logics are decidable but it is easy to get trapped into infinite (pointless) computations.

Question 1.4 *Can we devise techniques for blocking infinite computations, thus yielding a decision procedure?*

Proving that suitable techniques give a *decision method* is the first step but it is not enough. Termination checks can be expensive, and may require to handle the whole proof constructed so far. For instance see the work of Demri [49] on modal logic or Bhat et al. [16] on temporal logic.

Question 1.5 *Can we devise termination checks which consider each formula (or each rule) in isolation and only require polynomial time?*

Here we ask for a *local termination check.* For validity, we have a positive result: such checks are possible for the tableau calculus that we have designed and can be transferred to translation methods into first order logic, such as those by Ohlbach [136]. Unfortunately, they are not possible for logical consequence in modal and dynamic logics.

Dynamic logics are more expressive than modal logics and the deduction procedures becomes more complicated: representing models as graphs we can now express the fact that a property will eventually hold without specifying how long is the path to reach a node of the graph where this property holds.

These properties are called *eventualities* and their verification is one of the hardest problems in temporal and dynamic logic. Indeed, the classical way to cope with them is to construct first a tentative model for logics of knowledge and belief and then check whether the eventualities are satisfied in this model (see section 5.2 for an intuitive explanation). The problem is that the tentative model may well be very large (of exponential size) and we may discover that we have to discard it at once just after the first eventuality. Therefore:

Question 1.6 *Can we merge the verification of eventualities with the standard modal deduction?*

The last step is the analysis of the *computational complexity* of various search strategies and algorithms for validity and logical consequence. Validity checking is "only" CO-NPTIME-complete for classical propositional logic and the modal logics K45 and S5 (for belief and knowledge with perfect introspection), whereas it is PSPACE-complete for the modal logics between K and S4, a result which dates back to Ladner [98]. Logical consequence is EXPTIME-complete for modal logics (see Halpern & Moses [82]) and for dynamic logics (see the papers by Fischer & Ladner [60] or Pratt [146]). We would like these facts to be reflected into efficient strategies for theorem proving:

Question 1.7 *Can we design simple restrictions on the strategies for the proof search, so that the worst case complexity of validity checking with those strategies will match the lower bound complexity of the problem?*

For instance we would like a search strategy for validity in K which only requires an amount of polynomial space in the size of the input formula.

Some calculi may fail to match the lower bounds, and others can be very sensitive to the search strategy. For instance, translation methods based on generic first order resolution use exponential rather than polynomial space[1].

After we have given a positive answer to these questions (if at all possible), we are still far from efficient procedures.

We have already noted that some hybrid decision methods have been proved superior to tableaux (respecting the overall worst case complexity) and that probably there are some theoretical reasons behind it.

The traditional explanation, given for instance by D'Agostino & Mondadori [35] or Urquhart [171], is the absence of the cut-rule. There is also

[1]The superior performance of translation methods with generic resolution provers in Hustadt & Schmidt [90] may be simply due to better implementation.

another, orthogonal reason which can be better clarified by a comparison with the historical competitors of tableau methods. Since the very beginning, Davis-Putnam procedure [41] or its enhanced variant [42] in cooperation with Longeman and Loveland (`DPLL` for short), and the resolution of Robinson [148] included rules for the simplification of the formulae to be proved unsatisfiable (e.g. unit or subsumption) *without* changing the basic inference mechanism.

In a nutshell, techniques for "eliminating the irrelevant from mechanical proofs", as noted by Davis [40], were immediately considered and embedded into the proof procedure. Difficult problems could hardly (if at all) be solved by either resolution or `DPLL` without using them. On the contrary, even in the modern texts on tableaux, such as the monograph by Fitting [63], everything is proved by "first principles".

This leads to the following questions:

Question 1.8 *Can we devise rules for embedding constraint propagation in tableaux for classical and non-classical logics?*

The experiments reported in this dissertation show that constraint propagation has an important impact on the efficiency of the system. Indeed, efficient hybrid methods such as `KSAT` [72], or the very Davis-Putnam procedure, can be seen to be variants of tableaux using constraint propagation.

Yet, no matter the effects of constraint propagation or analytic cut, we are still bound by the computational complexity of the logical problems.

1.4 The Complexity Barrier and Approximation

Automated reasoning procedures are also (mainly) hindered by the hardness of the decision problem in the corresponding logics (we assume only a very basic knowledge of complexity theory[2], see [92] for an introduction to complexity classes). Deciding validity or logical consequence in propositional, modal and dynamic logics can almost never be done in polynomial time in the size of the input (the theorem to be proved and the set of assumptions).

[2]The class PTIME (NPTIME) is the class of decision problem that are solvable in polynomial time by a deterministic (nondeterministic) Turing machine. Problems in EXPTIME can be solved by a deterministic Turing machine using in exponential time i.e. time bounded by $O(2^{poly(n)})$ where $poly(n)$ is a polynomial function in the size n of the input. As for PSPACE and EXPSPACE, the machine is deterministic and works in polynomial (exponential) space. As far as the relations between complexity classes are concerned, it is known that PTIME $\subseteq$ NPTIME $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ EXPSPACE. Only the trivial relations PTIME $\subset$ EXPTIME and PSPACE $\subset$ EXPSPACE have been proved but it is widely believed that all containment relations are proper [92] and proving that a theorem holds only if PTIME=NPTIME is almost equivalent to prove that it is impossible.

Indeed, satisfiability for the propositional calculus [30] and the modal logics $\mathsf{S5}$ and $\mathsf{K45}$ are NPTIME-complete problems [82, 98] together with the logic Cxt whose satisfiability remains in NPTIME also for the multi modal case [116]. Deciding satisfiability and validity for all modal logics between K and $\mathsf{S4}$ is PSPACE-complete [98] and so is for all multi modal logics between K_m and $\mathsf{S5}_m$ with $m \geq 2$ [82]. Deciding logical consequence for K_m and for $\mathsf{S4}_m$ with $m \geq 2$ is EXPTIME-complete [82]. Validity for propositional dynamic logic is also EXPTIME-complete [144].

A visible effect of the membership of a problem in a complexity class is the bound on the size of the smallest model which can satisfy a formula (see Chapter 4 for further details).

Anyhow, the practical effect is that solving validity or logical consequence for these hard problems requires exponential time.

The basic method for getting a tractable inference procedure is brute force: *language restriction*. The typical case is the use of Horn clauses for propositional logics or similar restrictions for which polynomial time algorithms exists (e.g. Gallo & Scutellà [68]). Further references for nonmonotonic reasoning and knowledge representation can be found in the works of Cadoli [22] and Levesque [105].

Yet, cutting the construct which increases the complexity of the language does not seem a good choice. It has little sense for programming languages and, as successfully argued by Doyle & Patil [51], it is also not the case for logical languages for reasoning or formal specification.

To overcome this computational barrier, approximating algorithms for satisfiability (i.e. problems in NPTIME) have been proposed from the very beginning by Garey & Johnson [91] and are now an established field (see Ausiello et al. [6] for an overview). On the contrary, much less research has been carried out for its dual, deduction. Approximation is only used for proving properties of (exact) deduction in Frege systems [13] by using unsound steps in a proof.

The major research efforts have been carried in artificial intelligence for propositional entailment: knowledge compilation, semantical approximations, and syntactic approximations.

Knowledge compilation or "vivification" is a preprocessing step to transform a theory into another one for which deduction is faster. According to Selman & Kautz [161] we do not approximate the inference procedure, rather we simplify the theory using a Horn relaxation of the problem for which fast algorithms are known. Similar forms of compilation proposed by Dalal [37] or del Val [47] use target theories for which boolean constraints propagation and unit-resolution respectively are complete procedure. Compilation is a one-off processing of a set of formulae, and doesn't exactly match the intuition of approximation of logical consequence as an incremental process: if the procedure fails we must resort to the original calculus for an answer.

Another approach, proposed initially by Rantala [147] and later on by Dalal and Etheringthon [37, 39], uses syntactic approximation of inference procedure and can be made incremental by an essential use of cut or, in more constructive form, by iterated steps of knowledge compilation as done by Dalal [37] and del Val [47].

The major problem of both approaches is that the compilation process leads to an exponential blow up of the approximating theories [22, 37, 161].

Logical approximation,proposed initially by Levesque [104] and developed further by Fagin et al. [55, 56], Lakemeyer [99, 100] and Cadoli & Schaerf [155], is based on semantical approximation of classical logic by multiple valued logics. It is incremental and can be used for both sound or complete approximations. Although algorithms for incremental entailment exist [155] they are based on model enumeration and do not yield a notion of proof (besides making difficult to re-use past computations).

Notice that we should distinguish between logics or deduction for approximate reasoning, such as fuzzy logic by Ying [183] or the multi-valued logic of Belnap [14] from *approximating deduction*. Multi-valued logics aim at modeling uncertainty and imprecise concepts, and the corresponding satisfiability problem has been proven NP-complete by Mundici [131].

Beside the limitation to clausal normal form, which is common to almost all tractability results in the cited literature, something else is missing.

In automated reasoning and in formal verification in particular, the notion of *proof* is paramount [149]. For most interactive theorem provers, the ability to guarantee the correctness of the proof has taken precedence over the development of automatic tools for generating it.

We may be willing to trade-off accuracy for speed, but we do not want to throw away entirely the logical framework established by classical logic and classical deduction. In the words "unbounded logical omniscience" we are not disturbed by the "logical" or the "omniscience". As pointed out by Russel & Zilberstein [150] we are (computationally) annoyed by the word "unbounded". Exact proofs require unbounded (and hence computationally expensive) reasoning. We need a framework where we can trade-off the "precision" of a proof in change of deductive speed.

1.5 A Second Step: Approximate Deduction

What we are looking for, has been already advocated by Wang in 1960 [179]:

> If we compare calculating with proving... we do not have a clear conception of approximate methods in theorem proving. We possess efficient calculating procedures while with proofs... even in a decidable theory, the decision method is not practically feasible.

> [...] The concept of approximate proofs, though undeniably of a kind other than approximations in numerical calculations, is not incapable of more exact formulation in term of ... gradual improvements towards a correct proof.

The idea of approximating deduction stems from an engineering perspective:

Question 1.9 *If exact proofs are too hard to find, why not limiting the search to* approximate proofs*?*

To provide an answer in this direction, a comparison with the problem of approximating satisfiability is useful. Satisfiability requires a *witness* which satisfies all constraints (represented by formulae in some logic). An approximate witness only satisfies part of them. Deduction has the dual notion of *proof* e.g. a proof that a counterexample witness does not exists (with refutational theorem proving).

We can introduce the notion of *approximate proofs* which satisfy only some of the constraints a classical proof must have. For instance, we can consider the notion of a calculus whose proofs are complete approximation of classical proofs. If we cannot find a proof in the approximate calculus then we are guaranteed that no exact proof exists. However, finding a proof in the approximate calculus, i.e. an approximate proof of the exact proof, does not guarantee us that an exact proof exists. Sound approximations have a dual property: the existence of an approximate proof implies the existence of an exact proof. On the contrary, if we cannot find an approximate proof we are not guaranteed that an exact proof does not exist.

The development of approximate calculi requires the solution of additional problems. The first one regards the nature of the approximation:

Question 1.10 *Can we have sound (but incomplete), complete (but unsound) or multi-directional approximations of a classical proof?*

In a nutshell we are asking whether we can generate the proof space of Fig. 1.1.

Potentially incomplete approximation are quite common and well understood. But what about complete but potentially unsound approximations? This means that we are introducing more consequences than we should (in classical logic). So, we may end up introducing inconsistencies in our system. This is precisely what happens if one uses the methods based on abstraction by Giunchiglia & Walsh [73]. The introduction of inconsistency spreads over the whole systems and makes the approximation useless.

Question 1.11 *Can we devise an approximation calculus which is* robust*, so that an inconsistency due to a local approximation does not propagate throughout the systems?*

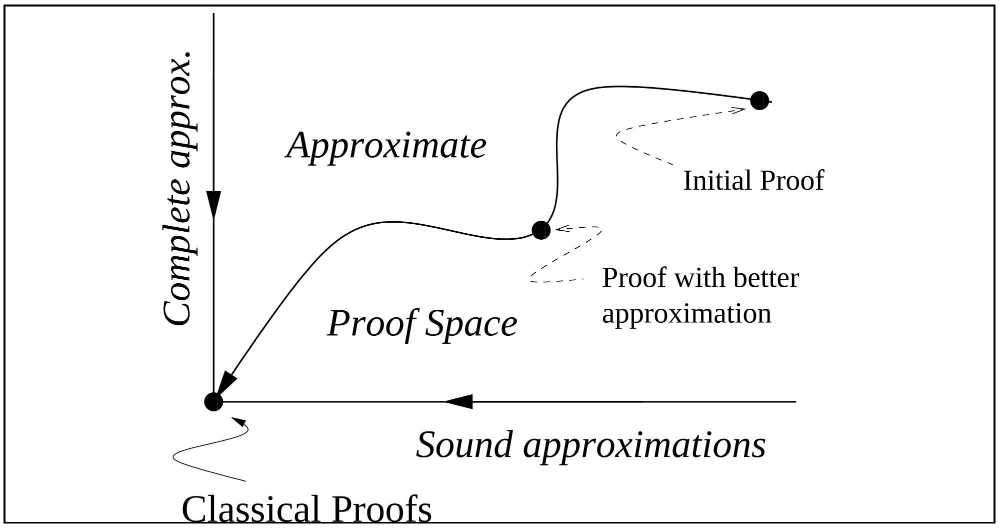


Figure 1.1: A Multi-directional Approximate Proofs Space

In this framework we need an approximating calculus where proofs converge to classical proofs, with increasing precision. This means that we expect some sort of *incremental convergence guarantee*:

Question 1.12 *Can we prove that approximate proofs eventually converge to classical proofs?*

In practice, more than simply approximate proofs, we want a calculus able to provide a *sequence of approximate proofs*. For instance a calculus for sound approximations should be able to produce a sequence of approximate proofs $\vdash^k$ such that

- if $KB \vdash^k A$ then $KB \models A$, i.e. the approximate proofs are sound with respect to classical entailment;
- if $KB \vdash^k A$ then $KB \vdash^{k+1} A$, i.e. the approximation increase monotonically with k
- for every A and KB there is an n such that $KB \vdash^n A$ iff $KB \models A$, i.e. the approximation converges to the exact classical answer.

There is a number of other characteristics which we have implicitly assumed throughout this thesis: we do not want a language restricted to a particular normal form and we would like to avoid any form of language restriction to guarantee tractability.

Among them, one seems particularly important:

Question 1.13 *Can we re-use the entire general deductive machinery we have defined for propositional, modal and dynamic logics just by adding some approximation steps in the proof?*

This is important if we want to upgrade existing deductive systems without devising ad-hoc systems for our notion of approximation. Intuitively we are asking whether a proof can be "almost classical" we only have to revise the approximation steps. We can ask for something more than simple convergence.

Question 1.14 *If no approximation step is used, do we obtain a classical proof?*

Classical and non-classical logics have a semantics and proofs are assumed to be sound and complete with respect to the semantics. We ask for an *approximate semantics* so that approximate proofs are approximately sound and approximately complete for the approximate semantics.

The final requirement regards the complexity of approximate proof search: approximations make sense if one trades accuracy for speed.

Question 1.15 *Can we find an approximate proof in (exponentially) less time than a classical proof?*

We need a more stringent requirement because, as we noted above, we are not interested in simply once-off approximation but want a sequence of approximating proofs, which incrementally converge to the exact (classical) proof. We want to generate $\vdash^0$, then $\vdash^1$ etc. until we find a solution or time out the search. So, the cost of obtaining the first and roughest approximation $\vdash^0$ should be cheaper than the cost of the second step, and so on. Possibly, the cost of the overall sequence of approximate steps should not be larger than the cost of the exact solution (except for some constant factor).

This requires a concept which as been introduced in planning by Dean & Boddy [44] and further analyzed in a number of other studies (see Russel & Zilberstein [150] for further references): *anytime reasoning* . The main intuitions behind an anytime algorithm are rather simple:

- an anytime algorithm can be interrupted at any moment and always yields a partial answer;
- the quality Q_t of answers improves with time (not necessarily in a continuous way)
- we can set in advance a discrete series of time points $t_0, t_1, \ldots t_k$ such that after time t_k we are guaranteed that the quality of the solution is at least Q_k.

For problems requiring exponential time a good engineering principle (well known also in automated deduction) is to require that the passage from t_k to t_{k+1} is not linear but exponential in k [150].

Sequences of approximating proofs should therefore provide the performance guarantee of anytime deduction algorithms: each $\vdash^k$ should require at most time $t_k = O(2^{ck})$ for some c. So, if an answer (possibly approximate) is desired in time t, then one could choose a k such that $t_k \leq t < t_{k+1}$.

Question 1.16 *Can we arrange the proof search into an anytime decision procedure?*

The main intuition behind our solution is to construct an almost classical proof, where the 'interesting' (non approximable) elements of the proof increase with time.

1.6 An Overview of the Dissertation

This dissertation is divided in three main parts and its overall structure is sketched in Fig. 1.2 (except for the introduction and the conclusions).

1.6.1 The design of efficient tableau methods

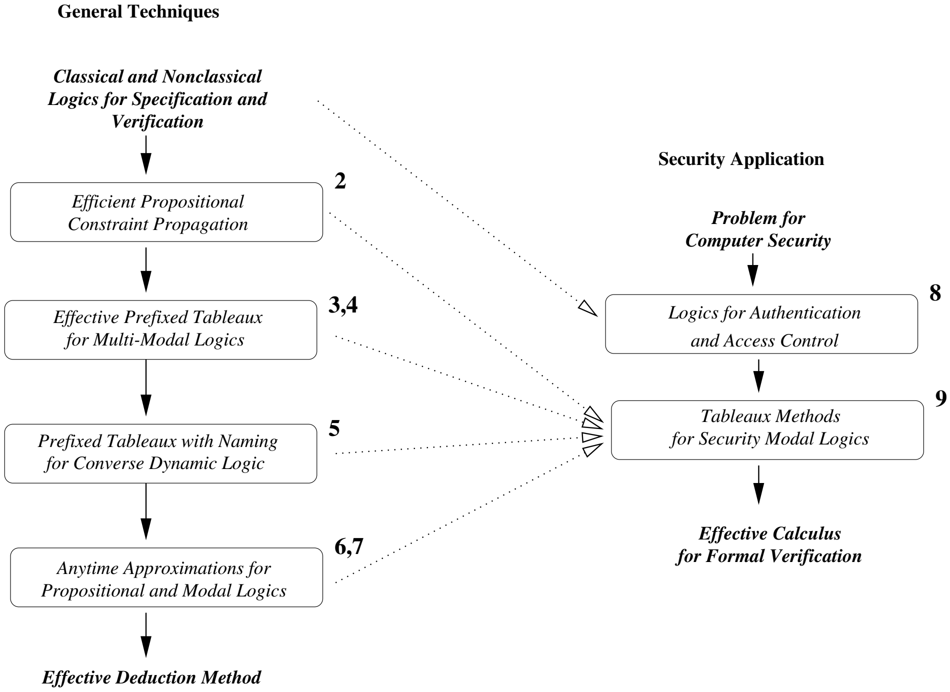
In Part I we discuss how to solve the various problems sketched in section 1.3 and tackle the design and (computational and experimental) analysis of efficient tableau prover for propositional, modal, and dynamic logics.

At first we focus on propositional logic, as the basic building block of almost all decision methods. We tackle, in particular, question 1.8.

We start *Chapter 2* by reviewing some of the enhancements of the standard tableau method and present the intuition behind our works (§2.1). In the rest of the chapter we recall the semantics of propositional logics (§2.2), introduce our tableau calculus enhanced with new rules for constraint propagation (§2.3), and give some examples to explain the way it works (§2.4).

Then we analyze related methods in more details (§2.5) and show how these new techniques can provide a uniform perspective of many different calculi proposed in the literature. For instance we show that the Davis-Putnam procedure is a simple variant of a tableau calculus with restricted version of cut and the rules for constraint propagation.

Beside this theoretical analysis we also carry an extensive experimentation on a number of standard benchmarks in artificial intelligence and hardware verification (§2.6). In particular we focus on the AI benchmark on random 3-SAT and the IFIP benchmark on hardware verification. For both of them



Results presented in this dissertation: an improved analysis of existing techniques (Chapters 2, 4, 5, 8), new techniques subsuming and improving current methods (Chapters 2, 3, 4, 5, 6, 7, 9), computational and experimental analysis of proposed methods (Chapters 2, 4, 5, 6, 7).

Figure 1.2: Results in this Dissertation

we show the important impact and the gain in efficiency of our method and relative importance of analytic cut and constraint propagation.

Chapter 3 focusses on (multi)modal logics for knowledge, belief and obligation. After a comprehensive overview of different methods of proofs for modal logic (§3.1), we recall the syntax and semantics of multi-modal logic (§3.2) and introduce a tableau calculus for a wide range of modal logics (§3.3). In particular we use prefixed tableaux i.e. we label modal formulae with special labels containing model-theoretic information and use rules for both formulae and labels. Also in this case we enhance the calculus with rules for modal constraint propagation (§3.4), and compare it with other traditional tableau methods and hybrid methods (§3.5). For instance we show that the "new" method `KSAT` is a restricted variant of a tableau calculus with only propositional constraint propagation. Finally we prove it sound and complete (§3.6).

The analysis of the computational properties in the case of modal logic is more complicated, as we have already pointed out in section 1.3. Having answered Question 1.8 with the introduction of modal constraint propagation we devoted *Chapter 4* to the remaining issues, which are briefly recalled in section 4.1.

At first (§4.2), we focus on modularity and query combination (Question 1.1 and 1.2 in section 1.3) and discuss the properties of our calculus in comparison with other formalism. In this case we are able to prove that it is substantially impossible (no matter the calculus).

Then we focus on proof confluence (§4.3) comparing our methods with others and proving its confluence, and design a decision procedure based on termination checks (§4.4). We extend these results to other deduction methods based on translation into first order logic. Finally, we focus on complexity theoretic issues and analyze the complexity of search strategies (§4.5) and find some minimal requirement to achieve the complexity lower bounds required in section 1.3. We have both positive and negative results. For instance, we discuss the impossibility of translation methods based on generic first order prover to match the lower bound, in contrast with some recent experimental claims. It is a caveat for experimental analysis: we must be sure that we are not comparing the relative efficiency of implementations.

We capitalize on this work in *Chapter 5* where we tackle propositional dynamic logic (also known as logic of programs) of which we recall the syntax and semantics in §5.3.

Designing efficient tableau methods for dynamic logic is more complicated. We gives an overview of the related methods in the literature and show how a number of techniques used in model checking for temporal logics and modal theorem proving are incorporated in this work (§5.2). This section provides a general sketch of the method without going into details. In particular we re-use prefixed tableaux for modal logics enhancing them with a new technique for dealing with eventualities. After the introduction of the tableau calculus (§5.4), we further clarify the method with a number of examples (§5.5).

We prove the soundness and the correctness of the method (§5.6) and discuss the design of efficient deduction algorithms (5.7). Many considerations in chapter 4 carry forward to the dynamic case (after all dynamic logic is a particular modal logic) and we only focus on the difficult issue of providing an answer to Question 1.7: the development of deduction procedures which matches the corresponding lower bounds.

We do not have space to deal with first order logic or nonmonotonic logics. The interested reader may look in Massacci [119] for constraint propagation techniques in first order logic. The application of these tableau methods to nonmonotonic logics can be found in Donini et al. [50].

1.6.2 Introducing anytime and approximation techniques

In Part II we discuss how to solve the various problems sketched in section 1.5: we introduce the concepts of propositional and modal approximations for the design of anytime approximate algorithms.

We capitalize on the general machinery developed in part I and modify the semantics and tableau methods just defined for the various logics to accommodate the notion of approximation.

The propositional case is treated in *Chapter 6* where we compare the various forms of approximation for the propositional calculus (§6.1). We provide a deduction oriented perspective of the field which makes these form of approximations less different from each other than they claim to be.

At first we modify the propositional semantics (§6.2) in a way that makes it possible to introduce sound, complete and multi-directional approximations to logical consequence, providing and answer to Question 1.10. The basic idea is to give a different status to different propositional letters so that a quick (polynomial time) analysis is sufficient to decide the approximate satisfiability and unsatisfiability of not interesting propositions and exponential resources are only spent on 'interesting formulae'.

Next we modify the classical tableau calculi in (§2.3) to cope with approximate logical consequence and present an anytime deduction procedure (§6.3). The quality of approximation is discussed at length in §6.4 where we provide an answer to the other questions of §1.5: approximate soundness and completeness of the calculus, the direction of different form of approximations for increasing the precision of the deduction process (with a convergence guarantee) and the anytime guarantee of the complexity of the algorithm.

We consider two alternative methods of approximation in details. In particular we analyze the semantical approximation designed by Schaerf & Cadoli [155] for sound and complete approximations (§6.5) and the syntactic family of anytime propositional reasoners by Dalal [37] for sound approximations. We show how both approaches are incorporated into our general framework.

The whole framework is upgraded to modal and dynamic logics, for which much less research work is available. In *Chapter 7* we explore two possible ways: lifting the propositional approximation 'as it is' to the modal framework and designing ad hoc approximations for modal logics. In the latter case the intuition is that agents have a fixed bound on the introspection ability. If the theorem we are trying to prove requires introspective reasoning beyond the bound we give up reasoning and take a credulous or skeptical attitudes. From the viewpoint of the logic of programs, this is equivalent to assume that, when the bound on the number of iterations is reached, then every program enter an incoherent state (where everything is possible) or an unknown state (where nothing is possible).

For both forms of approximation we provide a semantics for approximate reasoning (§7.2) which builds upon the classical modal semantics (§3.2) and the propositional approximation (§6.2).

In a way similar to the previous chapter we modify the classical modal tableau of section 3.3 to accommodate both modal and propositional approximation into an anytime algorithm (§7.3). Also in this case we discuss the semantical and computational guarantees that our anytime deduction procedure can offer (§7.4): approximate soundness and completeness, the directions of approximation, the increase in precision, and the convergence guarantee.

We also sketch the extension of the modal approximation to dynamic eventualities (§7.6).

1.6.3 An application to computer security

In Part III we present an application of the general techniques to a security problem: access control. We introduce a logic for reasoning about access control and develop a corresponding tableau method.

Chapter 8 starts with a general introduction to the problem of access control (§8.1) and presents an overview of the various logics which have been used for reasoning and formal analysis in this field (§8.2). This is a practical example of the use of logic for formal verification that we have presented in section 1.1.

In particular, we focus on the expressive modal logic for the analysis of access control in multi-agent distributed systems which have been introduced by Abadi, Lampson et al. [1, 101] at the Digital System Research Centre (DEC-SRC for short). We give the intuition and the semantics of the logic (§8.3) and clarify its computational features (e.g. its undecidability and the lack of the tree-model property).

We provide a number of examples of the possible situations which can be modeled with the logic (§8.4) from simple examples of multi-agent-system to more complex examples described in [1]. We also briefly discuss some possible extensions to tackle the concept of "role" of an agent (§8.5).

In *Chapter 9* we apply the exact and approximate techniques to the DEC-SRC calculus. The general machinery we have developed for modal and dynamic logic can now be re-used in large part and we can concentrate the proofs on the trickiest aspects of this logic. We present directly the whole tableau calculus (§9.1) and show some sample deductions (§9.2) for each of the examples we have given in §8.4. The soundness of our deduction method and the completeness for two large fragments of practical interest is shown in section 9.3.

Thank to our general machinery, there is no need to devise an ad ad-hoc method for adding constraint propagation or approximate techniques different from those of propositional and modal logics. So, we only briefly discuss the intuitive interpretation of our notion of approximation for this particular application (§9.4).

Part of the results of this dissertation appeared in preliminary form in [43, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122].

Part I

Efficient Tableau Deduction

Chapter 2

Tableaux with Constraint Propagation

We start the technical analysis of deduction methods with propositional logic, a brief review of the major approaches, and the intuition underlying this work.

The classical tableau calculus of Smullyan is enhanced with an operation for constraint propagation (simplification) and show how a number of seemingly different methods can be re-interpreted as variants of this approach. We show the practical impact of this operation with an extensive experimentation on industrial and standard benchmarks.

2.1 Making Propositional Tableaux Efficient

Tableau methods (or Gentzen-type calculi) are among the most used proof methods in fully automatic and interactive theorem proving (see also §1.2). For an introduction, the best text is undoubtedly the monograph by Smullyan [163] or the book by Fitting [63].

Their popularity stems for the ease of use and the clear semantics behind the rules. Yet, their use is hindered by the computational limitation of tableaux wrt other systems such as resolution (see the monograph by Urquhart [171]) and a number of works have been proposed to speed-up the proof search.

In the propositional case, the enhancement of the calculus has usually been done by adding extra rules, such as ad-hoc rules for modus ponens and tollens proposed by Baader & Hollunder for their system `KRIS` [7] or by Oppacher & Suen for `HARP` [139] and by D'Agostino & Mondadori for `KE` [35]. The use of controlled form of analytic cut has been advocated by Letz et al. for `SETHEO` [103] and by D'Agostino & Mondadori [35],

Other techniques affect the overall construction of the proof such as imposing regularity conditions as done by D'Agostino & Mondadori [35] or Letz et al. [103], or using factoring and merging of different part of the proof search as done by Murray & Rosenthal [132] and by Wallace & Wrightson [177]. We can also construct hybrid methods, combining features of the resolution rule into clausal tableaux as proposed by Baugarten et al. [8].

In some cases, Crawford & Auton [31], the quest for an efficient implementation has led from the initial tableau calculus to a (seemingly different) Davis-Putnam procedure [41, 42], which still is one of the most efficient propositional decision method.

We propose a general technique, which subsumes a number of these approaches, and whose intuition is due to an interesting comparison between tableaux and their "historical competitors". The key advantages of the Davis-Putnam procedure [41, 42], for short `DPLL`, and resolution [148] over tableaux is the presence of ad-hoc rules to simplify the formulae under analysis (respectively the unit rule and the subsumption procedure) *without* changing the basic inference mechanism. We delete redundant formulae or simplify their structure to avoid pointless branchings.

In a nutshell, techniques for "eliminating the irrelevant from mechanical proofs" (Davis [40]) were immediately considered and embedded in the proof procedure. The implementation of these simplification procedures is almost a research field in itself [176, 184] and their presence is essential for the solution of hard problems.

We advocate that this lack of "rules for simplification" is the main cause for the computational gap[1]. Hence we need a simple theoretical innovation: an operation *plays for the sequent calculus the same role as unit for* `DPLL` *and subsumption for resolution.* We call it *simplification.*

2.1.1 Intuitions

We assume a basic knowledge of the tableau calculus in this section. In the sequel formulae are denoted by A, B, C, sets of formulae by Γ, Δ and sequents by $\Gamma \Longrightarrow \Delta$.

Simplification can be explained by comparing it with subsumption.

The intuition behind simplification can be explained by comparing it with subsumption, to show the duality between top-down (tableaux) and bottom-up (resolution) methods. We assume that we have a refutation procedure: we prove that a formula is valid by refuting its negation (showing that it has no models). From this perspective we can see the formulae introduced during the proof search as *constraints* on the possible counter models for putative theorem.

[1]This has also been confirmed for modal logics by recent experimental studies by Hustadt & Schmidt [90] which clarified the gap between `DPLL`-based and tableau based procedures in modal logics.

Bottom-up methods (resolution, the inverse method) work in two stages:

Expansion generate new constraints by *combining* together old ones;

Contraction *discard* duplicate or redundant constraints which are subsumed by more concise information we already have.

A top-down method (Davis-Putnam procedure) works in a dual way:

Expansion generate new constraints, usually by *breaking* old ones;

Contraction *propagate* existing constraints to delete duplicate or redundant information.

The second step in the `DPLL` procedure is the unit rule: once we have established the constraint "literal l is true in the model", we can propagate this constraint throughout all other constraints to eliminate all duplicates of l.

In both methods *the contraction step is performed formula-wise.* For instance, in resolution we compare a clause with another clause and decide . Since individual comparisons are repeated for many formulae, we may use the set-at-a-time procedures by Voronkov [176] for efficiency's sake, but the essential nature of the contraction step remains an individual comparison between two formulae.

Yet, we have only expansion rules in the tableau calculus. For example, consider the following α-rule:

$$\frac{\Gamma, A \wedge B \Rightarrow \Delta}{\Gamma, A, B \Rightarrow \Delta}\,(\alpha)$$

After this rule a tableau will continue to break down connectives until a branch closure rule can be applied. We look for something like $\Gamma, C \Rightarrow C, \Delta$.

Yet, after the α-step, there are already some constraints that the calculus does not use to anticipate the outcome of its search: the constraint A must be satisfied, no matter which are the sub-constraints of A. We can use this information about A to simplify Γ and Δ before reducing other connectives.

The simplest way is syntactic search: look for all exact[2] occurrences of A in Γ and replace them with the constant $\top$; next perform some boolean reduction to eliminate $\top$ and $\bot$ from the formula; only afterwards continue with other rules.

The computational pay-off may seem doubtful: some time must be spent for scanning the formula and replacing its occurrences with $\top$; for eliminating boolean constants and so on. What could be the gain?

[2]With modal connectives or variables it is not so simple, but this is an intuitive explanation.

Intuition. We trade off some polynomial processing for an exponential gain: scanning the formula and looking for "exact" copies can be done in polynomial time, while the reduction in size can lead to exponentially shorter proofs.

The key observation is that the sequent proof of $\Gamma \Rightarrow \Delta$ may be exponential in the size of $O(2^{|\Gamma \cup \Delta|})$ [171]. If we reduce it to something smaller, if only by 1, we would reduce the potential search space at least by half.

Hence, if there is at least another occurrence of A (or B) in Γ, it always pays off to replace this occurrence with $\top$: the resulting sequent will be smaller and thus the proof tree will also be (exponentially) smaller.

We would like to follow some general principles:

- simplification should be *locally applicable* to single formulae;
- it must be a *sound and complete* admissible rules for the underlying tableau calculus;
- it should be substantially a *rewrite rule*;
- the time *complexity must be polynomial* in the size of the formulae involved in the rule;
- after simplification, the resulting formulae are (potentially) smaller.

Soundness is an obvious requirement. We need completeness only if we want to give precedence to simplification over other rules. If we waive this requirement we obtain an "unsafe" rule [141] i.e. a rule whose application may prevent the finding of a proof: it is still applicable but we may have to backtrack.

Local applicability is also a key property and leads to rules like "simplify B using A". The advantage of this kind of rules is that when we 'upgrade' the logic we do not change the rules but only specify the behavior of the operation for the *new* connectives we have introduced. Such flexibility and modularity make it easy to upgrade current implementations with these techniques.

The requirement of being a rewrite rule is mostly for ease of implementation. Almost all interactive (and automatic) provers have fairly developed procedures for rewriting and we would like to capitalize on these facilities.

2.2 Syntax and Semantics

The formulae A, B, C are constructed from a set of propositional atoms $p, q \in \mathcal{P}rop$ and the boolean constant $\top$ and $\bot$ with the logical connectives $\wedge$, $\vee$, $\neg$, $\supset$ etc. We use signed formulae and the α, β notation introduced by Smullyan [164]. Thus a *signed formula*, denoted by φ and ψ, is a pair $\mathtt{t}.A$ or $\mathtt{f}.A$. The intuitive interpretation is that A is assumed to be respectively true and false.

α	α_1	α_2
$\mathtt{t}.A \wedge B$	$\mathtt{t}.A$	$\mathtt{t}.B$
$\mathtt{f}.A \vee B$	$\mathtt{f}.A$	$\mathtt{f}.B$
$\mathtt{f}.A \supset B$	$\mathtt{t}.A$	$\mathtt{f}.B$

neg	*pos*
$\mathtt{t}.\neg A$	$\mathtt{f}.A$
$\mathtt{f}.\neg A$	$\mathtt{t}.A$

β	β_1	β_2
$\mathtt{f}.A \wedge B$	$\mathtt{f}.A$	$\mathtt{f}.B$
$\mathtt{t}.A \vee B$	$\mathtt{t}.A$	$\mathtt{t}.B$
$\mathtt{t}.A \supset B$	$\mathtt{f}.A$	$\mathtt{t}.B$

Table 2.1: Uniform Notation

The *conjugate* of a signed formula φ, denoted by $\overline{\varphi}$, is obtained by switching `t.` with `f.` and vice versa. Notice that $\overline{\overline{\varphi}} = \varphi$.

$$
\begin{array}{lcl}
A, B & ::= & \top \mid \bot \mid p \mid \neg A \mid A \wedge B \mid A \vee B \mid A \supset B \\
\varphi, \psi & ::= & \mathtt{t}.A \mid \mathtt{f}.A
\end{array}
$$

Since $\vee$ and $\wedge$ are commutative and associative we occasionally drop parentheses in the representation of a formula.

The uniform classification of formulae is given in Table 2.1. Negation has been classified in a class of its own.

The notion of *subformula closure* of a signed formula φ is defined as follows:

- $\varphi \in \mathrm{Subfml}(\varphi)$;
- if *neg* $\in \mathrm{Subfml}(\varphi)$ then *pos* $\in \mathrm{Subfml}(\varphi)$;
- if $\alpha \in \mathrm{Subfml}(\varphi)$ then $\alpha_1, \alpha_2 \in \mathrm{Subfml}(\varphi)$;
- if $\beta \in \mathrm{Subfml}(\varphi)$ then $\beta_1, \beta_2 \in \mathrm{Subfml}(\varphi)$.

We refer to the *atoms occuring in a formula* φ as the set

$$
\mathrm{vars}(\varphi) \quad = \quad \{\, p | \mathtt{t}.p \in \mathrm{Subfml}(\varphi) \text{ or } \mathtt{f}.p \in \mathrm{Subfml}(\varphi) \,\} \tag{2.1}
$$

This notion is extended to unsigned formula A as $\mathrm{vars}(\mathtt{t}.A)$.

One of the interesting features of the signed version is that conjugation is propagated uniformly down to the subformula level: the conjugates of the subformulae of φ are the subformulae of the conjugate of φ.

Proposition 2.1 *Let φ be a signed formula, then*

- *φ is of type α iff $\overline{\varphi}$ is of type β;*
- $\overline{(\alpha_i)} = (\overline{\alpha})_i$ *and* $\overline{(\beta_i)} = \left(\overline{\beta}\right)_i$.

The proof is by induction on the construction of φ.

In some cases we use formulae in *clausal normal form*. The standard definition of a *clause* is a multi set of *literals* i.e. propositional atoms, possibly negated. The intuition is that a clause is the disjunction of its literals.

With our general notation we can say that a clause is a β formula such that each β_i is either a signed (possibly negated) atom or another β formula.

2.2.1 Semantics

A PC-*model* $\mathcal{I}$ in propositional logic is a function $()^{\mathcal{I}}$, called *valuation*, from propositional letters to $\{\top, \bot\}$. Intuitively $(p)^{\mathcal{I}} = \top$ implies that p is true. If $\mathcal{I}$ is a model, the forcing relation $\mathcal{I} \Vdash A$ is defined as follows:

$$\begin{array}{lcl} \mathcal{I} \Vdash \top & \text{and} & \mathcal{I} \nVdash \bot \\ \mathcal{I} \Vdash p & \text{iff} & (p)^{\mathcal{I}} = \top \\ \mathcal{I} \Vdash A \wedge B & \text{iff} & \mathcal{I} \Vdash A \text{ and } \mathcal{I} \Vdash B \\ \mathcal{I} \Vdash \neg A & \text{iff} & \mathcal{I} \nVdash A \end{array}$$

The notion of forcing is extended to signed formulae as follows:

$$\mathcal{I} \Vdash \mathtt{t}.A \quad \text{iff} \quad \mathcal{I} \Vdash A \tag{2.2}$$

$$\mathcal{I} \Vdash \mathtt{f}.A \quad \text{iff} \quad \mathcal{I} \nVdash A \tag{2.3}$$

With this definition it is easy to prove by induction [163] that

$$\begin{array}{lcl} \mathcal{I} \nVdash \mathtt{f}.\top & \text{and} & \mathcal{I} \nVdash \mathtt{t}.\bot \\ \mathcal{I} \Vdash \mathtt{t}.\top & \text{and} & \mathcal{I} \Vdash \mathtt{f}.\bot \\ \mathcal{I} \Vdash neg & \text{iff} & \mathcal{I} \Vdash pos \\ \mathcal{I} \Vdash \alpha & \text{iff} & \mathcal{I} \Vdash \alpha_1 \text{ and } \mathcal{I} \Vdash \alpha_2 \\ \mathcal{I} \Vdash \beta & \text{iff} & \mathcal{I} \Vdash \beta_1 \text{ or } \mathcal{I} \Vdash \beta_2 \end{array}$$

Remark. In the sequel we only refer to the signed version of forcing.

When $\mathcal{I} \Vdash \varphi$ holds we say that $\mathcal{I}$ *satisfies* φ. For unsigned formulae, we say that $\mathcal{I}$ is a *model* for A if $\mathcal{I} \Vdash \mathtt{t}.A$ and it is a *countermodel* for A if $\mathcal{I} \Vdash \mathtt{f}.A$. If S is a set of formulae then $\mathcal{I}$ satisfies S iff for every $A \in S$ one has $\mathcal{I} \Vdash \mathtt{t}.A$.

For logical consequence we need a set of formulae U, the assumptions:

Definition 2.2 *A formula A is a* PC-logical consequence *of assumptions U (in symbols $\models_{\mathsf{PC}} U \Rightarrow A$) iff every* PC-*model which satisfies U is also a* PC-*model for A.*

When $U = \emptyset$, we abbreviate $\models_{\mathsf{PC}} \emptyset \Rightarrow A$ with $\models_{\mathsf{PC}} A$ and say that A is *valid*. Notice that U needs not be finite, although in practical case it is finite.

Remark. A (classically) equivalent definition can be formulated by saying that every model for U is not a countermodel for A.

Definition 2.3 *A formula A is a* PC-satisfiable *for assumptions U iff there is a* PC*-model which satisfies $U \cup \{ A \}$.*

A set of assumptions U is PC-*consistent* if $\top$ is satisfiable for U.

2.3 Tableaux with Simplification

The following definition of tableau is substantially taken from Smullyan [163] with a new rule for simplification (constraint propagation).

2.3.1 The basic tableau calculus

A *tableau* $\mathcal{T}$ is a (binary) tree whose nodes are labelled with signed formulae. A *branch* $\mathcal{B}$ is a path from the root to a leaf.

A tableau for a formula φ is constructed with the rules in Fig. 2.1 by labelling the root with φ and extending it as follows:

- if some *neg* formula occurs in the branch $\mathcal{B}$, then we can add a node at the end of branch labelled with *pos*;
- if some α formula occurs in the branch $\mathcal{B}$, then we can add a node at the end of branch labelled with α_1 and then add, to the resulting branch, a new node labelled with α_2;
- if some β occurs in $\mathcal{B}$, then we can split the branch adding a right and left successor at the end, one labelled with β_1 and one labelled with β_2;
- if $A \in U$ then we can add a node at the end of branch labelled with $\mathtt{t}.A$.

In the sequel we say that we extend the branch with φ as a shortcut for "extend the branch with a node labelled by φ".

After we have applied a rule to a formula, we say that the formula is reduced in the resulting branch(es). Formally,

Definition 2.4 *A formula ψ is* reduced *in a branch $\mathcal{B}$ if*

- *ψ is an α formula and both α_1 and α_2 are present in $\mathcal{B}$;*
- *ψ is a neg formula and pos is present in $\mathcal{B}$;*

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}}\,(\alpha) \qquad \frac{\beta}{\beta_1 \qquad \beta_2}\,(\beta) \qquad \frac{neg}{pos}\,(dneg) \qquad \frac{\vdots}{\mathtt{t}.A}\,(Loc) \quad A \in U$$

Figure 2.1: Basic Propositional Rules

- *ψ is a β formula and either β_1 or β_2 is present in $\mathcal{B}$.*

To avoid pointless reductions we always apply the following technique:

Technique 2.1 *Apply a rule to a formula φ in $\mathcal{B}$ only if φ is not reduced.*

The definition of proof is standard [63, 163]:

Definition 2.5 *A branch $\mathcal{B}$ is* contradictory *if $\mathtt{t}.\bot \in \mathcal{B}$, or $\mathtt{f}.\top \in \mathcal{B}$, or there is a formula A such that both $\mathtt{t}.A \in \mathcal{B}$ and $\mathtt{f}.A \in \mathcal{B}$. A tableau is closed if every branch is contradictory.*

Of course there is no sense in reducing formulae of a branch after we have proved that it is inconsistent and therefore we use the following technique:

Technique 2.2 *Apply a rule to a formula φ in $\mathcal{B}$ only if $\mathcal{B}$ is not contradictory.*

Definition 2.6 *A* tableau proof *for the formula A with assumptions U is the closed tableau for $\mathtt{f}.A$ with assumptions U.*

A branch is *completed* for the set U if for every $A \in U$ it is $\mathtt{t}.A \in \mathcal{B}$. A branch is *open* for the set of assumption U if it is completed for U, it is not contradictory and every formula in $\mathcal{B}$ is reduced. From an open branch we can construct a model $\mathcal{I}$ for all formulae in the branch as follows:

$$(p)^{\mathcal{I}} \quad = \quad \top \text{ iff } \mathtt{t}.p \in \mathcal{B}$$

If the corresponding tableau was a tableau for $\mathtt{f}.A$, an open branch is a counter model for A.

We state soundness and correctness and refer to [63, 163] for the proof.

Theorem 2.7 *The formula A is a logical consequence of assumptions U iff A has a tableau proof with assumptions U.*

$$\frac{\varphi}{\vdots}\,(thin) \qquad \frac{\vdots}{\varphi \quad \overline{\varphi}}\,(cut) \qquad \frac{\varphi}{\varphi[\psi]}\,(simp) \quad \text{where } \psi \text{ occurs on the branch}$$

Figure 2.2: Propositional Sequent Rules with Simplification

2.3.2 Simplification for constraint propagation

The calculus presented above is not very popular for its efficiency. We can enhance it with three possible rules: *cut*, *thinning*, and *simplification*. They are shown in Fig. 2.2.

We use the rule of (*cut*) as a β-rule. We can restrict its use as *analytic cut* following Smullyan [163, 164]: φ must be a formula in Subfml($\{\, \mathtt{t}.B | b \in U \,\} \cup \{\, \mathtt{f}.A \,\}$). The (*cut*)-rule can be eliminated without affecting completeness and soundness although it can affect the complexity of the proofs [171].

Rule (*thin*) is used to "delete" a formula from a branch: we consider it reduced even if other rules are applicable to it. It is clear that this rule can hamper the completeness of the proof procedure. We introduce cut and thinning only to show that some particular forms of simplification are obtained by combining them with simplification.

The rule of simplification is used in the following way: if both formulae φ and ψ are present in a branch, then we apply the operation $\psi[\varphi]$ adding the resulting formula to the branch. The formula ψ is now considered as *reduced* in the branch. Obviously, if $\psi[\varphi] = \psi$ then we do not re-apply again *simp* but try some other rules on ψ

The operation of simplification $\varphi[\psi]$ is substantially a *rewriting procedure*:

$$\varphi[\psi] \mapsto_{simp} \varphi^{*} \mapsto_{bool} \varphi^{**}$$

In the first stage we replace some subformulae of φ with $\top$ and $\bot$ and in the second stage we *eliminate the propositional constant* $\top$ *and* $\bot$ using standard boolean reductions. For instance:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\neg\top & \mapsto_{bool} & \bot & \neg\bot & \mapsto_{bool} & \top \\
A \wedge \top & \mapsto_{bool} & A & A \wedge \bot & \mapsto_{bool} & \bot \\
A \vee \top & \mapsto_{bool} & \top & A \vee \bot & \mapsto_{bool} & A \\
A \supset \top & \mapsto_{bool} & \top & A \supset \bot & \mapsto_{bool} & \neg A
\end{array}$$

The operation $\mapsto_{simp}$ will leave the sign of ψ unchanged and we only need to specify the progression over formulae.

$$A[\psi] \quad \mapsto \quad \begin{cases} \top & \text{if } \psi = \mathtt{t}.A \\ \bot & \text{elseif } \psi = \mathtt{f}.A \\ \neg(B[\psi]) & \text{elseif } A = \neg B \\ B[\psi] \wedge C[\psi] & \text{elseif } A = B \wedge C \\ B[\psi] \vee C[\psi] & \text{elseif } A = B \vee C \\ B[\psi] \supset C[\psi] & \text{elseif } A = B \supset C \\ A & \text{otherwise} \end{cases} \tag{2.4}$$

By $A = B$ we mean, at this preliminary stage, syntactic identity. So $A \wedge B$ is different from $B \wedge A$. Further extensions are described in the following subsections, where the commutative properties of the connectives are considered.

Signed formulae make rule (*simp*) extremely compact. Moreover,

1. we do not only reduce branching, we get smaller and simpler formulae
2. the operation of simplification is parametric wrt the connectives used;
3. the simplifying formula ψ needs not be a literal nor have any special normal form;
4. without restrictions, simplification can proceed recursively down any level of nesting of propositional connectives.

The last step is one of the key differences with modus ponens and tollens applied in `HARP` [139], the `KE` β^c-rules [35] and the $\vee - simp_{0,1}$-rule used in `KRIS` [7]. Indeed we show that they are particular forms of simplification where the recursion is stopped at the main connective of φ.

Since we are limited to syntactic search (i.e. replacement of regular expressions) it is immediate to prove the following property:

Proposition 2.8 *The operation $\varphi[\psi]$ can be performed in polynomial time in the size of ψ and φ.*

Definition 2.9 *A tableau rule is* invertible *iff whenever a proof can be found proof before the application of the rule, a proof also be found after the application of the rule.*

Intuitively the application of the rule does not prevent the finding of a proof. By induction on $\mapsto$ we can prove the following results:

Theorem 2.10 *Simplification (simp) is a sound and invertible rule for propositional logic.*

$$\frac{\mathcal{B}, \mathtt{f}.\top}{}\,(\top) \qquad \frac{\mathcal{B}, \mathtt{t}.\bot}{}\,(\bot) \qquad \frac{\mathcal{B}, \varphi, \overline{\varphi}}{}\,(Ax)$$

$$\frac{\mathcal{B}, \alpha}{\mathcal{B}, \alpha_1, \alpha_2}\,(\alpha) \qquad \frac{\mathcal{B}, \beta}{\mathcal{B}, \beta_1 \quad \mathcal{B}, \beta_2}\,(\beta) \qquad \frac{\mathcal{B}, neg}{\mathcal{B}, pos}\,(dneg)$$

$$\frac{\mathcal{B}, \psi, \varphi}{\mathcal{B}, \psi[\varphi], \varphi}\,(simp) \qquad \frac{\mathcal{B}}{\mathcal{B}, \varphi \quad \mathcal{B}, \overline{\varphi}}\,(cut) \qquad \frac{\mathcal{B}, \varphi}{\mathcal{B}}\,(thin)$$

Figure 2.3: A Gentzen Calculus with Simplification

Proposition 2.11 *The size of $\varphi[\psi]$ is never larger than the size of φ. It is strictly smaller whenever $\psi \in Subfml(\varphi)$ or $\overline{\psi} \in Subfml(\varphi)$.*

In the section on approximation we will also make an essential use of the following property, easily proved by induction:

Proposition 2.12 *If ψ has the form $\mathtt{t}.p$ or $\mathtt{f}.p$ then $p \notin vars(\varphi[\psi])$.*

This property does *not* hold if ψ is a formula. A simple example can be found by setting $\varphi = \mathtt{t}.(A \wedge (A \supset B)) \supset B$ and $\psi = \mathtt{t}.A \supset B$. It can be easily verified that $\psi \in \mathrm{Subfml}(\varphi[\psi])$.

2.3.3 A Gentzen-type formulation

It is very easy, with signed formulae, to pass from the tree-like version of tableau to the traditional Gentzen formulation of the sequent calculus for propositional logic. This is shown in Fig. 2.3 where $\mathcal{B}$ denotes the set of signed formulae along a branch.

The "classical formulation" of the Gentzen calculus [70, 63, 163] uses sequents, i.e. pairs of the form $\Gamma \Rightarrow \Delta$ where Γ and Δ are multisets of propositional formulae. Sequents can be reconstructed from the above calculi by setting $\Gamma_{\mathcal{B}} = \{\, A \mid \mathtt{t}.A \in \mathcal{B} \,\}$ and $\Delta_{\mathcal{B}} = \{\, A \mid \mathtt{f}.A \in \mathcal{B} \,\}$.

2.3.4 Further forms of simplification

We can devise more powerful forms if we replace $\wedge$ and $\vee$, which are commutative, associative and idempotent, with set-oriented versions $\bigwedge\!\!\!\bigwedge$ and $\bigvee\!\!\!\bigvee$. For instance the formula $(A_1 \vee A_2 \vee B_1) \wedge B_2 \wedge B_2 \wedge C$ can be transformed in the formula $\bigwedge\!\!\!\bigwedge \{B_2, C, \bigvee\!\!\!\bigvee \{A_1, A_2, B_1\}\}$. This is already a simplification since we eliminate duplicates.

Let $\otimes$ be a set oriented connective and S a be a set of formulae, so that $\otimes S$ is a 'set-oriented' formula. *Set-oriented simplification* is characterized as follows:

$$\otimes\{A|A \in S\}[\mathtt{t}.\otimes\{A \mid A \in S'\}] \mapsto \otimes\{A|A \in S \setminus S'\} \otimes \top$$
$$\otimes\{A|A \in S\}[\mathtt{f}.\otimes\{A \mid A \in S'\}] \mapsto \otimes\{A|A \in S \setminus S'\} \otimes \bot$$

where $S' \subset S$. If $S = S'$ we can just replace the initial formula by $\top$ or $\bot$ as with traditional syntactic simplification. We can also recursively simplify the remaining part.

For instance $\bigwedge\{A, B, \bigvee\{C, \neg D\}, \bigvee\{E, F\}\}$ simplified by the formula $\mathtt{t}.\wedge \wedge\{B, \bigvee\{E, F\}\}$ yields $\bigwedge\{A, \bigvee\{C, \neg D\}\} \wedge \top$ where $\top$ can then be eliminated.

Theorem 2.13 *Set-oriented simplification is a sound and invertible rule for propositional logic if $\otimes$ is associative, commutative and idempotent.*

It seems that we have obtained every form of syntactic simplification but it is not yet the case. Consider the rule for branching in sequent or tableau:

$$\frac{\mathcal{B}, \mathtt{t}.A \vee B}{\mathcal{B}, \mathtt{t}.A \qquad \mathcal{B}, \mathtt{t}.B}$$

Suppose that we have closed the branch(es) on the left-hand side. So, there is no model $\mathcal{I}$ which satisfies $\mathcal{B} \cup \{\mathtt{t}.A\}$ by the soundness theorem. Therefore, every $\mathcal{I}$ which satisfies $\mathcal{B}$ (if it exists) must satisfy $\mathtt{f}.A$. Among such models there are the models which satisfies $\mathcal{B} \cup \{\mathtt{t}.B\}$.

So we could simply add $\mathtt{f}.A$ on the right branch. This is what is done with *full lemmaizing*. The problem with this addition is that the resulting branch $\mathcal{B} \cup \{\mathtt{f}.A\} \cup \{\mathtt{t}.B\}$ has more formulae than $\mathcal{B} \cup \{\mathtt{t}.B\}$ and therefore has a potentially larger search space[3]. Using full fledged lemmata may yield some slow down, as it is shown by the comparison between `KE` and `leanTAP` by Pitt & Cunningham [143]. If the application of lemmas is restricted to unit clauses as done by Letz et al. [103] the performance is always improved.

The solution is *Tree Simplification* which replaces the previous branching rule:

$$\frac{\mathcal{B}, \beta}{\mathcal{B}, \beta_1 \qquad \mathcal{B}[\overline{\beta_1}], \beta_2[\overline{\beta_1}]}\, Tree - \beta$$

By $\mathcal{B}[\varphi]$ we indicate the simplification of *all* formulae of the branch.

The idea is close to *unit lemmas* [103], and the split rule of `DPLL` or the *principle of bivalence* employed by `KE`[35]. Indeed we can represent this technique as a derived rule in presence of (cut) and $(thin)$:

[3]Recall that in the worst case the tableau below the right branch could be exponential in the size of $\mathcal{B} \cup \{\mathtt{f}.A\} \cup \{\mathtt{t}.B\}$.

$$\cfrac{\mathcal{B},\beta}{\mathcal{B},\beta_1 \qquad \cfrac{\mathcal{B},\beta_2}{\cfrac{\mathcal{B},\beta_2,\beta_1}{\mathcal{B},\beta_1}\,thin \qquad \cfrac{\cfrac{\mathcal{B},\beta_2,\overline{\beta_1}}{\mathcal{B}[\overline{\beta_1}],\beta_2[\overline{\beta_1}],\overline{\beta_1}}\,simp_{\overline{\beta_1}}}{\mathcal{B}[\overline{\beta_1}],\beta_2[\overline{\beta_1}]}\,thin}\,cut}\,\beta$$

The last thinning step at the right-hand branches hinder completeness so we can only state a weaker property for tree simplification:

Theorem 2.14 *Tree simplification is a sound rule for propositional logic.*

Theorem 2.15 *Tree simplification is an invertible rule for propositional logic if β_1 is a signed propositional atom.*

The reason for the successes of the split-rule of `DPLL` or the folding operation with unit lemmas in `SETHEO` [103] is indeed their combination with (hidden) unit simplification: we add a unit literal (the negation of the left-hand side) and afterwards we proceed with unit simplification. The size of the right subtree is never larger.

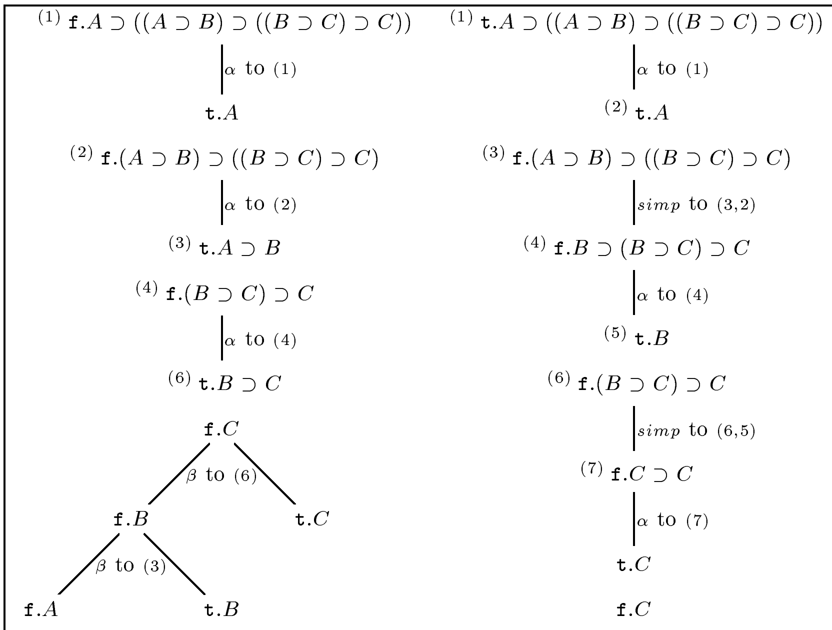
2.4 Examples

General simplification has further gains wrt `HARP` [139] or `KE` [35] or calculi with factoring. Proof which would be optimal for the above mentioned calculi, can still be (considerably) shortened.

The difference between a traditional derivation and a derivation using simplification is shown in Fig. 2.4. The branching factor is greatly reduced. If we carry the same proof using modus ponens as in `HARP` or the β^c-rules of `KE` we get the same branching structure of the simplified proof but the size of the simplified formulae is larger.

Simplification is superior to modus ponens or β^c-rules when there are formulae which are logically equivalent (but not identical) to the pattern used by MP or β^c-rules, and when there are many occurrences of the simplifying formula, possibly deeply nested within the main formulae of the sequent. An example is shown in Fig 2.5. To simplify the notation, we assume associativity and the usual precedence of $\wedge$, $\vee$, and $\supset$.

One may argue that the pattern $(A \supset B) \vee (A \supset C)$ could be recognized but it is easy to change it in $(A \supset B) \vee ((A \wedge \neg B) \vee D \supset C)$. This formula can still be proved with local simplification without any branching whereas `HARP`'s proof is harder than the one shown in Fig. 2.5. Moreover the point of simplification is that we do not want to remember a lot of patterns.



In the left proof we must apply both β and α rules. In the right proof we marked the steps relative to simplification (first wrt $\mathtt{t}.A$ and then wrt $\mathtt{t}.B$). There is no need of performing a case analysis with β-rules.

Figure 2.4: Classical Sequents Versus Simplification

2.5 Subsuming Other Approaches

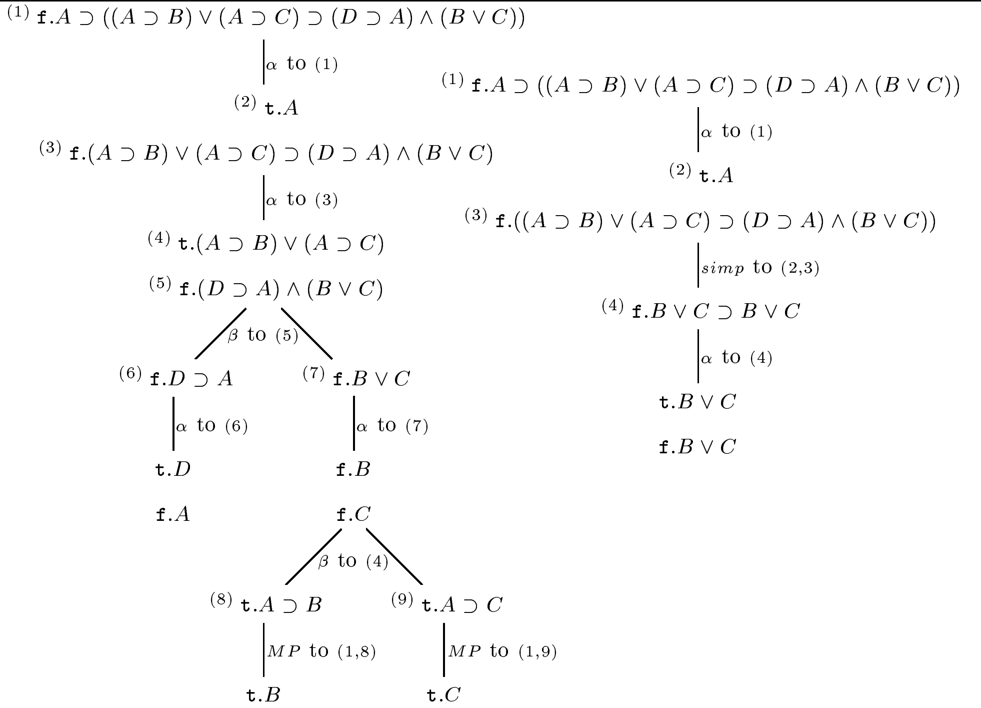
Many rules used by tableau provers are restricted and shallow forms of local simplification. Here we consider the case for the unit of `DPLL` [41] and Boolean Constraint Propagation [123], the β^c-rules of `KE` [35], the modus ponens and tollens of `HARP` [139], hyper-tableaux expansions [8].

2.5.1 Davis-Putnam procedure and Boolean Constraint Propagation

First consider the original definition of the `DPLL` unit rule: "let l be a unit clause, delete every clause which contains l and delete any occurrence of $\overline{l}$ from the remaining clauses" [42]. We recall that $\overline{\neg p} = p$ and $\overline{p} = \neg p$.

Let $\mathcal{B} \cup \{\, \mathtt{t}.l \,\}$ be a set of clauses. We proceed as follows:

- apply *dneg* to $\mathtt{t}.l$, if possible, and let φ be the resulting signed atom;
- apply simplification to every clause $\psi \in \mathcal{B}$ with φ.



In both cases we apply MP and simplification as soon as possible, then α-rules, and only when no other rule is available we apply β-rules.

Figure 2.5: HARP Modus Ponens Versus Simplification

Now suppose that l occurs in the clause, i.e. $\psi = \mathtt{t}.l \vee l_1 \vee l_2 \vee \ldots \vee l_n$. Clearly $l[\varphi] \mapsto \top$ and by applying the boolean reduction $\top \vee l_1 \vee \ldots l_n \mapsto \top$. Then $\mathtt{t}.\top$ needs not to be processed further and we consider ψ as reduced (and we could as well delete it). If $\bar{l}$ occurs, we have $\bot \vee l_1 \vee \ldots l_n \mapsto l_1 \vee \ldots \vee l_n$ i.e. the same result of the unit rule.

Fact 2.16 *The application of* DPLL*unit rule is an instance of a sequence of tableaux rules dneg and (simp) to formulae in clausal normal form.*

We know that DPLL-unit corresponds to *boolean constraint propagation* BCP [123, 46, 37] so we also have the following result:

Proposition 2.17 *The clause A is a logical consequence of a set of clauses U with boolean constraint propagation iff there is a tableau proof with only dneg and simp (restricted to literals) as rules of inference.*

We can extend these result to the general case of supsumption [148, 42]:

Fact 2.18 DPLL-*subsumption is an instance of simplification with set-oriented connectives.*

Tree simplification "compiles" these ideas into a direct and uniform rule without a restriction to unit lemmas:

Fact 2.19 *The* DPLL *procedure is an instance of the tableaux calculus with local and tree simplification (restricted to literals) and formulae in clausal form.*

If we do not like the use of tree simplification we can simply use *cut*:

Theorem 2.20 *The clause A is a logical consequence of a set of clauses U using the Davis-Putnam procedure iff there is a tableau proof with* (*dneg*)*,* (*simp*)*, and* (*cut*) *restricted to literals as the only rules of inference.*

It has been sometimes argued that the use of the name TABLEAU in [31] was somehow misleading since it was an efficient implementation of DPLL. In the light of this work, the choice of the name was indeed correct.

2.5.2 Hyper-tableaux

Hyper-tableaux by Baugarten et al. [8] are an hybrid deduction mechanism incorporating unit resolution and clausal tableau expansion.

The basic inference mechanism (at the propositional level) is the following:

$$\frac{\mathtt{t}.\neg q_1 \vee \ldots \vee \neg q_m \vee p_{m+1} \vee \ldots \vee p_n}{\mathtt{t}.p_{m+1} \quad \mathtt{t}.p_{m+2} \quad \cdots \quad \mathtt{t}.p_n} \text{ if all } \mathtt{t}.q_i \text{ are present in the branch}$$

Intuitively, we apply a clausal tableau expansion of the positive parts of β if we can find a unit resolution refutation of the negative part of the clause.

It is easy to show that the extension step of hyper-tableaux [8] can also be subsumed. We need only a preliminary definition: a formula φ is *positive* if $\mathtt{f}.p \notin \mathrm{Subfml}(\varphi)$ for all p. If φ is a clause we say that it is positive clause.

Theorem 2.21 *The clause A has a propositional hyper-tableaux proof for the set of clauses U iff there is a tableau proof using rule* (*simp*) *restricted to literals and rule* (β) *restricted to positive clauses as the only rules of inference.*

2.5.3 KE, HARP, and (propositional) KRIS

Many provers and calculi incorporate ad-hoc mechanism for dealing with limited form of simplification. The simplest example is the tableau prover HARP by Oppacher & Suen [139] which incorporated simple logical rules such as modus pones and tollens into the search strategy:

$$\frac{\mathtt{t}.A \supset B \quad \mathtt{t}.A}{\mathtt{t}.B} MP \qquad\qquad \frac{\mathtt{t}.A \supset B \quad \mathtt{f}.B}{\mathtt{f}.A} MT$$

Similar rules have been also used for disjunction in `KRIS` [7].

$$\frac{\mathtt{t}.A \vee B \quad \mathtt{f}.A}{\mathtt{t}.B} \vee - simp_0 \qquad\qquad \frac{\mathtt{t}.A \vee B \quad \mathtt{f}.B}{\mathtt{t}.A} \vee - simp_1$$

The β^c-rule of `KE` from [35] are a generalization to signed formulae of these rules. They are as follows:

$$\frac{\beta \quad \overline{\beta_1}}{\beta_2}\,\beta_1^c \qquad\qquad \frac{\beta \quad \overline{\beta_2}}{\beta_1}\,\beta_2^c$$

It can be shown, by considering all possible cases, that they are extremely restricted form of local simplification. Consider the following instance of β_1^c:

$$\begin{array}{cc} \mathtt{KE} - \text{rule} & simp - \text{rule} \\ \dfrac{\mathtt{t}.A \vee B \qquad \mathtt{f}.A}{\mathtt{t}.B}\,\mathtt{t}.A^c & \qquad \dfrac{\mathtt{t}.A \vee B \qquad \mathtt{f}.A}{\mathtt{t}.(A \vee B)[\mathtt{f}.A]}\,simp_{\mathtt{f}.A} \end{array}$$

If we apply the full-fledged version of simplification we would have had:

$$(A \vee B)[\mathtt{f}.A] \quad \mapsto \quad A[\mathtt{f}.A] \vee B[\mathtt{f}.A] \quad \mapsto \quad \bot \vee B[\mathtt{f}.A] \quad \mapsto \quad B[\mathtt{f}.A]$$

Thus, while `KE` only yields B, the full fledged use of simplification would have given us $\mathtt{t}.B[\mathtt{f}.A]$. If A occurred somewhere in B, then simplification would have reduced B and shortened the proof further.

Fact 2.22 *The β_i^c rules for* `KE` *are instances of the simplification rule* ($simp$) *restricted to β-rules and where recursive simplification is stopped at the main connective.*

After all `KE` by D'Agostino and Mondadori [35] is not an entirely new system:

Theorem 2.23 *The formula A has a proof for the assumption U for the system* `KE` *iff it has a tableau proof where dneg, α, simp and analytic cut are the only rules of inference (with the above mentioned restrictions).*

As a simpler corollary we get:

Fact 2.24 *Modus ponens, tollens etc. of* `HARP` *and the $\vee - simp_{0,1}$ rules in* `KRIS` *are instances of the simplification rule* ($simp$) *only applied to formulae of particular shape and where recursive simplification is stopped at the main connective.*

2.5.4 Regular tableau proofs

The next restriction that is imposed on clausal tableau proofs or model elimination techniques is *regularity* [8, 35, 103, 171]: a literal should never appear twice on a branch. In the setting of propositional logic and clausal theorem proving (such as [8, 103]) this is equivalent to say that a clause is never selected for the extension of a branch in the tableau if the extension step (the n-ary β-rule we have shown before) will yield a literal to appear twice. It is a system to constrain the search space and in particular the non-determinism in the selection of clauses for reduction.

Since we use tableau, and are not restricted to clausal normal form we rephrase it in more general terms: a proof is *regular* iff all subformulae introduced by α, β rules (i.e. $\beta_1, \beta_2, \alpha_, \alpha_2$) do not occur already in the branch before the application of the rule.

This requirement is always satisfied (and hence made redundant) by calculi with simplification:

Theorem 2.25 *If simplification is given precedence over other rules then all propositional tableau proofs are regular.*

For example consider the α rule: if α_1 already occurs in the branch then, by giving precedence to simplification we have that $\alpha[\alpha_1] \mapsto \alpha_2$. Then we can reduce α without reintroducing α_1.

Proof condensation [139] is also made redundant in most cases. The only exception is when we have two (or more) disjunctive formulae β^a and β^b on the branch and only the second (more properly β_1^b and β_2^b) is necessary for the closure of the tableaux.

2.6 An Experimental Assessment

We performed some experiments on random 3-SAT instances and the IFIP benchmarks with a tableaux prover (`Beatrix`) in `sicstus prolog`.

`Beatrix` has been designed in the spirit of lean theorem proving [11] with few modifications. At first, we used set oriented connectives (e.g. prolog lists of the form `[conj,a1,a2,a3]` to represent $a_1 \wedge a_2 \wedge a_3$) because the benchmarks are easily expressed in this formalism. Second we imposed the trivial heuristic that small formulae are reduced before large ones. Finally, before applying an α or β rule to a formula φ we use φ to simplify all other formulae in the current branch. Formula are reduced off-line in negation normal form without any optimization.

To make the comparison more interesting we used two β-rules: the standard rule of Fig. 2.1 and the asymmetric rule for a limited form of cut [35, 103]:

$$\frac{\beta}{\beta_1 \quad \beta_2}\,Dir \qquad\qquad \frac{\beta}{\beta_1 \quad \begin{array}{c}\overline{\beta_1}\\ \beta_2\end{array}}\,Lem$$

A simple implementation is already sufficient for interesting conclusions wrt the relative gains of simplification and lemmaizing. In the following tables the normal β rule is denoted by "Dir" and the lemmaizing version is denoted by "Lem".

None of the problems listed here could be solved by the version of `Beatrix` without simplification (in practice an enhanced version of `leanTAP` [11]) within two hours (or more), *no matter the branching rule used.* The outcome of these experiments is that analytic cut alone (in the form of lemmaizing) does not help to solve very hard problems.

2.6.1 Random 3-CNF

Table 2.2 shows the result for the standard random distribution of 3-SAT[4]: $3-sat(V, C)$ means that samples had C clauses, with 3 literals selected uniformly among V variables and each literal negated with probability 0.5.

A number of experimental studies [31, 72] have shown that `DPLL` (and indeed most complete satisfiability checking procedures) exhibits an easy-hard-easy computation pattern as the ratio between the number of clauses and variables increases, with the hardest formulae around $C/V = 4.2 - 4.3$. This phenomenon has been sometimes attributed to the "semantical" branching due to the splitting rule of `DPLL`. On the contrary,

Fact 2.26 *For random 3-CNF simplification produces already the easy-hard-easy pattern. Only in the transition region the addition of lemmaizing performs substantially better.*

2.6.2 IFIP benchmarks for hardware verification

Beside CNF benchmark we have tested our implementation on the IFIP benchmarks[5] for hardware verification by van Claesen [27]. These benchmarks are hard for (clausal) tableau methods [170] and are not formulated in clausal form, with an extensive use of equivalences ($\equiv$) and exclusive or ($\oplus$). Each benchmark problem requires to show that two circuit descriptions are equivalent (i.e. a specification matches an implementation).

[4] After $C = 10 \times V$ a linear relation between C and running time is due to inefficient list management by `prolog`.

[5] A prolog and an `Isabelle` versions of the benchmark are available from the author.

Table 2.2: Benchmark on Random 3-SAT

Problem	Dir	Lem
3-sat(32,96)	.3	.2
3-sat(32,128)	3.9	1.2
3-sat(32,136)	6.1	1.8
3-sat(32,144)	6.9	2.1
3-sat(32,160)	8.2	2.4
3-sat(32,192)	7.7	2.6
3-sat(32,224)	6.0	2.2
3-sat(32,256)	5.3	2.3
3-sat(32,288)	4.9	2.3
3-sat(32,320)	4.8	2.4
3-sat(32,640)	5.9	3.7
3-sat(32,1280)	10.5	6.5

Problem	Dir	Lem
3-sat(64,192)	1.4	1.0
3-sat(64,256)	334.6	38.4
3-sat(64,272)	554.3	56.4
3-sat(64,288)	1,050.9	72.0
3-sat(64,320)	568.6	60.0
3-sat(64,384)	240.3	39.4
3-sat(64,448)	139.3	30.6
3-sat(64,512)	74.3	23.5
3-sat(64,576)	69.6	21.8
3-sat(64,640)	59.6	20.4
3-sat(64,1280)	38.2	19.8
3-sat(64,2560)	52.3	24.9

An easy benchmark is shown in Fig. 2.6 as a sequent to be proved. It is a simple 2-bit binary ripple adder: it takes four bits of input (a_i and b_i) and returns three bits of output: the sums som_i and the carry $cout$. We are given two circuits as two sets of equivalences, each equivalence defining the output of a sub-circuit. The set of formulae U_1 in Fig. 2.6 describes the abstract specification and the set U_2 describes the implementation. Using these descriptions as assumptions we must derive a proof of the equivalence of the respective outputs.

In practice all benchmarks have been solved except large multipliers, known to be hard also for BDDs, timed out after 10 minutes (for a comparison between `DPLL` and OBDDs see Uribe & Stickel [170]). Satisfiable formulae (marked by *) are solved within few milliseconds. What is more interesting here is the comparison with a standard interactive prover such as `Isabelle` [141]. This is shown in Table 2.3, where $fast\_tac$ is the standard tactic used in `Isabelle`, and $ifip\_tac$ applies $fast\_tac$ after a preprocessing step which exploits the format of the benchmark and eliminates most equivalences.

The scaling factor for problems such as $rip0x.be$ is more interesting than the absolute values. Absolute values simply tell us about the relative efficiency of lean proving in prolog vs big provers in ML, somehow strengthening the argument by Beckert & Possega [11]. On the contrary the gap in the scaling factor (a factor of 1000 for `Isabelle` in the passage from $rip0n$ to $rip0n+2$ is matched by a factor of 10) tells that the use of simplification can substantially increase the tractability threshold for tableau calculi. Further benchmarks are shown in Table 2.4 and in Table 2.5.

Intuition. Simplification is must for efficient theorem proving.

Table 2.3: Simplification versus `Isabelle`

Problem	fast_tac	ifip_tac
ex2	13.1	1.3
transp	43.8	0.2
risc	-	9.8
counter	-	68.8
hostint1	-	96.5
mul	-	130.9
rip02	2848.2	1.6
rip04	-	994.5
rip06	-	-
rip08	-	-

Problem	Dir	Lem
ex2	0.0	0.0
transp	0.0	0.0
risc	0.4	0.6
counter	0.1	0.1
hostint1	0.3	0.2
mul	0.4	0.2
rip02	0.0	0.0
rip04	0.5	0.6
rip06	3.0	3.0
rip08	18.2	18.4

Table 2.4: Run Times on IFIP Benchmarks

Problem	Dir	Lem
counter	0.1	0.2
d3(*)	0.1	0.1
dk27	2.2	2.3
ex2	0.0	0.0
hostint1	0.2	0.2
misg	0.7	1.0
mul	0.2	0.2
risc	0.4	0.5
transp	0.0	0.0

Problem	Dir	Lem
ztwaalf1	0.8	0.8
ztwaalf2	0.8	0.4
mjcg_no (*)	0.0	0.0
mjcg_yes	1.8	1.1
mp2d	3.7	1.1
rip02	0.0	0.0
rip04	0.4	0.5
rip06	2.9	2.9
rip08	18.2	18.4

Table 2.5: More IFIP Benchmarks

Problem	Dir	Lem
add1	24.4	12.2
alu	21.3	7.1
dk17	19.8	3.0
f51m	21.4	5.7
table	13.9	2.8
vg2	12.8	7.0
x1dn	13.4	7.2
z4	2.9	2.3
z5xpl	11.5	4.1

Problem	Dir	Lem
alupla20	784.0	618.1
dc2	149.7	12.5
sqn	297.7	11.2
z9sym	166.8	9.8
mul03	-	20.1
pitch	-	5.7
rd73	-	30.4
rom2	-	2.5
root	-	33.7

$$U_{spec} = \left\{ \begin{array}{l} car_1 \equiv (a_1 \wedge b_1), \\ som_{1,spec} \equiv (a_1 \oplus b_1), \\ som_{2,spec} \equiv (a_2 \oplus b_2 \oplus car_1), \\ cout_{spec} \equiv (((a_2 \vee b_2) \wedge car_1) \vee (a_2 \wedge b_2)), \end{array} \right. ,$$

$$U_{imp} = \left\{ \begin{array}{l} cout_1 \equiv (b_1 \wedge a_1)), \\ som_{1,imp} \equiv \neg((\neg a_1 \wedge \neg b_1) \vee (a_1 \wedge b_1)), \\ som_{2,imp} \equiv \neg((((\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)) \wedge \neg cout_1) \vee \\ \qquad\qquad \vee(cout_1 \wedge \neg((\neg a_2 \wedge \neg b_2) \vee (a_2 \wedge b_2)))), \\ cout_{imp} \equiv ((a_2 \wedge cout_1) \vee (b_2 \wedge cout_1) \vee (a_2 \wedge b_2)) \end{array} \right.$$

$$\begin{array}{rcl} U_{spec} \cup U_{imp} & \Rightarrow & (som_{1,spec} \equiv som_{1,imp}) \wedge \\ & & \wedge(som_{2,spec} \equiv som_{2,imp}) \wedge \\ & & \wedge(cout_{spec} \equiv cout_{imp}) \end{array}$$

Figure 2.6: A 2-bit Ripple Adder `rip02.be`

Chapter 3

Prefixed Modal Tableaux

We continue the analysis of logical systems with multi modal logics for knowledge and belief. We give an overview of the field of modal theorem proving and a quick introduction to the semantics of a wide range of modal logics.

We upgrade the previous calculus for propositional logic to the modal framework introducing the concept of prefixed tableau and incorporating the notion of modal constraints propagation. Also in this case we show how a number of features from traditional and advanced tableau methods are captured by our formalism.

3.1 An Overview of Modal Theorem Proving

The choice of deduction procedures for modal logics is almost as large as the number of modal logics[1]. Comprehensive overviews of deduction methods can be found in the book by Fitting [62], in Goré [79], and in Ohlbach [136].

At one side of the spectrum, *direct deduction* methods use modal formulae and enhance classical deduction mechanism with rules for modal connectives. For instance Hilbert axiomatizations can be found in [82, 89], Fitch- or Gentzen-type calculi and tableaux in [23, 62, 79, 128] and modal resolution in [54, 128]. Direct systems reflect in the rules themselves the epistemic properties (omniscience, introspection etc.) a user would like to model. However, they have computational limitations. For instance, Hilbert systems are not analytic and thus cannot be thoroughly automated. Gentzen-type calculi and tableaux are usually analytic [62, 79, 138]. Nevertheless, the reduction of model connectives requires processing sets of formulae at once (or whole tableau branches). The absence of a confluence theorem makes deduction heavily dependent on the search methods (see Demri [49]). Moreover, logical

[1] The books by Fagin et al. [57], and by Hughes & Cresswell [89] or the review by Halpern & Moses [82] are good introductions to modal logics.

consequence for symmetric and euclidean logics, which play a major role in non monotonic reasoning, is still unsatisfactory. Gentzen calculi treat these logics in an entirely non-deterministic way, since cut has been eliminated only for K45 by Schwartz [160]. For S5 the most effective way is still the explicit construction of Kripke models done by Hughes & Cresswell [89] and Halpern & Moses [82], or the (possibly exponential) reduction to a suitable normal form followed by a specialized calculus as done by Enjalbert & Fariñas del Cerro [54] or Ohnishi and Matsumoto [138]. With symmetric logics, the elimination of cut is still an open problem for logical consequence [62].

On the other side, *indirect deduction* methods translate modal formulae into first order logic, "add" the properties of the underlying Kripke model and apply a (suitably modified) classical deduction technique. A general overview of these techniques has been compiled by Ohlbach [136]. A similar approach has been proposed by D'Agostino et al. [34] using a simplified set theory as the target logic. Some translations, e.g. the relational translation plus theory reasoning proposed by Gent [69] and Russo [151], reflect the underlying semantics of Kripke models but are often inefficient. The use of set theory may also prove to be hard for automated reasoning. Functional translation performs much better and has been used with the matrix proof methods by Wallen [178], and with resolution by Ohlbach [135, 136]. This approach requires either a strong equality handling mechanism or the compilation of logic dependent unification procedures in the prover (to cope with the terms corresponding to possible worlds). To overcome this limitation, a mixed approach has been proposed by Nonnegart [133] capturing the most common logics with simple Horn clauses and maintaining the original structure of the formula (at least for deontic logics). A common limitation of these proposals is that they seem to need a "user-friendly" interface for proofs presentation (see [178] for further references). As we shall see, they also have computational drawbacks, although the presence of effective resolution implementations does not make this immediate.

In the middle of the field we find *labelled deduction* systems à la Gabbay [66], usually in the form of prefixed tableaux as done by Fitting [62, Chapter 8]. In prefixed tableaux, formulae are labelled with a prefix, to "name" the world where each formula is supposed to hold and rules take into account both formulae and labels. Thus, proofs and proof search presentations are simple, since they closely reflect the semantics of possible worlds. Yet, we need to understand and use an algebraic or a relational theory of prefixes.

A general remark can be made for validity checking with the logics of knowledge and belief K45, KD45 and S5: this problem is known to be co-NPtime-complete but the corresponding proof systems are often based on extension of the logics K or S4 by adding further rules in tableau systems [62, 79, 82, 160], by strengthening the equational constraints in Ohlbach [135]

$\nu(a)$	ν_0
$\texttt{t}.\,[a]\,A$	$\texttt{t}.A$
$\texttt{f}.\langle a\rangle A$	$\texttt{f}.A$

$\pi(a)$	π_0
$\texttt{f}.\,[a]\,A$	$\texttt{f}.A$
$\texttt{t}.\langle a\rangle A$	$\texttt{t}.A$

Table 3.1: Uniform notation

or by directly embedding S5 in S4 as done by Fitting [62] or Ohnishi and Matsumoto [137]. In other cases ad hoc procedures are used for formulae of modal degree one[2] [89, 138, 54] and a potentially[3] exponential preprocessing step is necessary to avoid cut and prove termination. Simpler in theory, they are harder in practice. It is not by chance that most systems focus on S4.

In this chapter we extend the propositional tableau to modal logic using a particular variant of prefixed tableau: *Single Step Tableaux*, SST for short enhanced with modal constraints propagation. This tableau calculus, introduced in [113], lies between Gentzen-type and prefixed tableaux and tries to combine user-friendly presentation with effective computational properties. It is worth noting that new versions of SST based on free variable tableaux have been currently implemented in prolog by Beckert & Gorè [10].

3.2 Propositional Modal Logics

We construct *modal formulae* A, B, C from propositional letters $p \in \mathcal{P}rop$ and agent names $a \in \mathcal{A}$ with the propositional connectives and the modal connectives $[\cdot]$, $\langle\cdot\rangle$ as follows:

$$A, B \;::=\; p \mid \neg A \mid A \wedge B \mid [a]\,A \mid \langle a\rangle A$$

Again, we use *signed modal formulae* using the uniform notation due to Fitting [62] and shown in Table 3.1. Formulae of the kind $\texttt{t}.\,[a]\,A$ are referred as $\nu(a)$ formulae (from necessity) whereas $\texttt{t}.\langle a\rangle A$ as $\pi(a)$ formulae (from possibility).

We extend the notion of *subformula closure* (§2.2) of a signed formula φ from boolean to modal connectives by adding two extra rules:

- if $\pi(a) \in \text{Subfml}(\varphi)$ then $\pi_0 \in \text{Subfml}(\varphi)$.
- if $\nu(a) \in \text{Subfml}(\varphi)$ then $\nu_0 \in \text{Subfml}(\varphi)$.

[2] No modal connective occurs under the scope of another modal connective.

[3] Unless one uses the non-trivial definitional translation into CNF of Mints [128].

Table 3.2: Axioms and Accessibility Relations

Axiom	Accessibility Relation
K : $[a]\,(A \supset B) \supset ([a]A \supset [a]B)$	any
D : $[a]\,A \supset \langle a \rangle A$	$\forall w \exists v : w\mathcal{R}_a v$
T : $[a]\,A \supset A$	$\forall w : w\mathcal{R}_a w$
4 : $[a]\,A \supset [a]\,[a]\,A$	$\forall w, v, u : w\mathcal{R}_a v \ \&\ v\mathcal{R}_a u \supset w\mathcal{R}_a u$
B : $\langle a \rangle\,[a]\,A \supset A$	$\forall w, v : w\mathcal{R}_a v \supset v\mathcal{R}_a w$
5 : $\neg\,[a]\,A \supset [a]\,\neg\,[a]\,A$	$\forall w, v, z : w\mathcal{R}_a v \ \&\ w\mathcal{R}_a z \supset v\mathcal{R}_a z$
Cxt: $\neg\,[a]\,[a]\,A \supset [a]\,\neg\,[a]\,A$	$\forall wuvz : w\mathcal{R}_a v \ \&\ w\mathcal{R}_a u \ \&\ v\mathcal{R}_a z \supset u\mathcal{R}_a z$

The semantics is based on *Kripke models* i.e. triples $\langle W, \{\, \mathcal{R}_a \,\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ where W is a non empty set, whose elements are called *worlds*, $\{\, \mathcal{R}_a \,\}_{a\in\mathcal{A}}$ is a family of binary *accessibility relation* over W, and $()^{\mathcal{I}}$ is function, called *valuation* from propositional letter to subset of worlds. Intuitively the worlds in $(p)^{\mathcal{I}} \subseteq W$ are those where p is true.

Different modal logics are obtained by different properties of $\mathcal{R}_a$. In Table 3.2 we list the most common properties and the corresponding axiom schemas. The combination of the axiom schemata from K to 5 (or the properties of the accessibility relation) generates the traditional logics K, KB, K4, K5, K45, KD, KDB, KD4, KD5, KD45, T, B, S4, S5, whereas Cxt is McCarthy's logic for contextual reasoning [21, 124]. Our aim is to provide a modular calculus for these logics and therefore we refer to L-models, L-tableaux etc. when referring to models, tableaux etc. for one of the logics L above. Other logics are also discussed by Massacci [113].

Notice that each agent may be characterized by a different set of axioms so we can have a $\mathsf{K45}_a, \mathsf{S4}_b$ logic where the reasoning of agent a is described by the axioms of logic K45 and the reasoning of agent b by the axioms of logic S4. To be precise we should speak of $\{\, \mathsf{L}_a \,\}_{a\in\mathcal{A}}$-models and so on. We avoid this notation for sake of readability.

The forcing relation generalizes the propositional forcing of section 2.2. If $\mathcal{M}$ is an L-model, w is a world and φ is a signed formula, the forcing relation $\mathcal{M}, w \Vdash \varphi$ is defined in the following way:

$$
\begin{array}{lll}
\mathcal{M}, w \Vdash \mathtt{t}.p & \text{iff} & w \in (p)^{\mathcal{I}} \\
\mathcal{M}, w \Vdash \mathtt{f}.p & \text{iff} & w \not\in (p)^{\mathcal{I}} \\
\mathcal{M}, w \Vdash \alpha & \text{iff} & \mathcal{M}, w \Vdash \alpha_1 \text{ and } \mathcal{M}, w \Vdash \alpha_2 \\
\mathcal{M}, w \Vdash \beta & \text{iff} & \mathcal{M}, w \Vdash \beta_1 \text{ or } \mathcal{M}, w \Vdash \beta_2 \\
\mathcal{M}, w \Vdash neg & \text{iff} & \mathcal{M}, w \Vdash pos \\
\mathcal{M}, w \Vdash \nu(a) & \text{iff} & \forall v \in W : w\mathcal{R}_a v \text{ implies } \mathcal{M}, v \Vdash \nu_0 \\
\mathcal{M}, w \Vdash \pi(a) & \text{iff} & \exists v \in W : w\mathcal{R}_a v \text{ and } \mathcal{M}, v \Vdash \pi_0
\end{array}
$$

When $\mathcal{M}, w \Vdash \varphi$ holds we say that *"w satisfies φ in $\mathcal{M}$"*; in particular, if φ has the form $\mathtt{t}.A$ ($\mathtt{f}.A$), we may say that "A is true (false) at w in $\mathcal{M}$". If S is a set of formulae then a world w satisfies S iff for all $A \in S$ one has $\mathcal{M}, w \Vdash \mathtt{t}.A$. We omit $\mathcal{M}$ whenever it is clear from the context. An L-model $\langle W, \{\mathcal{R}_a\}_{a \in \mathcal{A}}, \mathcal{I}\rangle$ *satisfies* S iff every world in W satisfies S. A formula A is L-*valid* if every L-model satisfies A.

For logical consequence we need two sets of modal formulae [64]: global assumptions G and local assumptions U. Intuitively the former are true in every world and the latter in the current world.

Definition 3.1 *A formula A is an* L-logical consequence *of global assumptions G and local assumptions U (in symbols $G \models_{\mathsf{L}} U \Rightarrow A$) iff in every L-model $\langle W, \{\mathcal{R}_a\}_{a \in \mathcal{A}}, \mathcal{I}\rangle$ which satisfies G if a world w satisfies U then w satisfies $\mathtt{t}.A$.*

If $\emptyset \models_{\mathsf{L}} \emptyset \Rightarrow A$ then A is L-valid. We say that the model $\langle W, \{\mathcal{R}_a\}_{a \in \mathcal{A}}, \mathcal{I}\rangle$ is a *counter model* for A if it satisfies G and there is world $w \in W$ which satisfies U and $\mathtt{f}.A$.

Definition 3.2 *A formula A is* L-satisfiable *for the global assumptions G and the local assumptions U if there is an L-model $\langle W, \{\mathcal{R}_a\}_{a \in \mathcal{A}}, \mathcal{I}\rangle$ which satisfies G and a world $w \in W$ which satisfies U and $\mathtt{t}.A$.*

We speak of local (global) logical consequence when $G = \emptyset$ ($G \neq \emptyset$). From a semantical perspective, local logical consequence is identical to validity. Indeed we have that $G \models_{\mathsf{L}} U \Rightarrow A$ is equivalent to $G \models_{\mathsf{L}} \emptyset \Rightarrow (\bigwedge U) \supset A$. It can make a difference for the proof theory: the Gentzen calculus for logic B in [62, 79] requires cut when U is used instead of $\bigwedge U$.

Global assumptions play an important role from a computational viewpoint (§4.5): they shift the complexity of the decision problem from PSPACE (validity) to EXPTIME (logical consequence).

3.3 Single Step Tableaux

Single Step Tableaux (SST for short) use *prefixed formulae* i.e. pairs $\sigma : \varphi$ where σ is an alternating (non empty) sequence of integers and agents names called *prefix* and φ is a signed modal formula. Formally, if $a \in \mathcal{A}$ and n is a positive integer we have $\sigma ::= 1 \mid \sigma.a.n$. Intuitively σ "names" a possible world which satisfies φ.

We say that a prefix σ *starts (arrives)* with a if a is the first (last) agent name which occurs in σ. Formally, $\mathrm{first}(1) = \mathrm{last}(1) = *$ and for $k \geq 1$ it is

$$\begin{aligned} \mathrm{first}(1.a_1.n_1.a_2.n_2.\ldots n_{k-1}.a_k.n_k) &= a_1 \\ \mathrm{last}(1.a_1.n_1.a_2.n_2.\ldots n_{k-1}.a_k.n_k) &= a_k. \end{aligned}$$

$$\alpha: \ \frac{\sigma:\alpha}{\begin{array}{c}\sigma:\alpha_1\\ \sigma:\alpha_2\end{array}} \qquad \beta: \ \frac{\sigma:\beta}{\sigma:\beta_1 \qquad \sigma:\beta_2} \qquad \text{dneg}: \ \frac{\sigma:neg}{\sigma:pos}$$

Figure 3.1: Propositional Tableaux Rules

$$\begin{array}{lll}
Loc: & \dfrac{\vdots}{1:\mathtt{t}.B} & \text{if } B \in U \\
Glob: & \dfrac{\vdots}{\sigma:\mathtt{t}.B} & \text{if } \sigma \text{ is present in the branch and } B \in G \\
\pi: & \dfrac{\sigma:\pi(a)}{\sigma.a.n:\pi_0} & \text{with } \sigma.a.n \text{ new in the branch}
\end{array}$$

Figure 3.2: SST rules common to all logics

If $\sigma = 1.a_1.n_1.a_2.n_2.\ldots n_{k-1}.a_k.n_k$ is a prefix, the length of the prefix σ is $k+1$ and is denoted by $|\sigma|$.

3.3.1 Single step rules

A L-*tableau* $\mathcal{T}$ is a (binary) tree whose nodes are labelled with prefixed formulae. A L-*branch* $\mathcal{B}$ is a path from the root to a leaf. Nodes are labelled in the usual way: if the antecedent of a rule appears in a branch we extend the branch (and possibly split it) labeling the new nodes with the consequent(s). A prefix is L-*present* in a branch $\mathcal{B}$, if there is a prefixed formula in $\mathcal{B}$ with that prefix, and it is *new* if it is not present.

Propositional tableau rules are shown in Fig. 3.1. Figure 3.2 shows the rules which do not depend on the particular L-logic. Finally, the rules which may vary from logic to logic are shown in Fig. 3.3. They are mostly for ν-formulae.

The set of prefixes present in a branch describe a *tree*, spanning from 1 to $1.a.1$, $1.a.1.b.1$ and $1.a.1.b.2$ etc. Loosely speaking, the proof search can be seen as a backward and forward visit of this tree until a contradiction is found: create a node (π-rule), work inside a node (propositional, *Glob* or *Loc* rules), add a formula *forward* from a parent σ to a child $\sigma.n$ or *backward* from child to parent (ν-rules). For instance, rule (4), if we have in mind the knowledge interpretation of $[a]$, corresponds to positive introspection and allows us to "inherit knowledge forward".

$$K: \frac{\sigma : \nu(a)}{\sigma.a.n : \nu_0} \qquad D: \frac{\sigma : \nu(a)}{\sigma : \mathtt{f}.\,[a]\,\bot} \qquad T: \frac{\sigma : \nu(a)}{\sigma : \nu_0}$$

$$4: \frac{\sigma : \nu(a)}{\sigma.a.n : \nu(a)} \qquad 4^R: \frac{\sigma.a.n : \nu(a)}{\sigma : \nu(a)} \qquad B: \frac{\sigma.a.n : \nu(a)}{\sigma : \nu_0}$$

$$4^\pi: \frac{\sigma.a.n : \pi(a)}{\sigma : \pi(a)} \qquad 4^D: \frac{\sigma.a.n : \nu(a)}{\sigma.a.n.a.m : \nu(a)} \qquad Cxt: \frac{\sigma.a.n : \nu(a)}{\sigma : \nu^2(a)}$$

Where σ, $\sigma.a.n$ and $\sigma.a.n.a.m$ must be present in the branch. For rule Cxt, if the antecedent is $\nu(a) = \mathtt{t}.\,[a]\,A$ then $\nu^2(a) = \mathtt{t}.\,[a]\,[a]$ and similarly for $\langle a \rangle A$.

Figure 3.3: SST-rules characterizing logics

Table 3.3: Modal Logics and SST-rules

L	$SST-$ Rules	L	$SST-$ Rules	L	$SST-$ Rules
K	K	KD	K, D	T	K, T
KB	K, B	KDB	K, B, D	B	K, T, B
K4	$K, 4$	KD4	$K, D, 4$	S4	$[K,]\, T, 4$
K5	$K, 4^D, 4^R, Cxt$	KD5	$K, D, 4^D, 4^R, Cxt$	S5	$[K,]\, T, 4, 4^R$
K45	$K, 4, 4^R$	KD45	$K, D, 4, 4^R$	Cxt	K, Cxt
K45	$K, 4^\pi, 4^R$	KD45	$K, D, 4^\pi, 4^R$	S5	$K, T, 4^\pi, 4^R$

The various logics mentioned in Table 3.2 are captured by different sets of SST rules as shown in Table 3.3. In the sequel SST-rule are in italics and between parenthesis e.g. K is the logic whereas (K) is the ν-rule. Square bracketed rules are derivable from the others.

Rule (K) can be simulated by rules (T) and (4):

$$\begin{array}{cc} K - \text{Rule} & T4 - \text{simulation} \\ \dfrac{\sigma : \nu(a)}{\sigma.a.n : \nu_0}\,K & \dfrac{\dfrac{\sigma : \nu(a)}{\sigma.a.n : \nu(a)}\,4}{\sigma.a.n : \nu_0}\,T \end{array}$$

Every agent may be characterized by different sets of rules corresponding to the underlying logic. For instance we can choose rules $K, 4, 4^R$ for an agent characterized by $\mathsf{K45}_a$ and rules $K, T, 4$ for another agent characterized by $\mathsf{S4}_b$.

Remark. For the logics K5 and KD5 we can restrict the applicability of rule (Cxt) to the case where $\text{last}(\sigma) \neq a$.

We have two possible sets of rules for the logics K45, KD45, S5. The first can be seen as the standard cumulative extension of K4 with the addition of extra rules. The second dispatched forward rules in favor of backward ones[4]. The difference is the way in which rules encode the transitivity axiom. Rule (4) encodes $[a]\,A \supset [a]\,[a]\,A$ while rule (4^π) encodes $\langle a\rangle\langle a\rangle A \supset \langle a\rangle A$. Although the combination $K + 4^\pi$ is not complete for K4, the addition of (4^R) is enough for K45. Surprisingly, the proof search in both calculi has the same computational complexity (§4.5).

3.3.2 Tableaux proofs and satisfiability witnesses

The definition of tableau proof is standard [62, 79]:

Definition 3.3 *A* branch $\mathcal{B}$ *is* contradictory *if there is a σ such that, either $\sigma : \mathtt{t}.\bot$ is present or $\sigma : \mathtt{f}.\top$ or for some φ, both $\sigma : \varphi$ and $\sigma : \overline{\varphi}$ are present in $\mathcal{B}$. A* tableau *is* closed *if every branch is closed.*

Definition 3.4 *A* L-tableau proof *for the formula A with global assumptions G and local assumptions U is the closed* L*-tableau for G and U starting with $1 : \mathtt{f}.A$.*

In section 3.6 we will prove the soundness and correctness for any logic L among K, KB, K4, K5, K45, KD, KDB, KD4, KD5, KD45, T, B, S4, S5.

To provide a counter-example for the validity of A, i.e. a satisfiability witness for $\mathtt{f}.A$, we must find a branch $\mathcal{B}$ without contradictions. To this extent we need some preliminary definitions.

We say that a prefix σ is *completed* in a branch $\mathcal{B}$ for a set of formulae S if $A \in S$ implies $\sigma : \mathtt{t}.A \in \mathcal{B}$.

Definition 3.5 *A prefixed formula $\sigma : \psi$ is* reduced *in a branch $\mathcal{B}$ iff*

- *ψ is an α formula and both $\sigma : \alpha_1$ and $\sigma : \alpha_2$ are present;*
- *ψ is a neg formula and $\sigma : pos$ is present;*
- *ψ is a β formula and one of $\sigma : \beta_1$ or $\sigma : \beta_2$ is present;*
- *ψ is a $\pi(a)$ formula and $\sigma.a.m : \pi_0$ is present for some m;*
- *ψ is a $\nu(a)$ formula and for every applicable ν-rule $\sigma : \psi \Rightarrow \sigma^* : \psi^*$ (Table 3.3) and for every σ^* present in $\mathcal{B}$ the prefixed formula $\sigma^* : \psi^*$ is also present.*

A prefix σ is reduced *in $\mathcal{B}$ if every prefixed formula $\sigma : \psi$ is reduced in $\mathcal{B}$.*

[4]This calculus has been found independently by Goré [79].

For instance reduction w.r.t. rule (4^R) is: $\sigma.a.n : \nu(a) \in \mathcal{B}$ implies $\sigma : \nu(a) \in \mathcal{B}$. Notice that a formula may be reduced even if no rule have been applied to that particular formula.

Definition 3.6 *A* branch $\mathcal{B}$ *is* completed *if the prefix* 1 *is completed for* U*, and all prefixes in* $\mathcal{B}$ *are completed for* G *and reduced in* $\mathcal{B}$*; it is* open *if it is completed and not closed. A* tableau *is* open *if at least a branch is open.*

Definition 3.7 *An* L*-satisfiability proof for the formula* A *with local assumptions* U *and global assumptions* G *is the open* L*-tableau for* U *and* G *starting with* $1 : \mathtt{t}.A$.

Soundness and completeness can be reformulated with this definition.

With Def. 3.6, open branches may be infinite, as in first order logic. To generate infinite branches, one can use global assumptions: if $\langle a \rangle A \in G$, then rules (*Glob*) and (π) can generate an infinite sequence of prefixes. The rules (π), (K), and (4) can also generates an infinite branch when applied to formulae like $1 : \mathtt{t}.\langle a \rangle B \wedge [a] \langle a \rangle A$.

If the focus is provability and we do not look for decision procedures, a completed branch is simply the result of any systematic and fair search strategy (see [79] for a formal definition of fair strategy). If we look for decision procedures, we must restrict the applicability of tableau rules.

Technique 3.1 *Apply a rule to* $\sigma : \varphi$ *in* $\mathcal{B}$ *only if it is not reduced in* $\mathcal{B}$.

This is not enough to tame the examples mentioned above. The typical solution is 'loop-checking': never reduce a *set of formulae* if we have already met and reduced this set beforehand. We discuss this issue in section 4.4.

3.4 Incorporating Simplification

To incorporate simplification we do not need to change our general framework nor to impose clausal normal form, as done in `KSAT` [72]. We simply use the following rule:

$$\frac{\sigma : \varphi \qquad \sigma : \psi}{\sigma : \varphi[\psi]} \, \mathsf{L} - simp$$

We inherit simplification on propositional connectives. The only things we need to specify is how $A[\varphi]$ will behave in presence of $[a]$ and $\langle a \rangle$. Since this may depend on the logic, we called the rule L-simplification.

The simplest way is to consider $[a] B$ or $\langle a \rangle B$ as atoms ("modal atoms") and apply boolean simplification (§2.3). This is the strategy of Giunchiglia and Sebastiani [72] for their `KSAT` system (see §3.5).

$$
\begin{array}{lll}
\multicolumn{3}{l}{\overline{\langle a\rangle\,(([a]\,p \supset [a]\,(p \supset r)) \vee \langle a\rangle\neg r)}^{[\mathtt{t}.[a][a]p]}} \\
\mapsto_{\mathsf{K}} & \langle a\rangle & \left(\overline{([a]\,p \supset [a]\,(p \supset r)) \vee \langle a\rangle\neg r}^{[\mathtt{t}.[a]p]}\right) \\
\mapsto_{\mathsf{PC}} & \langle a\rangle & \left(\left(\overline{[a]\,p \supset [a]\,(p \supset r)}^{[\mathtt{t}.[a]p]}\right) \vee \overline{\langle a\rangle\neg r}^{[\mathtt{t}.[a]p]}\right) \\
\mapsto_{\mathsf{PC}} & \langle a\rangle & \left(\left(\overline{[a]\,p}^{[\mathtt{t}.[a]p]} \supset \overline{[a]\,(p \supset r)}^{[\mathtt{t}.[a]p]}\right) \vee \overline{\langle a\rangle\neg r}^{[\mathtt{t}.[a]p]}\right) \\
\mapsto_{\mathsf{K}} & \langle a\rangle & \left(\left(\top \supset [a]\,\overline{(p \supset r)}^{[\mathtt{t}.p]}\right) \vee \langle a\rangle\overline{\neg r}^{[\mathtt{t}.p]}\right) \\
\mapsto_{\mathsf{PC}} & \langle a\rangle & \left(\left(\top \supset [a]\left(\overline{p}^{[\mathtt{t}.p]} \supset \overline{r}^{[\mathtt{t}.p]}\right)\right) \vee \langle a\rangle\neg r\right) \\
\mapsto_{\mathsf{PC}} & \langle a\rangle & ((\top \supset [a]\,(\top \supset r)) \vee \langle a\rangle\neg r) \\
\mapsto_{bool} & \langle a\rangle & ([a]\,r \vee \langle a\rangle\neg r)
\end{array}
$$

Figure 3.4: Step by Step Modal K-simplification

To enhance simplification we look inside modal atoms. In the sequel we skip boolean rules as they remain unchanged. The only new cases for simplification of $A[\varphi]$ are those where the simplifying formula is a modal formula i.e. where φ is either a $\nu(a)$ formula or a π-formula.

*Modal-*K *simplification* is the simplest:

$$
A[\nu(a)] \quad \mapsto \quad \begin{cases} [a]\,(B[\nu_0]) & \text{if } A = [a]\,B \\ \langle a\rangle(B[\nu_0]) & \text{if } A = \langle a\rangle B \\ \ldots & \quad\text{as for propositional logic} \end{cases}
$$

The idea is that if we want to simplify $[a]\,A$ with $[\mathtt{t}.\,[a]\,C]$ (a $\nu(a)$-formula) then we can exploit the properties of the modal operators and transform this operation in the simplification of A with $[\mathtt{t}.C]$ (the corresponding ν_0-formula). See again section 3.2 and Table 3.1 for the $\nu(a)$-formulae.

Intuition. Propositional connectives leave the simplifying formula unchanged in the recursive steps (reduction 2.4). With modal operators we must do the recursion on both the simplified and the simplifying formula.

For instance suppose we have $\langle a\rangle([a]\,p \supset [a]\,(p \supset r) \vee \langle a\rangle\neg r)$ and want to simplify it with the $\nu(a)$-formula $\mathtt{t}.\,[a]\,[a]\,p$. We show the computation step by step in Fig.3.4. For readability we put the simplifying formula on the top:

Modal K4*-simplification* exploits the properties of transitivity (represented by the axiom $[a]\,A \supset [a]\,[a]\,A$):

$$
A[\nu(a)] \quad \mapsto \quad \begin{cases} [a]\,(B[\nu_0][\nu(a)]) & \text{if } A = [a]\,B \\ \langle a\rangle(B[\nu_0][\nu(a)]) & \text{if } A = \langle a\rangle B \\ \ldots & \quad\text{as for propositional logic} \end{cases}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\overline{[a]\,((p \wedge q \supset [a]\,(q \supset p)) \wedge (\neg p \wedge q \supset [a]\,(q \supset \neg p)))}^{[\mathtt{t}.[a]p]}} \\
\mapsto_{\mathsf{K4}} & [a]\left(\overline{(p \wedge q \supset [a]\,(q \supset p)) \wedge (\neg p \wedge q \supset [a]\,(q \supset \neg p))}^{[\mathtt{t}.p]}\right)[\mathtt{t}.\,[a]\,p] \\
\mapsto_{\mathsf{PC}} & [a]\,\overline{\left(\left(\overline{p \wedge q \supset [a]\,(q \supset p)}^{[\mathtt{t}.p]}\right) \wedge \left(\overline{\neg p \wedge q \supset [a]\,(q \supset \neg p)}^{[\mathtt{t}.p]}\right)\right)}^{[\mathtt{t}.[a]p]} \\
\mapsto_{\mathsf{PC}} & [a]\,\overline{\left(\left(\overline{p \wedge q}^{[\mathtt{t}.p]} \supset \overline{[a]\,(q \supset p)}^{[\mathtt{t}.p]}\right) \wedge \left(\overline{\neg p \wedge q \supset [a]\,(q \supset \neg p)}^{[\mathtt{t}.p]}\right)\right)}^{[\mathtt{t}.[a]p]} \\
\mapsto_{\mathsf{PC}}\mapsto_{bool} & [a]\,\overline{\left((q \supset [a]\,(q \supset p)) \wedge \left(\overline{\neg p \wedge q \supset [a]\,(q \supset \neg p)}^{[\mathtt{t}.p]}\right)\right)}^{[\mathtt{t}.[a]p]} \\
\mapsto_{\mathsf{PC}} & [a]\,\overline{\left((q \supset [a]\,(q \supset p)) \wedge \left(\overline{\neg p \wedge q}^{[\mathtt{t}.p]} \supset \overline{[a]\,(q \supset \neg p)}^{[\mathtt{t}.p]}\right)\right)}^{[\mathtt{t}.[a]p]} \\
\mapsto_{\mathsf{PC}} & [a]\,\overline{\left((q \supset [a]\,(q \supset p)) \wedge \left(\bot \wedge q \supset \overline{[a]\,(q \supset \neg p)}^{[\mathtt{t}.p]}\right)\right)}^{[\mathtt{t}.[a]p]} \\
\mapsto_{bool} & [a]\,\overline{(q \supset [a]\,(q \supset p))}^{[\mathtt{t}.[a]p]} \\
\mapsto_{\mathsf{PC}} & [a]\left(\overline{q}^{[\mathtt{t}.[a]p]} \supset \overline{[a]\,(q \supset p)}^{[\mathtt{t}.[a]p]}\right) \\
\mapsto_{\mathsf{K4}} & [a]\left(q \supset [a]\,\overline{\left(\overline{q \supset p}^{[\mathtt{t}.p]}\right)}^{[\mathtt{t}.[a]p]}\right) \\
\mapsto_{\mathsf{PC}}\mapsto_{bool} & [a]\,(q \supset [a]\,\top) \\
\mapsto_{modal} & [a]\,(q \supset \top) \\
\mapsto_{bool}\mapsto_{modal} & \top
\end{array}
$$

Figure 3.5: Step by Step Modal K4-simplification

Thus, for any logic containing the transitivity axiom we can first simplify B with $\nu(a)$ and then simplify the result with ν_0. Transitivity adds a further level of simplification with respect to simple modal logic K.

In Fig.3.5 we show an example for the simple formula $\mathtt{t}.\,[a]\,((p \wedge q \supset [a]\,(q \supset p)) \wedge (\neg p \wedge q \supset [a]\,(q \supset \neg p))$ simplified by $\mathtt{t}.\,[a]\,p$. Formulae of this form are used to construct hard formulae for K4 in [82, 81]. Here we use the standard depth first approach for rewriting.

The recursive calls to simplification always terminate since there is a decrease in the nesting of modal operators of the simplified formula.

Modal-K5 *simplification* is the following

$$
\begin{array}{rcl}
A[\nu(a)] & \mapsto & [a]\,(B[\nu_0]) \text{ if } A = [a]\,B \\
A[\nu(a)] & \mapsto & \langle a\rangle(B[\nu_0]) \text{ if } A = \langle a\rangle B \\
A[\pi] & \mapsto & [a]\,(B[\pi]) \text{ if } A = [a]\,B \\
A[\pi] & \mapsto & \langle a\rangle(B[\pi]) \text{ if } A = \langle a\rangle B
\end{array}
$$

The rule can be understood by considering the corresponding axiom, $\langle a\rangle A \supset [a]\,\langle a\rangle A$: it tells that possibility-like formulae can propagate inside modalities.

When we combine modal logics we combine the corresponding rules for simplifications. For instance K45 (K4+K5):

$$A[\varphi] \quad \mapsto \quad \begin{cases} [a]\,(B[\nu(a)][\nu_0]) & \text{if } A = [a]\,B \text{ and } \varphi = \nu(a) \\ \langle a\rangle(B[\nu(a)][\nu_0]) & \text{if } A = \langle a\rangle B \text{ and } \varphi = \nu(a) \\ [a]\,(B[\pi]) & \text{if } A = [a]\,B \text{ and } \varphi = \pi \\ \langle a\rangle(B[\pi]) & \text{if } A = \langle a\rangle B \text{ and } \varphi = \pi \\ \ldots & \text{ as for propositional logic} \end{cases}$$

Theorem 3.8 *For every logic* L*,* L*-Modal simplification is a sound and invertible rule.*

The proof follows by an analysis of the semantical properties defined in 3.2.

Theorem 3.9 L*-Modal simplification can be performed in polynomial time in the size of the simplified and simplifying formula.*

The proof is based on a simple counting argument (see appendix A.1).

The size of the simplified modal formula $\psi[\varphi]$ is never longer. In general can be much shorter, although there are few cases where we do not gain after simplification. For instance if we have the formulae $\psi = \mathtt{t}.\,[a]\,p$ and $\varphi = \mathtt{f}.\langle a\rangle p$. After simplification one has $\psi[\varphi] = \mathtt{t}.\,[a]\,\bot$. We cannot simplify further $[a]\,\bot$ since its consistency or inconsistency depends on the other formulae.

3.5 From Gentzen-type and Prefixed Tableaux to SST

We can see a tableau proof as a failed attempt to find a countermodel for a valid formula. Loosely speaking, the proof search is a "journey" through a tentative Kripke model, jumping from world to world, chasing a contradiction. The difference between Gentzen-type, prefixed and single step tableaux is precisely the way they travel and keep track of such a journey.

3.5.1 Forward rules in Gentzen-type and prefixed tableaux

The main intuition behind SST is inherited from Kripke [96]:

Intuition. SST view the prefixes within branch as a tree, label nodes with a single formula, reduce formulae one by one, and moves only by one level in the tree of prefixes (towards a child or a parent node).

Gentzen-type tableaux reduce modal connectives by transforming a whole set of modal formulae S into another set S^a. For instance, the logic K4 is

K4 Gentzen-type reduction

$$\frac{\{\, \mathtt{t}.\langle a\rangle A, \mathtt{t}.\,[a]\,B, \mathtt{t}.C \wedge D \,\}}{\{\, \mathtt{t}.A, \mathtt{t}.B, \mathtt{t}.\,[a]\,B \,\}}\,\mathsf{K4}$$

The first set describes world 1 the second describes world 1.1.

K4 single step reduction

$1 : \mathtt{t}.\langle a\rangle A$	(a)
$1 : \mathtt{t}.\,[a]\,B$	(b)
$1 : \mathtt{t}.C \wedge D$	
$\mid$	
$1.a.1 : \mathtt{t}.A$	(π) to a
$\mid$	
$1.a.1 : \mathtt{t}.B$	(K) to b
$\mid$	
$1.a.1 : \mathtt{t}.\,[a]\,B$	(4) to b

Figure 3.6: Gentzen-type tableau vs SST for K4

characterized by the following rule:

$$\frac{\{\, \pi(a) \,\} \cup S}{\{\, \pi_0 \,\} \cup S^a}\, K4 \text{ where } S^a \doteq \{\, \nu_0, \nu(a) \mid \nu(a) \in S \,\}$$

In this formalism tableaux are trees where each node is labelled by a set of formulae[5].

Intuition. Such rules correspond to a jump forward from the world described by S to the world described by S^a. It is a jump along one arc of the accessibility relation. One never looks back again.

Intuition. Single Step Tableaux decompose such a 'large' jump into many 'smaller' steps which only take one formula as input.

The difference between such global rule and the SST rules is shown in Fig. 3.6. We can simulate the K4 jump with many subsequent application of the (K) and (4) SST rules.

We delete formulae in the Gentzen calculus while the SST α-rule can still be applied to the formula $1 : C \wedge D$.

Soundness is at stake: a complex rule can be globally sound but its "stepwise computations" may not be such. The key point is that the correctness proofs of Gentzen-type tableaux are stepwise. In a nutshell, if $w\mathcal{R}_a v$ we can

[5]If we arrange deduction in a way that a single formula labels a node of the tree, then we must use a *global branch modification rule*: delete all nodes in a branch and replace them with others, see Fitting [62]. There is no substantial difference between the two methods.

$\mathsf{K4}$ prefixed tableau	$\mathsf{K4}$ single step tableau	
$1 : \mathtt{t}.\,[a]\,A$	$1 : \mathtt{t}.\,[a]\,A$	
$\vert$	$\vert$	
$\vert$	$1.a.1 : \mathtt{t}.\,[a]\,A$	(4)
$1.a.1.a.1 : \mathtt{t}.A$	$\vert$	
Since $1 \rhd^{a}_{\mathsf{K4}} 1.a.1.a.1$	$1a.1.a.1 : \mathtt{t}.A$	(K)

Figure 3.7: Prefixed Rule vs SST for $\mathsf{K4}$

pick each $\varphi^a \in S^a$ and show that its truth value in v is forced by the truth value in w of the corresponding formula $\varphi \in S$.

Prefixed tableaux á la Fitting [62] introduce prefixes to "name" worlds and mimic the behavior of Kripke semantics with a syntactic relation $\rhd$ between prefixes. For instance, in the case of $\mathsf{K4}$, if $1 \rhd_{\mathsf{K4}} 1.a.1$ and $1.a.1 \rhd_{\mathsf{K4}} 1.a.1.a.1$ then we have $1 \rhd_{\mathsf{K4}} 1.a.1.a.1$. Deduction is performed by considering a single formula $\sigma : \nu(a)$ and adding $\sigma^* : \nu_0$ for every σ^* such that $\sigma \rhd^{a}_{\mathsf{L}} \sigma^*$.

Intuition. This deduction steps corresponds to a jump from the world named σ to another (possibly far away) world named σ^*.

Intuition. SST break a 'long' jump into a sequence of 'shorter' steps.

The difference between prefixed tableaux and SST is shown in Fig. 3.7. SST emulate the effect of transitivity by first moving to world $1.a.1$ a ν-formula and then using rule (K).

Completeness is under siege: in a prefixed tableau we can jump far away whereas in a single step tableau we may lack the intermediate worlds needed to fill the gaps between our small jumps. Again, the analysis of the completeness proof for prefixed tableaux à la Fitting reveals that all intermediate possible worlds are there, although the calculus does not use them.

3.5.2 Backward rule and analytic cut

Euclidean and symmetric logics have accessibility relations where the existence of an arc between two worlds in one direction ('forward') imposes the existence of another arc in the opposite direction ('backward').

Intuition. In these logics, discovering a formula in a possible world forces constraints on the past worlds we have seen. For strong completeness, Gentzen-type tableaux must use cut to guess the formulae that will be discovered.

Gentzen-type deduction for KB

$$\begin{array}{c}
\{\, \mathtt{t}.\langle a\rangle\,[a]\,A \,\} \\
\hline
\begin{array}{cc}
\begin{array}{c}
\{\, \mathtt{t}.\langle a\rangle\,[a]\,A, \mathtt{f}.A \,\} \\
\hline
\{\, \mathtt{t}.\,[a]\,A, \mathtt{f}.\,[a]\,A \,\} \\
\hline
\bot
\end{array} \;\mathsf{KB}
&
\{\, \mathtt{t}.\langle a\rangle\,[a]\,A, \mathtt{t}.A \,\}
\end{array}
\end{array}\; cut$$

Single step tableau

$$\begin{array}{ll}
1 : \mathtt{t}.\langle a\rangle\,[a]\,A & (\pi) \\
\quad | & \\
1.a.1 : \mathtt{t}.\,[a]\,A & (B) \\
\quad | & \\
1 : \mathtt{t}.A &
\end{array}$$

For the Gentzen-type derivation, the initial set corresponds to world 1, the first KB step leads to world $1.a.1$.

Figure 3.8: Backward Rule rather than Cut for $\mathsf{K45}$

Intuition. SST use backward rules to bring back newly discovered formulae in previous worlds. We gain some of the computational power of cut without its choking branching factor.

In Figure 3.8 we compare the Gentzen-type rule for KB with the corresponding backward rule in SST. For the Gentzen calculus citeFitt-83,Gore-95 the KB-rule transforms $\{\, \pi(a) \,\} \cup S$ into $\{\, \pi_0 \,\} \cup \{\, \nu_0, \pi(a)' \mid \nu(a), \pi_0' \in S \,\}$.

Notice that cut is necessary for the completeness of local logical consequence in KB. An example is $\models_{\mathsf{KB}} \{\, A, B \,\} \Rightarrow [a]\,\langle a\rangle(A \wedge B)$ [62].

3.5.3 Simulation of KSAT

The decision procedure `KSAT` has been introduced in [72] as an alternative to standard tableaux procedures for modal logics.

The major intuition is to combine a decision method for propositional logic (namely `DPLL`) with a standard Gentzen-type calculus for modal logic. This hybrid method has been claimed by the authors to be a new, more efficient deduction method. We have already shown (§2.5) that `DPLL` is an instance of tableaux with simplification and therefore the next step is immediate.

Fact 3.10 *Basic* `KSAT` *is an instance of the Gentzen-type calculus for logic* K *with propositional simplification where* $G = \emptyset$*, and* $\{\, \mathtt{t}.B | B \in U \,\}$ *and* $\mathtt{f}.A$ *are reduced in modal clausal normal form with only the* $[a]$ *operator.*

This restriction is so strong that `KSAT` may not even simplify directly $[a]\,p \wedge \langle a\rangle\neg p$ without reduction to normal form. Indeed, a more careful experimental analysis by Hustadt & Schmid [90] has shown that `KSAT` is not substantially more efficient than traditional tableau methods.

3.6 Soundness and Completeness for SST

As a preliminary result we can prove that SST satisfy a proof theoretical analogue of the Generation Theorem by Segerberg [64].

Definition 3.11 *A set of prefixes* Σ *is* tree generated *iff*

1. *the prefix* $1 \in \Sigma$ *and it is the only prefix of length* 1*;*
2. *if prefix* $\sigma.a.n \in \Sigma$ *then also* $\sigma \in \Sigma$.

By a simple induction on the rules used to construct a branch it is possible to prove the following theorem:

Theorem 3.12 *Let* $\mathcal{B}$ *be a branch of a single step tableau. The set of prefixes occurring in the branch* $\{\sigma | \sigma : \varphi \in \mathcal{B}\}$ *is tree generated.*

3.6.1 Strong soundness

To prove the correctness of prefixed and labelled systems one establishes a mapping between 'names' (prefixes) and 'things' (possible worlds) and shows that the mapping is an homomorphism preserved by tableau rules. To prove the soundness of Gentzen-type tableaux one shows that if a set of formulae holds in a world then a related set holds in an adjacent world.

The soundness proof of SST is a combination of both techniques: set a mapping between prefixes and worlds, do not worry about homomorphisms and use the stepwise proof of Gentzen-type tableaux.

To this extent, we need to establish some properties of the logics at hand, in a natural correspondence with its SST rules. We focus on ν rules (for the others see Fitting [62]) and divide them into three main classes, according the form of the prefixes which are used in the antecedent or in the consequent of each rule. In *static, rules* both the consequent and the antecedent formula have the same prefix. For example, rules (D) and (T) are static. In *forward rules* the prefix of the consequent formula is one step longer than the prefix of the antecedent. Rules (K), (4^D), and (4) are examples of forward rules. In *backward rules* the prefix of the consequent formula is one step shorter than the prefix of the antecedent. Rules (4^R), (B), or (Cxt) are backward rules.

Rules can be *delayed*. This classification is orthogonal to the previous one. It refers to rules applicable only to prefixes longer than two.

Table 3.4 identifies the correspondence between rules and properties of L-models needed for the proof (see appendix A.1) of the following lemma.

Lemma 3.13 *Let* $\sigma : \varphi \Rightarrow \sigma^* : \psi$ *be an SST rule for logic* L*. Every* L*-model verifies the corresponding semantic property of Table 3.4.*

Table 3.4: SST-Rules and Model Properties

Rule Form	Semantic Property
$\sigma : \varphi \Rightarrow \sigma : \psi$	$w \Vdash \varphi$ implies $w \Vdash \psi$
$\sigma : \varphi \Rightarrow \sigma.a.n : \psi$	$w\mathcal{R}_a v$ and $w \Vdash \varphi$ implies $v \Vdash \psi$
$\sigma.a.n : \varphi \Rightarrow \sigma : B$	$w\mathcal{R}_a v$ and $v \Vdash \varphi$ implies $w \Vdash \psi$
$\sigma.a.m : A \Rightarrow \sigma.a.m : \psi$	$\exists w_0.w_0\mathcal{R}_a w$ and $w \Vdash \varphi$ implies $w \Vdash \psi$
$\sigma.a.m : \varphi \Rightarrow \sigma.a.m.n : \psi$	$\exists w_0.w_0\mathcal{R}_a w$ and $w\mathcal{R}_a v$ and $w \Vdash \varphi$ implies $v \Vdash \psi$
$\sigma.a.m.n : \varphi \Rightarrow \sigma.a.m : \psi$	$\exists w_0.w_0\mathcal{R}_a w$ and $w\mathcal{R}_a v$ and $v \Vdash \varphi$ implies $w \Vdash \psi$

The next step is the definition of a mapping between names and things:

Definition 3.14 *Let $\mathcal{B}$ be a branch and $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ be an* L*-model. An* SST-interpretation *is a mapping from prefixes to possible world $\imath(\sigma) \in W$ such that for all σ and $\sigma.a.n$ present on $\mathcal{B}$ it is $\imath(\sigma)\mathcal{R}_a\imath(\sigma.a.n)$.*

SST interpretations do not depend on the logic L in contrast with prefixed tableau interpretations proposed by Fitting [62]. Loosely speaking, SST interpretations are just K-interpretations.

Definition 3.15 *Let $\mathcal{M} = \langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ be an* L*-model. A tableaux branch $\mathcal{B}$ is* L*-SAT for $\mathcal{M}$ with local assumptions U and global assumptions G iff there is an SST-interpretation $\imath()$ such that*

1. *the* L*-model $\mathcal{M}$ satisfies G;*
2. *the world $\imath(1)$ satisfies U in $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$;*
3. *for every prefixed formula $\sigma : \varphi$ in $\mathcal{B}$, the world $\imath(\sigma)$ satisfies φ.*

A tableau is L*-SAT if at least a branch is such.*

Now we have the machinery to prove a *safe extension lemma* by induction on the number of rules we have applied (for the proofs see appendix A.1)

Theorem 3.16 *Let $\mathcal{T}$ be an* L*-SAT tableau, the tableau $\mathcal{T}'$ obtained by an application of an SST rule for* L *is also* L*-SAT.*

The soundness theorem is now standard [62, 79]: if A were not valid then $\mathtt{f}.A$ would be satisfiable and therefore any tableau expansion will always have a SAT branch. If we obtain a tableau proof then A is surely valid since SAT branches cannot obviously be closed.

Theorem 3.17 (Strong Soundness) *If A has a* L*-tableau proof with local assumptions U and global assumptions G then A is a* L*-logical consequence of U and G.*

Logic	Conditions on the accessibility relation over prefixes $\sigma \rhd_a \sigma^*$
K	$\sigma \rhd_a \sigma.a.n$
KB	$\sigma \rhd_a \sigma.a.n$ and $\sigma.a.n \rhd_a \sigma$
K4	$\sigma \rhd_a \sigma(.a.n_i)^k$ with $k \geq 1$
K5	$\sigma \rhd_a \sigma.a.n$ and $\sigma(.a.m_j)^h \rhd_a \sigma(.a.n_i)^k$ with $k \geq 1$ and $h \geq 1$
K45	$\sigma(.a.m_j)^h \rhd_a \sigma(.a.n_i)^k$ with $k \geq 1$
KD-logics	as K-logics provided that there is an n such that $\sigma \rhd_a \sigma.a.n$ or $\sigma \rhd_a \sigma$
T	$\sigma \rhd_a \sigma.a.n$ and $\sigma \rhd_a \sigma$
B	$\sigma \rhd_a \sigma.a.n$ and $\sigma.a.n \rhd_a \sigma$ and $\sigma \rhd_a \sigma$
S4	$\sigma \rhd_a \sigma(.a.n_i)^k$
S5	$\sigma(.a.m_j)^h \rhd_a \sigma(.a.n_i)^k$
Cxt	$\sigma(.a.m_j)^k \rhd_a \sigma(.a.n_i)^{k+1}$

Table 3.5: Conditions on the syntactic relation $\rhd_a$

3.6.2 Strong completeness

The completeness proof follows the same path of Fitting's proof for his prefixed tableaux [62]: apply a systematic strategy to the tablea and use an open branch to build a model for the initial formula $\neg\varphi$ i.e.a counter-model for A. To this extent, identify prefixes present in the branch with worlds and show that if $\sigma : \varphi$ occurs in the branch then also $\langle\sigma\rangle \Vdash \varphi$.

The first part is the construction of a syntactic relation $\rhd_a$ between prefixes with the properties of the semantic relation $\mathcal{R}_a$. Each logic L has its relation $\rhd_a$ as shown in Table 3.5. For readability we abbreviate the prefix $\sigma.a.n_1.a.n_2 \ldots a.n_k$, with k a-steps by $\sigma(.a.n_i)^k$. Formally we have

$$\begin{array}{rcl}\sigma(.a.n_i)^0 & = & \sigma \\ \sigma(.a.n_i)^{k+1} & = & \sigma(.a.n_i)^k.a.n_{k+1}\end{array}$$

Theorem 3.18 *For every logic* L*, the syntactic relation* $\rhd_a$ *over prefixes has the same properties of the semantic relation* $\mathcal{R}_a$ *over worlds.*

The proof is long and tedious: replace possible worlds with prefixes in the properties of $\mathcal{R}_a$ (Table 3.2), use for $\rhd_a$ the conditions of Table 3.5 and reason by cases[6] (see appendix A.1).

[6]Note that the relation $\rhd_a$ for K5 in [113] is incomplete and thus the original system is incomplete. See also Goré [79] for a detailed proof of K5 completeness.

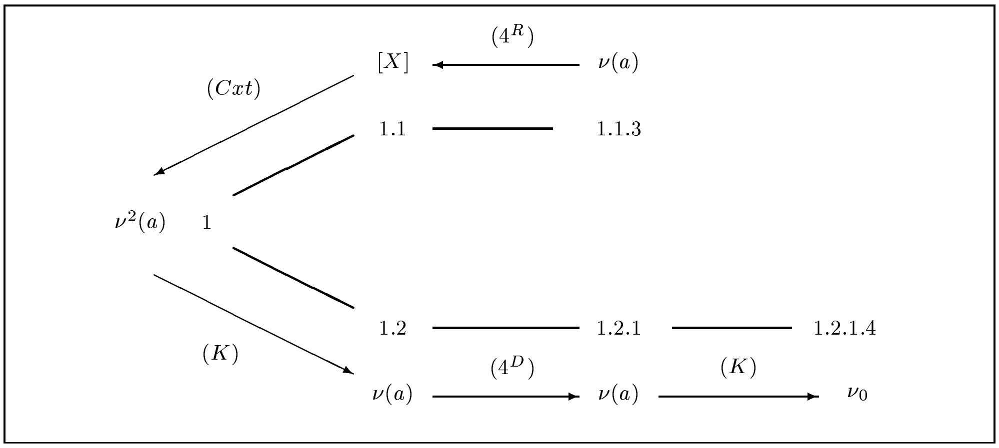


Figure 3.9: Traveling Throughout Prefixes in K5

The second part is the proof of *complete reduction* i.e. if $\sigma : \nu(a)$ is in the branch and $\sigma \rhd_a \sigma^*$ then $\sigma^* : \nu_0$ is also there.

Prefixed and labelled tableaux [62, 66, 151] use $\rhd_a$ in the calculus and have only one ν-rule: if $\sigma : \nu(a)$ is present and $\sigma \rhd_a \sigma^*$ with σ^* also present, then add $\sigma^* : \nu_0$. Thus the proof procedure itself grants complete reduction. However, complete reduction is too powerful since a set can be completely reduced without being the result of a tableau proof e.g. $\{1 : \nu(a), 1.a.2.a.3.a.4 : \nu_0\}$ is completely reduced for K4 but doesn't correspond to any tableaux proof.

SST can recover complete reduction from SST-reduction by letting a formula 'travel' along prefixes. A simple example is shown in Fig. 3.9 for K5. For sake of readability we have dropped the name of agent a. In this case we have that $\langle 1.1.3.1\rangle \rhd_a \langle 1.2.1.4.1\rangle$ and thus the presence of $\langle 1.1.3.1 : \nu(a)\rangle$ should imply the presence of $\langle 1.2.1.4.1 : \nu_0\rangle$.

We classify each logic according the "ability" of its syntactic accessibility relation $\rhd_a$. The relation $\rhd_a$ connects

immediate neighbors if $\rhd_a$ connects prefixes which are one a-step shorter or longer (logics K, KB and their serial, or reflexive variants);

far forward prefixes where the relation $\rhd_a$ can link a prefix σ with another prefix σ^* which can be longer than σ but still σ must be an initial subsequence of σ^* (logic K4 or its variants);

far backward prefixes $\rhd_a$ can link a long prefix σ with a shorter σ^* such that σ^* is the initial subsequence of σ (logic K5 or its variants).

A logic can also satisfy a combination of these properties, e.g. K45.

So we need to prove a *complete reduction lemma* (see appendix A.1):

Lemma 3.19 *Let $\mathcal{B}$ be a branch of a tableau, if $\mathcal{B}$ is completed for the logic* L *then it is also completely reduced.*

The basic intuition is that for syntactic relation $\rhd_a$ which access only *close neighbors*, then complete reduction coincides with SST reduction.

If $\rhd_a$ accesses *forward (backward) prefixes* then use rule 4 (respectively rule 4^R) for copying $\nu(a)$ from σ to $\sigma.a.n$ (respectively from $\sigma.a;n$ to σ) and so on until arrived at the immediate predecessor of σ^*. Then apply rule (K).

Then we can prove a *strong model existence theorem* (appendix A.1).

Theorem 3.20 *If $\mathcal{B}$ is open branch for the rules of logic* L*, then there is an* L*-model for G and U where $\mathcal{B}$ is SAT.*

The intuition is to construct the model by using each prefix σ as world $\langle\sigma\rangle$ and then proving by structural induction on the construction of every prefixed formula that $\sigma : \varphi \in \mathcal{B}$ implies $\langle\sigma\rangle \Vdash \varphi$.

The only trick is to construct the model so that $\langle\sigma\rangle\, \mathcal{R}_a \,\langle\sigma^*\rangle$ only if $\sigma \rhd_a \sigma^*$. In this way we can use Theorem 3.18 in the proof to show that the open branch also satisfies complete reduction for ν-formulae: if $\sigma : \nu(a)$ is present and $\sigma \rhd_A \sigma^*$ then $\sigma^* : \nu_0$ is also present. Then we can apply the induction hypothesis to ν_0 and we are done.

The strong completeness theorem is now standard [62].

Theorem 3.21 (Strong Completeness) *If A is a* L*-logical consequence of local assumptions U and global assumptions G then A has a* L*-tableau proof with U and G.*

Chapter 4

A Computational Analysis of Modal Tableaux

Modal theorem proving introduces new difficulties with respect to propositional logics.

We discuss the methodology and the computational characteristics of Single Step Tableaux and compare them with other systems. In particular we focus on computational properties such as modularity, query combination, the design of decision procedures and the complexity of search strategies.

4.1 Introduction

Computational properties of different calculi can be compared in two possible ways: by extensive experiments on large sets of (possibly random) instances or by theoretical analysis of computational properties. Both techniques are equally necessary, since theoretical results may not tell us enough about practical cases whereas experiments may only test the relative efficiency of implementations or be misled by wrong test suites. For instance see how the experimental claims of Giunchiglia & Sebastiani [72] have been rebutted by Hustadt & Schmidt [90] which showed that the test suites where biased.

In the absence of a reference benchmark suite analogous to the IFIP benchmark or the random 3-SAT benchmarks for propositional logic, we prefer the reliable analysis of computational properties. Among them, we focus on complexity theoretic properties (space requirements), confluence, decidability, locality and query combination.

The first feature one would like in a modal prover is *query combination*: with so many logics around, we would like a prover able to answer something like "A is a theorem for the following logics...". The prover (and the calculus behind it) should be able to combine validity queries in a more efficient way than solving the problem from scratch for every logic. We would like to spend exponential resources on one logic, capitalize on this effort and spend a polynomial amount of time on the others.

This seems hard for tableau methods but fairly simple for methods which translate modal logics into first order logic and then use logic-dependent unification procedures. We may even be leead to believe that "If many different unification algorithms unify these labels then the same proof suffices for the many different modal logics. Consequently these systems give answers like 'A is a theorem for the following logics'." [79, pag. 93]. We show that this combination of queries cannot be done (unless PTIME=NPTIME).

The next property under consideration is proof confluence which is satisfied by resolution methods (either direct or translations based) and which is *not* satisfied by Gentzen-type calculi. We show that SST are confluent.

For the construction of decision procedures, we focus at first on the traditional method of loop-checking [49, 86, 181] and show how it can be captured by our prefixed tableau framework. We will incorporate it directly in our construction for dynamic logic.

Since loop-checking is expensive, we show that also for transitive logics it can be replaced by a trivial termination check using SST and how this result can be generalized to other systems[1]. We focus on translation methods and give the conditions for transforming resolution in a decision procedure with functional and mixed translation methods.

We analyze the complexity of search strategies for both validity and logical consequence. By complexity theoretic consideration we show that any (polynomial) local check is impossible for logical consequence in non transitive logics.

We analize the complexity of the two SST calculi for the logics K45, KD45 and S5 and show that they achieve the complexity lower bound for satisfiability checking [98].

Finally we discuss the complexity of search strategies for validity in a number of systems and we show that translation methods in first order logic are inherently inefficient since they require exponential space. Last the design of PSPACE search strategies for validity checking algorithm is shown.

4.2 Modularity and Query Combination

The first property of interest is *modularity*. Hilbert axiomatizations are modular and cumulative "par excellence": once we fix the set of rules and axioms that we want to use, it is clear that the logic KD4 is an extension of (has more

[1] Similar results have been shown independently by Cerrito & Cialdea [25] and by Schmidt [157] with much larger (exponential) bounds.

theorems than) K. Translation methods based on the relational translation [69] or the functional translation with an explicit equality theory [136] are modular: change the first order (equality) axioms representing the accessibility relation and change the logic.

Gentzen calculi and the corresponding tableaux [62, 79] are much less modular since the logic dependent part is compiled into one single rule for π-formulae. To change the logic we must change the whole rule. The only modular part is the presence of a single rule for reflexivity.

Simple step tableaux have the same cumulative flavor of Hilbert-style axiomatizations: the set of available rules is given in Table 3.3 and by taking different subsets of rules we obtain different logics. The calculus with rule (K) and (4) is clearly an extension of the calculus using only rule (K).

Remark. There is not a 1-1 mapping between axioms and SST-rules.

For instance SST can simulate the effect of three axioms $K + T + 4$ with two rules $(T) + (4)$. On the other side, rule (4^R) alone cannot simulate the power of axiom 5. In a similar way $(K) + (T) + (4) + (B)$ is not enough to get completeness for S5. Overlooking this fact may lead to incomplete formalizations. For instance the rules for K5 of Massacci [113] are not complete and the rules for S5 of Kripke [96] are not complete.

4.2.1 Query combination

Modularity makes it easy to pass from the rules and axioms of logic L_i to those of logic L_j. So, having a modular calculus should be of some help in processing queries like "Is A a theorem for the logics $\mathsf{L}_1, \ldots \mathsf{L}_n$?". We should be able to get something better than simply recomputing the query for each logic L_i: during the proof of L_i, our modular calculus should make some use of the (costly) information about validity in L_{i-1}, to speed up the search.

We are asking whether we can build the efficient prover in Fig. 4.1 for *query combination*. For instance we could try T as the fixed logic and then query the polynomial box for the logic K, K4, S4 etc. Using this construction, we can answer queries like "A is a theorem for L_{fix} and L_{var}" in a more efficient way than computing the query from scratch for each logic. This construction is not immediately impossible: deciding validity for L_{var} still takes PSPACE.

To prove that such construction is impossible (at least for the most common modal logics) we assume that the input language is the same for all logics, namely the standard language introduced in §3.2. What is inside the boxes (translation method, Gentzen calculi, etc.) is of no concern to us.

In the sequel we refer to a modal logic L, possibly with subscripts, to denote any of the modal logics K, KB, K4, K5, K45, KD, KDB, KD4, KD5, KD45, T, B, S4, S5 (see §3.2). We recall that PC is the classical propositional calculus.

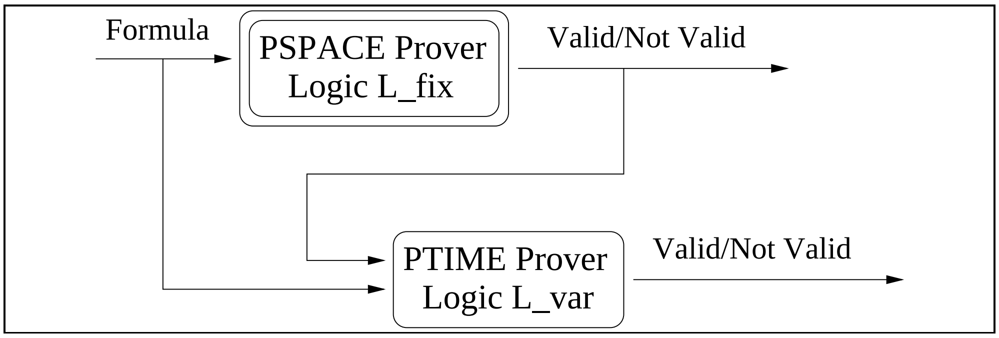


Figure 4.1: A 'Streamlined' Prover for Many Modal Logics

Definition 4.1 *Modal logic* L' extends *modal logic* L*, in symbols* $\mathsf{L} \subset \mathsf{L}'$ *iff*

1. *the set of valid formulae of* L' *contains the set of valid formulae of* L *and*

2. *there is an axiom schema* A *whose instances are valid formulae of* L' *but not of* L*.*

For instance $\mathsf{S4}$ extends $\mathsf{K4}$ which extends K.

Definition 4.2 *A calculus for the modal logics* L_{fix} *and* L_{var} *satisfies the property of* query combination *iff it can be used to construct the system of Fig. 4.1 for deciding validity in both logics.*

Definition 4.3 *Query combination is* trivial *for formula* A *and modal logics* L_{var} *and* L_{fix} *if any of following conditions holds*

- $\models_{\mathsf{L}_{fix}} A$ *and* $\mathsf{L}_{fix} \subset \mathsf{L}_{var}$
- $\not\models_{\mathsf{L}_{fix}} A$ *and* $\mathsf{L}_{var} \subset \mathsf{L}_{fix}$*,*

For example, if A is a theorem of T, it is clearly a theorem of $\mathsf{S4}$ and $\mathsf{S5}$. For the second condition, if A is not a theorem for $\mathsf{S4}$, then it is obviously not a theorem for K, T and $\mathsf{K4}$. Everybody can combine queries when these conditions hold; there is no need of a modular calculus.

Theorem 4.4 *Let* L_{fix} *and* L_{var} *be two modal logics such that*

- *each of them extends* PC*,*
- *either* $\mathsf{L}_{fix} \subset \mathsf{L}_{var}$ *or* $\mathsf{L}_{var} \subset \mathsf{L}_{fix}$*,*

then query combination is trivial or PTIME=NPTIME*.*

The key part of the proof (see appendix A.2) is to show that, whenever query combination is not trivial, the construction of Fig. 4.1 can be used to solve propositional satisfiability in polynomial time.

The trick is embedding propositional formulae B into suitable input formulae $A[B]$, so that the outcome of the PSPACE box is constant (independent from B), while the outcome of the PTIME box coincides with satisfiability of B in PC. If embedding B into $A[B]$ takes polynomial time, then we have found a system for answering NPTIME-complete queries in PTIME. □

We have used validity as the output of the PSPACE box but the same result can be obtained for satisfiability. Without difficulties, the theorem can be generalized to the case when L_{fix} 'differs' from L_{var} because of an axiom schema (e.g. KB differs from K4).

Notice that we cannot use the proofs of the calculus as the output of the PSPACE-box, since proofs (or counter models) can be exponentially large in the size of the input formula unless PTIME=PSPACE. If we start passing proofs (or counter models) to the 'quicker' box, the overall system will take exponential space, and the PTIME-box itself will take exponential time just to read the input. This would be more inefficient than re-computation from scratch.

In conclusion, the worst case complexity of any system which pretends to answer "A is a theorem for the following modal logics..." cannot be substantially better than the complexity of a system which computes these answers from scratch for every modal logic.

4.3 Proof Confluence

The study of confluence has a long tradition in rewrite systems and we refer to the survey by Huet [88] for an introduction.

4.3.1 Preliminary definitions and comparisons

In the sequel x, y, z, u will be stages of the computation and $\rightarrow$ will be a relation between them. Here, the computation is the proof search and its stages are tableaux, sets of clauses etc. If x and y are tableaux, then the relation $x \rightarrow y$ holds y is obtained from x with an application of a tableau rule. For resolution, x is set of clauses and y the same set plus a resolvent of two clauses in x or minus a subsumed clause. The relation $\rightarrow^*$ is the reflexive, transitive closure of $\rightarrow$ whereas $\rightarrow^\epsilon$ is the reflexive closure.

Definition 4.5 *[88, p. 779] The relation $\rightarrow$ is* strongly confluent *iff*

$$\forall xyz.\ \text{if } x \rightarrow y \textit{ and } x \rightarrow z \text{ then } \exists u.\ y \rightarrow^* u \ \textit{ and } \ z \rightarrow^\epsilon u$$

Exact confluence is too strict and it is only satisfied by *modal resolution* [54]. We are usually interested in confluence modulo an equivalence relation. For instance, in first order resolution we are interest in confluence up to renaming of bound variables. If $\sim$ is an equivalence relation between stages of the computation we have (see [88, page 802]):

Definition 4.6 *The relation* $\rightarrow$ *is* strongly confluent modulo $\sim$ *iff*

$$\begin{array}{lll} \forall x_1 x_2 y_1 y_2 & \textit{if} & x_1 \sim x_2 \textit{ and } x_1 \rightarrow y_1 \textit{ and } x_2 \rightarrow y_2 \\ & \textit{then} & \exists u_1 u_2.\, u_1 \sim u_2 \textit{ and } y_1 \rightarrow^* u_1 \textit{ and } y_2 \rightarrow^\epsilon u_2 \end{array}$$

It is immediate to see that *functional and relational translation methods* based on resolution [69, 133, 135, 136] are strongly confluent modulo isomorphic renaming of bound variables.

Among direct methods *Fitch-style tableaux or Gentzen calculi are not confluent*. The 'guilty' rule is the π rule (see in §3.5 for examples): replace the set $S \cup \{ \pi(a) \}$ with the set $S^a \cup \{ \pi_0 \}$, where S^a is obtained from S by considering a subset of the formulae of S. Typically one focuses on $\nu(a) \in S$ (or $\pi(a)$) and deletes other formulae [61, 62, 79].

Systematic strategies may be no longer fair so that backtracking becomes the only choice and the search strategy becomes critical (see Demri [49]). We have a partially negative answer to Question 1.3: some search strategy may lose completeness. Consider the following selection technique:

Technique 4.1 *Select formulae which have been reduced the least number of times; then select π formulae then β formulae.*

This strategy is systematic but is not complete for Gentzen-type tableau: try $\{ [a]\, A \vee [a]\, B, \langle a \rangle (\neg A \wedge \neg B) \}$. In constrast, it is complete for SST. More examples of systematic yet incomplete strategies for Gentzen-type tableaux are discussed by Demri [49].

The practical consequence is that we must do a lot of (maybe pointless) case analysis with β-rules before trying any π-rules. This substantially hinders the effectiveness of the proof search as shown by Hustadt & Schmidt [90].

4.3.2 SST confluence

Single step tableaux are strongly confluent modulo a renaming of prefixes. To prove this result we need some preliminary definitions.

Definition 4.7 *A injective and surjective function h from the set of prefixes onto itself is a* renaming *iff* $h(1) = 1$ *and* $h(\sigma.a.n) = h(\sigma).a.m$ *for some integer* m.

Intuitively, a renaming permutes the integers and leaves the structures of prefixes unchanged.

Definition 4.8 *The sets of prefixed formulae $\mathcal{B}_1$ and $\mathcal{B}_2$ are equivalent modulo a renaming of prefixes, in symbols $\mathcal{B}_1 \sim \mathcal{B}_2$, iff there are two renamings h_{12} and h_{21} such that $h_{ij}(h_{ji}(\sigma)) = \sigma$ and if $\sigma : \varphi \in \mathcal{B}_i$ then $h_{ij}(\sigma) : \varphi \in \mathcal{B}_j$ for $i, j = 1, 2$.*

This definition can be extended to tableaux as sets of branches.

Theorem 4.9 *SST rules are strongly confluent modulo isomorphic renaming of prefixes, provided that ν-formulae can be reduced more than once.*

We discuss the key steps, leaving details in appendix A.2. The difficulties are due to rules with side conditions such as the π-rule (Fig. 3.2) and the ν-rules (Fig. 3.3), because side conditions impose an ordering.

The first case is when a prefix must be introduced by a (π) rule before we can apply rule (K) to it. This also happens for matrix based methods (e.g. K-admissible substitution [178, page 113]). However, confluence does not require an arbitrary shuffling of rules: it requires that **if** we can choose which rule to apply, **then** the choice should be irrelevant. With rules (π) and (K) we cannot choose: if we can apply (K) to a prefix $\sigma.m$ present in $\mathcal{B}$, this prefix is not new. Dually, if we apply a (π)-rule to a branch $\mathcal{B}$ and introduce a new prefix $\sigma.n$, this prefix is obviously not present in $\mathcal{B}$ and we cannot apply (K).

The other case is when two π-formulae, $\sigma : \pi^A(a)$ and $\sigma : \pi^B(a)$ can be reduced in a branch $\mathcal{B}$. After the reduction of $\sigma : \pi^A(a)$ the prefix $\sigma.a.n_1$ is not anymore new. Thus the reduction of $\sigma : \pi^B(a)$ needs to introduce another new prefix $\sigma.a.m_1$. By changing the order we do not get the same result. The partial computation $\mathcal{B}_1 = \mathcal{B} \cup \left\{ \sigma.a.n_1 : \pi_0^A,\ \sigma.a.m_1 : \pi_0^B \right\}$ is potentially different from $\mathcal{B}_2 = \mathcal{B} \cup \left\{ \sigma.a.n_2 : \pi_0^A, \sigma.m_2 : \pi_0^B \right\}$. Then observe that the branches are identical except for a renaming of prefixes and we are done.

Remark. For the logics K45, KD45 and S5, we can choose the calculus with rule (4^π). Then π-formulae must also be reducible more than once.

4.4 Decision Procedures

The propositional modal logics considered here are decidable. So, the next issue is the use of a calculus for designing *decision procedures.*

For simplicity we show the case for modal logic with only one agent a. The techniques can be easily generalized to the multi modal case. Thus, to improve the readability of the discussion we drop the names of agent from both prefixes, accessibility relations and abbreviations for ν and π formulae.

Formula $p \wedge \Box(p \vee \Diamond\neg p) \wedge \Diamond\neg p$ in $\mathsf{KD45}$

Translation clauses	SLD resolvents
$r(U, [X\|V])$.	$\leftarrow p([f, i])$
$p(i)$.	$\leftarrow p([g, f, i])$
$p(W) \leftarrow r([i], W), p([g\|W])$	$\leftarrow p([g, g, f, i])$
$\leftarrow p([f, i])$.	$\leftarrow p([g, g, g, f, i])$
	$\ldots$

Figure 4.2: Infinite Resolvents for Translation Methods

4.4.1 General discussion and comparisons

There are a number of way the proof search in a calculus can be constrained into a decision procedure. The traditional way is to keep the original calculus and constrain the applicability of rules (e.g. Demri [49]). An alternative approach is to revise the calculus, incorporating the constraints directly in the rules themselves (e.g. Hüdelmayer et al. [86]). Both approaches have their advantages and limitations: the former uses a simple calculus by keeping implicit the termination conditions. The second makes termination and the search strategy explicit features of the (new) calculus at the expenses of its clarity and flexibility.

As mentioned in section 1.3 we would like some simple (local) conditions which prevent the application of the rules in the *original* calculus.

This objective seems extremely difficult for *translation methods*. After all, first order logic is not decidable. For functional translations à la Ohlbach [133, 136, 178], the mapping of worlds into terms containing variables makes it possible to generate infinite sets of resolvents even with simple modal formulae. In Fig. 4.2 we show a simple (Horn) satisfiable formula with Nonnegart's translation [133].

The prolog terms $[i]$, $[f, i]$, $[g, f, i]$ are sometimes called world paths [136]. The corresponding intuition is that there is a an $\mathcal{R}$ path in the model such that $w_i \mathcal{R} w_{i:f} \mathcal{R} w_{i:f:g}$ and so on.

Still, not everything is lost. For $\mathsf{S5}$, Wallen [178] has shown a technique to bound the proof search in the matrix method using Hintikka multiplicities of variables. Intuitively the multiplicity is just the number of times a variable can be used. A similar technique which gives an exponential upper bound on the multiplicities has been developed for the case of $\mathsf{S4}$ and K by Cerrito & Cialdea [24] and by Schmidt [157]. We shall see how a simple technique, a local restriction on first order terms corresponding to worlds, makes translation methods decidable.

To design decision procedure for *direct methods* such as modal resolution [54] or Gentzen-type calculi or tableau [62, 79], one typically uses the property of analyticity. The basic idea, which dates back to Smullyan [163], is that a calculus is *analytic* iff its rules and only uses formulae in Subfml($\mathtt{f}.A$) if A is the formula we want to prove valid. In the case of logical consequence we admit also subformulae of the set of local and global assumptions U and G.

If every analytic rule introduces in the consequents only proper subformulae of the formulae of the antecedent, then technique T.3.1 is the only thing we need to terminate. This is the case for Gentzen-type calculi and the related tableau methods for the logic K, *if we restrict ourselves to validity*, i.e. with an empty set of global assumptions.

In the general case, we must use *loop checking* as done by Demri [49] or Goré [79]: before applying any rule we check that we haven't applied it already to the same antecedent in the course of the proof. If the calculus is analytic, this techniques guarantees the termination of the proof search [49, 79].

A weaker property corresponds to the notion of Fischer-Ladner closure for dynamic logics [60]: we can use a *finite superset* of the formulae contained in Subfml($\{\, \mathtt{t}.B | B \in U$ or $B \in G \,\} \cup \{\, \mathtt{f}.A \,\}$). Then a calculus is *super-analytic* iff the consequent of any rule uses only signed formulae in that superset [79]. Using loop checking, this is sufficient for termination.

Implementations must store the trace of previous computations and, more or less often, verify whether a rule has been already applied. This process may be extremely expensive [49].

It is possible to compile the loop checking technique into the rules of the calculus, as done by Hüdelmayer et al. [86]: we add a constraint corresponding to previous application of a rule and then modify the rules so that they propagate both formulae and constraints. This approach doesn't avoid the problem of storing relevant information about previous computations: it simply stores this information in the constraints.

An alternative approach, proposed by Cerrito & Cialdea [25] is to translate each logic into an easier logic, such as K, for which blind search always terminates. The difficult part of this techniques is crafting a polynomial translation, which may vary from logic to logic.

Designing decision procedures for *prefixed tableaux* is easier than translation methods (there are no variables) but harder than direct methods (we have "extra" prefixes).

If we neglect prefixes, Fitting's tableaux [62, Chap.8] are analytic and SST are also analytic with the exception of the logics with the rule (Cxt). If the rule (Cxt) is not coupled with rule (4) or (4^D) than it is easy to prove that SST are super-analytic w.r.t. the set $S^* = S \cup \Box S \cup \ldots \Box^d S$ where S is the set of subformulae of $G \cup \{\, \neg A \,\}$ and d is the maximum modal nesting of formulae in S. In case of $\mathsf{K5}$ the calculus remains super-analytic if we prove the *once-off* property as in [62]: rule (Cxt) can be applied only once. Indeed this is the case after we have restricted the application of rule (Cxt) to $\sigma : \nu(a)$ to prefixes with $\mathrm{last}(\sigma) \neq a$. In the case of the single agent case this corresponds to say that $\sigma = 1$. The superset we are looking for is $S^* = S \cup \Box S$ (see also [79]).

4.4.2 Incorporating 'loop checking' in SST

The intuition behind loop-checking is stopping the search whenever two prefixes are "a different name for the same state". We can revise the definition of completed branches to incorporate this idea.

We say that a prefix σ is a *copy* of a prefix σ_0 for branch $\mathcal{B}$ if for every formula φ it is $\sigma : \varphi \in \mathcal{B}$ if and only if $\sigma_0 : \varphi \in \mathcal{B}$. The intuition is that infinite computations can be avoided if π-rules are not applied to formulae belonging to copies.

Definition 4.10 *A prefixed formula $\sigma : \pi(a)$ is implicitly reduced if there is a copy σ_0 of σ in $\mathcal{B}$ such that $\sigma_0 : \pi(a)$ is reduced. All other prefixed formulae are implicitly reduced iff they are reduced. A prefix is implicitly reduced if all prefixed formulae with that prefix are implicitly reduced.*

Definition 4.11 *A* branch *$\mathcal{B}$ is* π-completed *if the prefix* 1 *is completed for U, and all prefixes in $\mathcal{B}$ are completed for G and implicitly reduced in $\mathcal{B}$.*

We can reformulate Thm. 3.21 as follows:

Theorem 4.12 *If the L-tableau with local assumptions U and global assumptions G starting with $1 : \mathtt{t}.A$ terminates with a π-completed and not closed branch then A is L-satisfiable for U and G.*

The extension of the completeness proof to π/completed branches of can be done along the same lines of the completeness proofs in (§5.6) for prefixed tableaux or by Goré [79] for completeness via model graphs.

Then we use the following technique:

Technique 4.2 *Apply a rule to a formula φ in $\mathcal{B}$ only if the formula is* not implicitly reduced.

Any complete search strategy must now prove that it always terminate into a π-completed branch. A simple strategy that works in combination with technique T.4.2 is the following:

Technique 4.3 *Select applicable rules which introduce prefixed formulae with the shortest prefix.*

This is the loop-checking method by Demri [49].

Remark. The difference between SST and Gentzen-type tableau method is that SST are proof confluent. We do not need to backtrack if we find a loop, we can leave the "copies" unaltered and focus on other parts of the branch.

The condition of π-completeness can be sharpened with the notion of *modal copy*: σ is a modal copy of σ_0 in $\mathcal{B}$ if it has the same ν formulae. Then we can use the notion of modal copy in the definition of implicitly reduced formulae (Def 4.10) and consequently use the notion of *π-modal completed branches* upgrading the corresponding definition (Def. 4.11).

Strategies which guarantee π-modal-completeness are complete for the logics K, KD, T, $\mathsf{K4}$, $\mathsf{KD4}$, $\mathsf{S4}$. We can shorten a branch without loosing π-modal-completeness. First we introduce the *forward tree* of prefixes rooted at σ in branch $\mathcal{B}$:

$$\mathrm{Ftree}(\sigma) \quad = \quad \{\, \sigma^* : \varphi | \sigma^* : \varphi \in \mathcal{B} \text{ and } \sigma \text{ is an initial subsequence of } \sigma^* \,\}$$

The prefix σ is in $\mathrm{Ftree}(\sigma)$, since σ is an initial subsequence of itself.

Second, we say that a branch $\mathcal{B}$ is *pruned* into $\mathcal{B}/\mathrm{Ftree}(\sigma)$ if the set of prefixed formulae $\mathcal{B}/\mathrm{Ftree}(\sigma)$ is obtained from $\mathcal{B}$ by deleting all prefixed formulae in $\mathrm{Ftree}(\sigma)$. Third, the prefixed formula $\sigma : \pi$ is *fulfilled* by $\sigma.n$ in $\mathcal{B}$ if $\sigma : \pi$ is reduced for rule (π) in $\mathcal{B}$ because $\sigma.n : \pi_0$ is in $\mathcal{B}$ (see Def. 3.5). The result we are looking for is the following:

Lemma 4.13 (Pruning Lemma) *Let $\mathcal{B}$ be a branch and $\sigma.n$ a prefix such that for every $\sigma : \pi$ which is fulfilled by $\sigma.n$ there is shorter modal copy σ_0 such $\sigma_0 : \pi$ is reduced in $\mathcal{B}$. If $\mathcal{B}$ is π-modal-completed then $\mathcal{B}/Ftree(\sigma.n)$ is π-modal-completed.*

Proof. This operation only changes the status of the π- formulae $\sigma : \pi$ in $\mathcal{B}/\mathrm{Ftree}(\sigma.n)$. By definition of modal completeness, either $\sigma_0 : \pi$ is reduced or there must be a shorter modal copy σ_{00} such that $\sigma_{00} : \pi$ is reduced. By hypothesis σ is a modal copy of σ_0 and in either cases we have a reduced modal copy of the prefix σ. $\square$

Intuition. With π/modal-completeness we have a sort of minimal requirement: when pruning a branch that satisfies the preconditions of the pruning lemma, we are just deleting useless reductions.

4.4.3 Decidability without loop checking

Loop checking is a very expensive method and does not match our second objective (Problem 1.5): a simple technique on the original rules of the calculus which checks only the formula to be reduced.

Table 4.1: Input Size Parameters

P.	Definition
d	max. nesting of modal operators in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in U\,\} \cup \{\,\mathtt{f}.A\,\})$
n	number of ν-subformulae in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in U\,\} \cup \{\,\mathtt{f}.A\,\})$
p	number of π-subformulae in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in U\,\} \cup \{\,\mathtt{f}.A\,\})$
d_p	the maximum nesting of π subformulae not under the scope of a ν subformulae in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in U\,\} \cup \{\,\mathtt{f}.A\,\})$
p_G	number of π-subformulae in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in G\,\})$
n_G	number of ν subformulae in $\mathrm{Subfml}(\{\,\mathtt{t}.B \mid B \in G\,\})$
f_G	number of formulae of G.

Table 4.2: Bounds for decidability checks

Logic L	Bound hb_{L}
K, D, T, KB, KDB, B	$1 + d$
K4, KD4, S4	$2 + d_p + p \times n$

In the rest of the section, we identify such simple techniques when the set of global assumptions is empty and extend it to translations methods. We focus on *validity* or at most local logical consequence.

We introduce some parameters to evaluate the size of G, U and A in Table 4.1. The measure d is also called the "modal depth" of the formula. In $\mathtt{t}.\Box(A \wedge \Diamond\Diamond B) \vee \Diamond C$ we have that $d = 3$, $n = 1$, $p = 3$ and $d_p = 1$. It is $p = 3$ because we must count both $\Diamond\Diamond B$ and $\Diamond B$.

The intuition is that for each logic L, there is a height bound hb_{L} on the length of prefixes in a branch after which either there are no more modal operators or formulae just repeat themselves with a longer prefix.

Technique 4.4 *For every logic L the application of the single step tableau π-rule is limited to prefixes whose length is less than hb_{L}.*

This is exactly the kind of *local termination check* we are looking for: it requires to look only to single formulae and, if hb_{L} is polynomial in the size of the input formula, it can also be done efficiently. In Table 4.2 we give the upper bound for a first set of logics:

It is straightforward to prove the following property:

Lemma 4.14 *For local L-logical consequence $\models_{\mathsf{L}} U \Rightarrow A$ in the logic L every strategy which applies techniques T.4.2 and T.4.4 with the bound in Table 4.2 terminates.*

The difficult part is proving that we preserve completeness. This proof is extremely difficult for Gentzen-type tableaux (see e.g. [25, 86]). The major problem is that we cannot universally quantify over strategies because the calculus is not proof confluent and there are incomplete strategies (§4.3). As a consequence, termination proofs have an existential flavor "if there is a proof, then there is a proof such that...".

Thanks to proof confluence, we can prove something more general (the full proof can be found in appendix A.2):

Theorem 4.15 *Let* L *be any of the logics* K*,* KD*,* T*,* KB*,* KDB*, and* B*. For the local* L*-logical consequence* $\models U \Rightarrow A$ *every strategy which applies techniques T.4.2 and T.4.4 with the bound of Table 4.2 terminates in one of the following conditions:*

1. *either a tableaux proof has been found,*
2. *or in every branch some rule is still applicable,*
3. *or at least a branch is* π*/completed.*

Theorem 4.16 *Let* L *be any of the logics* $\mathsf{K4}$*,* $\mathsf{KD4}$*,* $\mathsf{S4}$*. For the local* L*-logical consequence* $\models U \Rightarrow A$ *every strategy which applies techniques T.4.2 and T.4.4 with the bound of Table 4.2 terminates in one of the following conditions:*

1. *either a tableaux proof has been found,*
2. *or in every branch some rule is still applicable,*
3. *or at least a branch can be pruned into a* π *modal completed branch.*

The only difficult case is when the strategy terminates and no rule is applicable in at least a branch $\mathcal{B}$. We have to prove that the branch $\mathcal{B}$ can be pruned into a π modal completed branch. The proof goes through the following steps:

- prove that if $\sigma : \pi$ is not reduced then σ must be long hb_{L} (all other π-formulae are reduced);
- prune subtrees of prefixes until a branch $\mathcal{B}'$ is found which no longer satisfies the preconditions of the pruning lemma;
- show that the longest chain of prefixes 1, $1.n_1$ up to $1.n_1 \ldots n_k$ which does not trigger the pruning lemma is such that $k \leq hb_{\mathsf{L}}$.

The logics K,D,T,KB,KDB, and B can be treated without difficulties while $\mathsf{K4}$ and $\mathsf{S4}$ require more work, especially for the third step. We use the following properties of branches reduced for rule (4):

Proposition 4.17 *If the prefix* σ_0 *is an initial subsequence of* σ *in the branch* $\mathcal{B}$ *then* $\sigma_0 : \nu \in \mathcal{B}$ *implies* $\sigma : \nu \in \mathcal{B}$*.*

Proposition 4.18 *If every prefix in the sequence σ, $\sigma.n_1$ up to $\sigma.n_1 \ldots .n_k$ has the same ν-formulae then each $\sigma.n_1 \ldots .n_i$ has been generated by a different π formula.*

Given these two results, a counting argument shows that the longest sequence of prefixes that does not trigger the pruning lemma has exactly length $hb_{\mathsf{L}} = 2 + d + n \times p$.

Remark. We do not actually prune the branch when the proof search terminates. It is enough to know that we can do it. The Gentzen calculus of Hüdelmayer et al. [86] corresponds to a visit of the tree of prefixes directly in the pruned branch $\mathcal{B}'$.

We have *replaced loop checking* for SST with a simple and local check.

4.4.4 Decidability for translation methods

This result can be easily extended to the *free variable SST* proposed by Beckert and Gorè [10] and to other methods. It gives a direct decidability check for functional [135, 136] and mixed [133] translation methods.

Observe that there is a 1-1 mapping between prefixes in (single step) prefixed tableaux and ground terms with functional or mixed translation methods [159, 178] for the logics K, KD, T, $\mathsf{K4}$, $\mathsf{KD4}$ and $\mathsf{S4}$. Use the following technique:

Technique 4.5 *Delete resolvents with terms corresponding to worlds paths longer than hb_{L} for a logic L among K, KD, T, $\mathsf{K4}$, $\mathsf{KD4}$ and $\mathsf{S4}$.*

This is particularly effective for methods which directly employ resolution. It is also much simpler to implement than the exponential upper bound on multiplicity derived in [25].

Combining Theorem 4.16 with the mapping in [159, 178, 25] we have

Theorem 4.19 *Resolution with technique T.4.5 is a decision procedure for the modal logics K, KD, T, $\mathsf{K4}$, $\mathsf{KD4}$, $\mathsf{S4}$, with the functional and mixed translation methods.*

An $\mathsf{S4}$ model is also an $\mathsf{S5}$ model. So, this technique also works for $\mathsf{K45}$ and $\mathsf{S5}$. In the example at the beginning of the section, we have that $hb_{\mathsf{S4}} = 2 + 1 + 2 \times 1 = 5$ and thus we can terminate after few resolvents.

4.5 Complexity and Search Strategies

After the design of a decision procedure for validity using a calculus, we can ask how efficient it is.

The first step is trying to prove that we can efficiently upgrade this procedure to solve global logical consequence. The second step is finding further restrictions on the proof search to match the theoretical complexity bounds. We recall that deciding validity is a PSPACE-complete problem for modal logics between K and S4 [98] and deciding logical consequence is EXPTIME-complete [82]). For K45, KD45, and S5 deciding validity is "only" CO-NPTIME-complete [82]

We need some preliminary observations on the size of the smallest model that can satisfy a given formula [82].

Fact 4.20 *Let* L *be a logic for which deciding satisfiability is an* NPTIME-*complete problem. For every* L*-satisfiable formula* A *there is an* L*-model for* A *such that its size (number of worlds) is polynomially bounded in the size of* A.

Fact 4.21 *Let* L *be a logic for which deciding satisfiability is a* PSPACE-*complete problem. There are* L*-satisfiable formulae* A *such that the size (number of worlds) of the smallest* L*-model for* A *is exponential in the size of* A.

Fact 4.22 *Let* L *be a logic for which deciding satisfiability is a* PSPACE-*complete problem. For every* L*-satisfiable formula* A*, there is an* L*-model* $\langle W, \mathcal{R}, \mathcal{I}\rangle$ *for* A *where the length of the longest simple* $\mathcal{R}$*-path*[2] *is bounded by a polynomial in the size of* A.

The termination check in section 4.4 is a reformulation of this property. Theorem 4.16 simply says that the length of the longest $\mathcal{R}$-path is hb_{L}.

Fact 4.23 *Let* L *be a logic for which deciding satisfiability is an* EXPTIME-*complete problem. There are formulae* A *such that* $G \not\models_{\mathsf{L}} A$ *and that the length of the longest simple* $\mathcal{R}$*-path in the smallest* L*-countermodel for* A *is exponential in the size of* A *and* G.

These facts have a direct impact on the complexity of the proof search, in particular for calculi which work by refuting the theorem, i.e. by trying to construct a countermodel.

With an NPTIME-complete problem we can generate the countermodels: we "know" that some of them are small (although it may take exponential time to find one or prove that there is none).

With a PSPACE-complete problem we cannot generate whole models since there are formulae which have exponentially large models. Still, we "know" that we can visit some of them using polynomial space.

[2]A model $\langle W, \mathcal{R}, \mathcal{I}\rangle$ is simply a directed graph. A simple path is a sequence of worlds $w_1, \ldots w_n$ without repetitions such that $w_i \mathcal{R} w_{i+1}$.

Remark. If the completeness of our proof procedure requires the generation of the whole model, the corresponding algorithm will take *exponential space* rather than polynomial space in the size of the input.

4.5.1 Dealing with logical consequence

We can use the notion of π-completed branches and techniques T.4.2 and T.4.3 to derive an upper bound on the maximum lenght of a prefix in a closed or open tableau.

Consider modal logic K. Notice that for every prefix we can only introduce a formula in $\mathrm{Subfml}(\{\,\mathtt{t}.B | B \in U \cup G\,\} \cup \{\,\mathtt{f}.A\,\})$. Let n be the number of formulae of this set (this is also a reasonable measure of the input), then we can have at most 2^n different sets. It is easy to prove the following upper bound:

Proposition 4.24 *Let* L *be any of the logics* K, KD, T, KDB, KDB, B, $\mathsf{K4}$, $\mathsf{K45}$, $\mathsf{KD45}$, $\mathsf{S5}$. *Every search strategy which applies Techniques T.4.2 and T.4.3 for logical consequence* $G \models_{\mathsf{L}} U \Rightarrow A$ *cannot generate prefixes longer that* $2^n + 2$.

Observe that after at most 2^n π-steps we must reintroduce the same set of subformulae of a previously constructed prefix and the shorter reduced copy triggers technique T.4.2.

We would like something better than this. For instance, the extension of technique T.4.4 and T.4.5 to logical consequence seems fairly easy using the following procedure:

1. keep the rules and the strategy of the calculus for validity (whether direct or based on translation into first order logic);
2. apply the modal deduction theorem and transform (global) logical consequence into validity (local logical consequence);
3. recompute the upper bounds for the resulting problem;
4. use the decision procedure with the new bounds.

The key question is how efficient is this procedure. Reconsider again the modal deduction theorem [62, 64]:

$$G \models_{\mathsf{L}} U \Rightarrow A \quad \text{iff} \quad \text{for some } n \text{ it is } \models_{\mathsf{L}} U \cup \bigcup_{i=0}^{n} \Box^i G \Rightarrow A \tag{4.1}$$

where $\Box^0 = G$ and $\Box^{i+1} G = \{\,\Box A \mid A \in \Box^i A\,\}$.

We can reformulate the question in term of the value of n. If the value of n is polynomially bounded in the size of G, U and A, then this can be a feasible approach. If n is not polynomially bounded, then the decision procedure could receive an intermediate input with an exponential blow-up.

One may argue that the translation used in the deduction theorem is too naive. For instance, the decision procedure for S4 by Cerrito & Cialdea [25] translates S4 into K using two steps: first it generates an S4-satisfiability preserving modal CNF with the techniques of Mints [128] and then uses a naive translation of S4 into K. The naive translation would yield an exponential blow-up but the preprocessing guarantees a polynomial translation. Unfortunately, clever translations are not possible:

Theorem 4.25 *For logics* K*,* KD*,* T*,* KB*,* KDB*, and* B *a polynomial translation of logical consequence into validity implies* PSPACE=EXPTIME*.*

This is an obvious consequence of the fact that validity for K is PSPACE-complete whereas logical consequence is EXPTIME-complete.

Corollary 4.26 *For the logics* K*,* KD*,* T*,* KB*,* KDB*, and* B *the value of n in (4.1) cannot be bounded by any polynomial in the size of U, G and A.*

The proof (appendix A.2) stems from the preceding theorem by noting that the deduction theorem is 'monotone' in n: if the theorem holds for some $n = n_0$ then it holds for $n = n_0 + 1$. If a polynomial bound exists we could chose the value of the bound for the value of n.

These negative results can be extended to the decision procedures for SST and functional/mixed translation methods:

Corollary 4.27 *For logical consequence $G \models_{\mathsf{L}} U \Rightarrow A$ in the logics* K*,* KD*,* T*,* KB*,* KDB*, and* B *the value of hb_{L} cannot be bounded by any polynomial in the size of G, U and A.*

If an polynomial upper bound hb_{L} existed we could simply set $n = hb_{\mathsf{L}}$ in the modal deduction theorem (4.1).

The situation is better for K4, KD4, and S4.

Theorem 4.28 *For the global logical consequence $G \models_{\mathsf{L}} U \Rightarrow A$ in logics* K4 *and* S4 *every strategy which applies technique T.4.4 with $hb_{\mathsf{L}} = 2 + (d_p + p_G) + (p + p_G) \times (n + n_G + f_G)$ terminates in one of the following conditions:*

1. *either a tableau proof has been found,*
2. *or in every branch some rule is still applicable,*
3. *or at least a branch can be pruned into a π modal completed branch.*

Logics with the axiom $\Box A \supset \Box\Box A$ satisfy a particular form of the deduction theorem [89, 62, 64]: $G \models_{\mathsf{L}} U \Rightarrow A$ if and only if $\models_{\mathsf{L}} G \cup \Box G \cup U \Rightarrow A$.

Apply Theorem 4.16 to the new set $U' = U \cup G \cup \Box G$ and we are done.

4.5.2 NPTIME-search strategies for K45 and S5 with SST

From the preliminary facts we have recalled, proof search for logics like K45 and S5 should be much simpler than for other logics.

Yet, the presence of symmetry makes things harder for most direct methods, either Gentzen-type tableaux or modal resolution. Cut is necessary when faced with local logical consequence for arbitrary formulae. The prototypical example is $\models_{\mathsf{S5}} \{A, B\} \Rightarrow \Box\Diamond(A \wedge B)$. The common solution [54, 89, 137, 138] is to use an equivalence-preserving preprocessing step which reduces the depth of the modal connectives. Then cut-free calculi for the reduced formula can be found. The problem is that the preprocessing step can lead to an exponential blow-up. This can be partly avoided using a further preprocessing with the definitional translation into modal CNF by Mints [128]. The immediacy of the translation is now lost, and still this procedure requires at least a quadratic increase in size. An alternative approach is the translation into S4 used by Demri [49]. This means translating a simpler problem into an harder one.

For translation methods, the main result is by Wallen [178]. It gives a linear upper bound on the Hintikka multiplicities[3] of every first order variable.

In §3.3 we have two sets of rules for each of K45, KD45, and S5. A simple technique to restrict rule π yields the desired NPTIME-bound.

Technique 4.6 *With the calculus using rule* (4) *for* K45, KD45, *and* S5, *apply rule* (π) *once to each* π *formula (no matter its prefix).*

Technique 4.7 *With the calculus using rule* (4^π) *for* K45, KD45, *and* S5, *apply rule* (π) *only to the prefix* $\sigma = 1$.

Each technique gives a decision procedure when combined with technique T.4.2 and provides an equivalent of Theorem 4.16. (Proofs are in appendix A.2).

Lemma 4.29 *For logical consequence* $G \models U \Rightarrow A$ *in logics* K45, KD45, *and* S5 *every strategy which applies technique T.4.2 and T.4.6 (T.4.7) terminates in one of the following conditions:*

1. *either a tableau proof has been found,*
2. *or in every branch some rule is still applicable,*
3. *or at least a branch is can be extended to a* π*-modal-completed branch.*

We can also show that this optimal:

Theorem 4.30 *For logical consequence* $G \models U \Rightarrow A$ *in logics* K45, KD45, *and* S5 *every strategy with technique T.4.6 (T.4.7) generates SST branches with size polynomial in the size* G, U *and* A.

[3] Loosely speaking the number of times we need to instantiate a variables.

We simply need to observe that in every branch we can generate a number of prefixes at most equal to the number of π-subformulae of $\mathrm{Subfml}(\{\,\mathtt{f}.A\,\} \cup \{\,\mathtt{t}.B \mid B \in U \cup G\,\})$.

Moreover, in the model constructed by the completeness theorem, the number of worlds is also *linearly* bounded by the number of π-formulae. This corresponds to the lowest possible bound. We can extend this technique to translation methods:

Technique 4.8 *Let* L *be one of the logics* K45, KD45, *and* S5, *the first order terms corresponding to worlds must contain at most one occurrence of each skolem function/constant corresponding to a* π*-formula.*

In Fig. 4.2, we can now stop after the generation of three resolvents. Resolution with technique 4.8 is a decision procedure for K45, KD45, S5.

4.5.3 Space complexity of different calculi

The case for the logics K, KD, T, K4, KD4, and S4 is more interesting because deciding validity is PSPACE-complete and therefore we may have formulae only with "large" models.

Gentzen-type calculi require only polynomial space if we use loop checking with backtracking as in [49, 98, 82]. This works if the calculus actually *deletes* the formulae before backtracking to a previous stage. Loosely speaking we can describe this depth-first search as follows:

1. start from a set of formulae;
2. reduce propositional connectives (possibly creating new branches);
3. if we get an inconsistent set then stop and pass to a new branch;
4. choose a π-formulae and save the remaining choiches in the stack;
5. reduce the π-formula generating a new starting set;
6. if the same set has been already generated then delete the set and backtrack; otherwise go to 1.

This strategy guarantees the use of polynomial space but has its disadvantages. The worst problem is the reduction of disjunctions before π-formulae to retain completeness (see §4.3). In practice, this forces a lot of (probaly pointless) case analysis [90].

More space, but still a polynomial amount, is required for the calculus by Hüdelmayer et al. [86]. Indeed the major difference is that the loop checking mechanism is compiled into the rules. We can now check for loops without looking back in the stack. Since we are simply encoding a part of the stack in each node nodes are bigger.

Remark. The usage of polynomial space is not a characteristic of Gentzen system. It is so because depth-first search is considered the 'natural' way to search for a proof in Gentzen-type calculi. If we use a breadth-first search for visiting choice points of π-formulae, then we need exponential space.

Tableau methods where the accessibility relation is used explicitly, as done by Catach [23] or Kripke [96], are bound to take exponential space: for "bad formulae" they create the exponential models.

For the same reasons, *relational translation methods* based on resolution or similar saturation methods proposed by Gent [69] require exponential space. The key observation is that the completeness of first order resolution forces the addition of (non subsumed) resolvents to the current set of clauses. If no contradiction is found, the final set of clauses describes the accessibility relation between worlds of the (counter)model. This model may have exponential size (see Fact 4.21 above).

Functional translation methods à la Ohlbach are better, since the accessibility relation between worlds is embedded in the unification theory. Therefore, we do not need to store (and not even generate) the clauses describing the accessibility relation. This is also true for the mixed translation method by Nonnegart [133] where all literals describing the accessibility relation can be resolved away.

Yet, we have the literals describing properties of worlds (recall that a term is just a name for a world). Again, the saturated set of clauses for satisfiable formulae must describe all these worlds and these may be in exponential number. The conclusion is the following:

Fact 4.31 *For relational, functional, and mixed translation methods any resolution strategy complete for full first order logic requires worst case exponential space.*

This may seem strange given the experimental results of Hustadt & Schmidt [90] where resolution outperforms tableaux. Again, the caveat is that we may be simply comparing the efficiency of implementations, or using benchmarks which are not PSPACE-hard.

It is, of course, possible to recover polynomial space by ad-hoc mechanisms for deduction in modal logics. At a guess, this may be done by a combination of specialized rules for deleting (not-subsumed) clauses and selection functions for the next resolvent. In such a way we could mimic the search strategy of Gentzen-type tableaux. Yet, this would be the very negation of the motivation behind translation methods proposed by Ohlbach [134, page 513]: "there is no need to develop specialized theorem provers for modal logic."

Modal resolution by Enjalbert & Farinas del Cerro [54] also requires exponential space.

The worst case complexity of the matrix proof method by Wallen [178] is unclear. An exponential upper bound has been found by Cerrito & Cialdea [24] but no conclusion regarding its worst case space requirements is known.

The algorithms for *prefixed tableaux* (single step or not) by Fitting [62], Goré [79], and Massacci [113] require exponential space. This is not a surprise: being a saturation method, prefixed tableaux suffer from the same illness of resolution.

Fact 4.32 *Any complete strategy for prefixed (single step) tableaux which never deletes formulae requires worst case exponential space.*

4.5.4 PSPACE-search strategies with SST

To recover a PSPACE algorithm for validity we need two techniques:

- delete formulae as soon as they are not needed anymore;
- block the generation of new prefixes if too many formulae are around.

We know that this is possible: we can always fall back on a depth-first traversal of the tree of prefixes and simulate the depth first strategy of Gentzen-type tableaux. Since the practical (negative) consequences of such choice are known, we would like to exploit proof confluence for a more flexible strategy.

We focus on the logics between K and $\mathsf{S4}$, and then extend the techniques to logics KB, KDB, B. The *backward tree* of a prefix in branch $\mathcal{B}$ is the set

$$\text{Btree}(\sigma) \quad = \quad \{\, \sigma_0 : \varphi \mid \sigma_0 : \varphi \in \mathcal{B} \text{ and } \sigma_0 \text{ is an initial subsequence of } \sigma \,\}$$

We need to introduce the concept of *confined formula*. The intuition is that a confined formula with prefix σ does not interfere with the consistency of formulae having different prefixes. So we can delete the confined formulae as soon as they are reduced.

Let $\mathcal{B}'$ be a set[4] of prefixed formulae. A prefixed formula $\sigma : \pi \in \mathcal{B}'$ is *confined* in $\mathcal{B}'$. All other prefixed formulae $\sigma : \varphi$ are confined if either $\nu \notin \text{Subfml}(\varphi)$ for all ν or φ is reduced in $\mathcal{B}'$ (Def.4.10). A formula $A \in U$ is confined in $\mathcal{B}'$ iff $1 : \mathtt{t}.A$ is confined in $\mathcal{B}'$.

Definition 4.33 *A prefix is* confined *in a branch $\mathcal{B}$ if all formulae in U and all prefixed formulae in Btree(σ) are confined in Btree(σ).*

[4] It is not necessarily a branch, it can also be a subset of it.

A alternative definition is possible: σ is confined if all prefixed formulae $\sigma : \varphi$ are confined and its immediate predecessor is confined (and all U formulae are confined for the prefix 1). Confined prefixes satisfy the following properties:

Lemma 4.34 (Local Stability) *Let σ be a confined prefix in $\mathcal{B}$. No sequence of rules (α), (β), (Loc), (D), (K), (T), (4), (4^D) applied to formulae with prefixes different from σ can introduce a new prefixed formula $\sigma : \varphi$ in $\mathcal{B}$.*

The proof (appendix A.2) is by a double induction on the length of the prefix σ and the length of the sequence. We use this lemma for proving that confined prefixes are preserved by the techniques we have used for decidability.

Proposition 4.35 *A strategy using techniques T.4.2 and T.4.4 cannot introduce a new prefixed formula $\sigma : \varphi$ in a branch $\mathcal{B}$ if σ is a confined and reduced prefix in $\mathcal{B}$.*

Now, choose a parameter $wb \geq 1$ (width bound) equal to some preset constant value[5] and apply the following technique:

Technique 4.9 *For every integer n and every prefix σ of length n, do not apply rule (π) to prefixed formulae $\sigma : \pi$ if there are more than wb unreduced prefixes of length $n + 1$.*

The next step is proving that the unreduced part of each branch has a polynomial size.

Theorem 4.36 *For the logics* K, KD, T, K4, KD4, *and* S4 *every strategy using techniques T.4.2, T.4.4, and T.4.9 the number of unreduced prefixes in a branch is bounded by a polynomial in the size of U and A.*

Proof. By technique T.4.2 the number of (unreduced) prefixed formulae is linearly bounded by the number of subformulae of U and A. By technique T.4.4 the maximum length of prefixes is hb_{L}, which is polynomially bounded in the size of U and A (Thm. 4.16). By technique T.4.9 we have at most wb unreduced prefixes of length n for each $n \geq 1$. The claim follow by a multiplication. □

The polynomial space algorithm we are looking for must use this additional technique (beyond T.4.2, T.4.4, T.4.9) in each branch $\mathcal{B}$:

Technique 4.10 *Delete all prefixed formulae of every confined and reduced prefix in $\mathcal{B}$. If a prefixed formula $\sigma.n : \pi_0$ is deleted in this process then consider the corresponding $\sigma : \pi$ as still reduced.*

[5] In general, it can be a polynomial function $wb \leq poly(|U|, |A|)$.

We may keep deleted prefixes somewhere to exhibit the model but then the auxiliary storage is bound to take exponential space (Fact 4.21).

The combination of these techniques preserves soundness (using Thm. 4.35 and 4.9), completeness (Thm 4.12), provides a decision procedure (Thm. 4.15 and Thm. 4.16) and guarantees the use of polynomial space in each branch (Thm. 4.36). This, without dropping but rather exploiting proof confluence.

We are free to reduce a polynomial number of π formulae before reducing a single disjunctive β-formula. Still we do not lose neither completeness nor the polynomial space bound for the size of a branch.

Moreover, technique T.4.10 is so liberal that the part of the branch we keep in memory may resemble a tree with gaps. For example, the part of the branch we keep in memory can be composed only by the prefixes 1, 1.1.1, 1.1.2 and 1.1.2.1, because the prefix 1.1 has been deleted. Thm. 4.35 guarantees that such gaps are harmless.

For logics with rule (B) these techniques are not sufficient. To be precise, reckless deletion of prefixes may lead to incomplete search strategies, since $\sigma.n : \nu_0$ can be introduce by $\sigma : \nu$ *and* by $\sigma.n.m : \nu$. To prove that a prefix can be safely deleted, it is not enough to look only to shorter prefixes, we must also look to longer ones.

At first we need to revise technique T.4.10:

Technique 4.11 *Delete all prefixed formulae of every confined and reduced prefix σ in $\mathcal{B}$ such that for every $\sigma.n$ the forward tree Ftree$(\sigma.n)$ is empty (has been previously deleted).*

Second, use the following trick:

Technique 4.12 *If a new prefixed formula $\sigma : \nu$ is introduced in $\mathcal{B}$, then for every $\sigma : \pi$ check whether the corresponding $\sigma.n : \pi_0$ has been deleted. If so, change status to $\sigma : \pi$: it must be reduced again.*

Chapter 5

Propositional Dynamic Logic

We extend the treatment of modal logics to propositional dynamic logic with converse, also known as logic of programs.

After an intuitive description of the combination of old and new deduction and model checking techniques employed in this chapter, we present the syntax and the semantics of converse dynamic logic and extend the machinery of prefixed tableaux. For a better understanding of the method, we provide a number of examples before proving its soundness and completeness.

We also discuss the issue of computational complexity, and in particular the design of search algorithms which can be proved to require only EXPTIME.

5.1 Introduction

Propositional Dynamic Logics (PDLs) are multi modal logics introduced by Fisher & Ladner [60] to model the evolution of the computation process by describing the properties of states reached by programs during their execution.

We focus on *Converse*-PDL (CPDL) [60], obtained from the basic logic PDL by adding the converse of a program, which is interpreted as the converse of the (input-output) relation interpreting the program. A possible use of the converse is for formalizing preconditions, e.g. $[P^{-}]\,A$ can be interpreted as "before running program P, property A must hold".

There are certain applications of PDLs where the ability of denoting converse programs is essential. For instance, in the mapping between dynamic logics and expressive knwoledge representation formalisms by Schild [156].

From the formal point of view, CPDL shares many characteristics with the basic PDL, and many results for PDL extend to CPDL without difficulties. For instance the proofs of finite model property for PDL by Fischer & Ladner [60] are easily extended to CPDL, as well as the proof of EXPTIME-completeness by Pratt [146]. Practical inference procedures have been successfully developed for PDL, but their extension to CPDL has proved to be a difficult task and unsuccessful till now (to the best of our knowledge).

To be precise, inference procedures based on model enumeration by Fisher & Ladner [60] or by Pratt [146] or on automata on infinite trees by Vardi & Wolper [174] have been extended to converse of programs. Yet, these procedures are more suited for proving theoretical results than for being used in applications. Tableaux procedures for PDL by Pratt [145, 144], which are typically simpler in practice, have never been extended. In [145], Pratt says "We do not have a practical approach to this difficulty with converse, and our 'practical' procedure therefore does not deal with converse".

In this chapter we extend the prefixed tableaux calculus for modal logics to converse dynamic logic, thus obtaining the first tableaux calculus for CPDL.

5.2 Combining Deduction and Model Checking

The problem we face with PDLs, temporal logics or modal μ-calculus is that they cannot be directly tamed by the methods used for logics of knowledge and belief:

- Some formulae impose fixpoint properties on paths (in the sequel we call them *iterated eventualities or necessities*) so that the traditional local validation of a model is not enough.
- PDLs, like the modal μ-calculus, has an added difficulty w.r.t. temporal logics (which have only one transition, next, and simpler eventualities) since programs can be extremely complicated and deciding whether two states are reached by a program may have the same complexity of the original problem.
- The converse operator (or past for temporal logics) imposes constraints on previously visited states so that these must be re-processed.
- The models of a formula can be exponential in its size and therefore direct depth-first search with backtracking at choice points may lead to a double exponential algorithm (although decidability for these logics is in EXPTIME).

Tableaux approaches for PDLs and expressive temporal logics proposed by Pratt [145, 144] and resp. by Emerson [53] or Manna & Pnueli [107][Sec. 5.1-5.3] cope with these problems in two conceptual phases:

1. Construct an AND-OR *graph* based on the Fischer-Ladner closure which satisfies the constraints of a normal modal logic, where nodes with the

same formulae are identified and nodes that are locally inconsistent are deleted.

2. model check the pseudo-model thus obtained for iterated eventualities (deleting nodes with unfulfilled eventualities).

The separation into two phases is present also in automata techniques where the automaton corresponding to a formula[1] is the product of two automata: one for local modal conditions and one for eventualities [167, 174, 175].

The first intuition is to build the AND-OR graph of the phase 1 in *incremental fashion*, as done by Manna & Pnueli with particles tableaux for temporal logics [107]:

Intuition. A node of the tableau is a set of formulae and new nodes are generated only by reducing already present formulae.

In this way we generate nodes and check for local consistency "on-the-fly". In contrast with maximal model techniques by Pratt [146], tableaux methods generate all subsets of the Fischer-Ladner closure only in the worst cases.

The idea of proceeding in an incremental fashion can be pushed further by executing the two phases at the same time[2]. In this way we need not to wait for the construction of a (large) pseudo-model just to discard it with the first eventuality we checked.

Intuition. Merge the pseudo-model checking stage for iterated eventualities with the incremental tableaux construction.

Thus an algorithm may close a branch due to an unfulfilled eventuality *before* the construction of the whole model, improving its average time behaviour. For temporal logics this problem has been solved by Bhat et al. [16] or Manna & Pnueli [107]: as soon as the same node is repeated along the path and we find again the same eventuality (i.e. the same formula) we conclude that this is a bad (unfulfilling) loop.

For PDL/CPDL there are more difficulties: to conclude that an eventuality of the form $\langle P^* \rangle A$ is not fulfilled, it is not enough to "find again" the same eventuality into a duplicate node. One must verify that the two nodes are connected by some P-steps and P can be a complex program. This problem does not arise in temporal logic because P can only be "next".

The solution developed by Pratt in [144] has been the introduction of an operator $\Rightarrow$ to link every eventuality of the form $\langle P \rangle A$ with the corresponding fulfilling node where A holds. This may lead to a cumbersome tableau

[1] One constructs an automaton that accepts the tree models of the formula and hence a formula is satisfiable iff the automaton accepts some models [53, 174, 175].

[2] There are incremental model checking techniques for temporal logics [16, 15, 107].

and to many unnecessary relations between formulae. Moreover it does not completely address the issue of merging the two phases, because one needs to construct the transitive closure of $\Rightarrow$ and this can be done only after the pseudo-model construction is terminated. The automata theoretic approaches of Vardi & Wolper [174, 175], instead of building such $\Rightarrow$-links, use suitable eventuality automata, which as well require the construction of the pseudo-model first.

There is another way that fully addresses this problem, based on the same intuition of *history variables* by Manna & Pnueli [107] i.e. "auxiliary variables whose values in the current states reflect all the facts we need to know about the past". A related idea is used for automata by Vardi & Wolper [174] by introducing state formulae like $cycle(P)$ to represent the fact that there is a P-loop which passes through that state. This idea has been used for model checking techniques for the modal μ-calculus, introducing new propositional variables (names) for denoting fixpoints by Stirling [165].

If we introduce a name X to denote $\langle P^* \rangle A$ and use the name for subsequent reductions then we have a direct way to decide whether two nodes are connected by some P-steps and therefore use the simple check of temporal logics (repeated eventuality into duplicated nodes).

Intuition. Associate iterated eventualities with new names $X \doteq \langle P^* \rangle A$ and define reduction rules which use the name rather than the formula.

Next we focus on the converse operator. One of the intuition used for automata theoretic techniques by Vardi & Wolper [174] is to use *two-way* tree models where a transition can go from the current state to its child (direct) or vice versa (converse). We combine this idea with the notion of prefixed tableaux developed for modal logics in Chapter 3 to get the following:

Intuition. Associate to each a formula a prefix formed by the sequence of states and direct/converse transitions starting from the initial state and ending in the current state where the formula is supposed to hold. Devise rules which take into account both formulae and prefixes.

In this framework we devise tableaux rules for the converse of an (atomic) program with the techniques for symmetric modal logics B or $\mathsf{S5}$:

Intuition. Necessity-like formulae with direct programs such as $[a]\,A$ move formulae forward when the last prefixed transition is direct and backward when the last transition is converse. $[a^-]$ does the opposite.

The next step, when looking for duplicated nodes, is to take into account what has been also noted for automata by Vardi & Wolper [174]: "we also have to know what program connects this node to its predecessor." With prefixed tableaux this check is immediate: it is embedded into the label of the formulae of each state.

We still need to 'guess' a small subset of formulae that characterises past computations, especially when we are trying to discard unfulfilled eventualities[3]. The existence of such a set of formulae for logics where the past matters, such as modal logic B, was pointed out by Fitting [62].

Intuition. To identify two duplicate nodes, the formulae which compose them must also completely specify the past (the arriving program), so we use analytic cut on a small subset of subformulae to complete the nodes.

Last but not least we want to lower the complexity of the tableaux procedure from NEXPTIME to EXPTIME. The key point is that to avoid exponential behaviour on average it is not enough to store only the visited nodes of a branch as done by [144]. We must also track the nodes already proven to be inconsistent as proposed by Bhat et al. [16].

Intuition. Use a global data structure to store "bad" sets, which have been proven to lead to contradiction in previous expansion of the tableau. Before expanding a node always check that it is not a previously seen "bad" set.

5.3 Syntax and Semantics

We sketch some basic notions on CPDL [95, 165]. If $\mathcal{A}$ is a set of direct atomic programs (a, b etc.), $\mathcal{P}rop$ a set of propositional letters (p, q etc.), the formulae A, B and the programs P, Q of CPDL are

$$\begin{array}{lcl} A, B & ::= & p \mid \neg A \mid A \wedge B \mid A \vee B \mid \langle P \rangle A \mid [P]\, A \\ P, Q & ::= & a \mid P; Q \mid P \cup Q \mid P^* \mid P^- \mid A? \\ \varphi, \psi & ::= & \mathtt{t}.A \mid \mathtt{f}.A \end{array}$$

Other connectives can be seen as abbreviations. W.l.o.g. we restrict the converse operator to atomic programs, by using equivalences such as $(P;Q)^- \equiv (Q^-;P^-)$ or $(P^*)^- \equiv (P^-)^*$ etc. The metavariable $\mathbf{a}$ denotes either a direct or converse atomic program (considering $(a^-)^- \doteq a$) and we refer to it as an *atomic program*, without specifying direct or converse.

Signed formulae make possible to use the compact notation for propositional and modal rules. See Table 2.1 and Table 3.1 for standard connectives. The only difference for the ν and π formulae is that we have $\nu(\mathbf{a})$ and $\pi(\mathbf{a})$ rather than simply $\nu(a)$ and $\pi(a)$. Table 5.1 shows how the notation is used for and, not, sequence, choice and test. Mutatis mutandis, we follow [107] in the classification of PDL-like connectives.

[3]Remember that in this case we are trying to prove that an eventuality has not been fulfilled and never will be (no matter how we continue the computation).

α	α_1	α_2	β	β_1	β_2
$\mathtt{t}.\langle B?\rangle A$	$\mathtt{t}.B$	$\mathtt{t}.A$	$\mathtt{f}.\langle B?\rangle A$	$\mathtt{f}.B$	$\mathtt{f}.A$
$\mathtt{f}.\langle P\cup Q\rangle A$	$\mathtt{f}.\langle P\rangle A$	$\mathtt{f}.\langle Q\rangle A$	$\mathtt{t}.\langle P\cup Q\rangle A$	$\mathtt{t}.\langle P\rangle A$	$\mathtt{t}.\langle Q\rangle A$
$\mathtt{f}.[B?]\,A$	$\mathtt{f}.B$	$\mathtt{f}.A$	$\mathtt{t}.[B?]\,A$	$\mathtt{t}.B$	$\mathtt{t}.A$
$\mathtt{t}.[P\cup Q]\,A$	$\mathtt{t}.[P]\,A$	$\mathtt{t}.[Q]\,A$	$\mathtt{f}.[P\cup Q]\,A$	$\mathtt{f}.[P]\,A$	$\mathtt{f}.[Q]\,A$

neg	*pos*
$\mathtt{f}.\langle P;Q\rangle A$	$\mathtt{f}.\langle P\rangle\langle Q\rangle A$
$\mathtt{t}.\langle P;Q\rangle A$	$\mathtt{t}.\langle P\rangle\langle Q\rangle A$
$\mathtt{f}.[P;Q]\,A$	$\mathtt{f}.[P]\,[Q]\,A$
$\mathtt{t}.[P;Q]\,A$	$\mathtt{t}.[P]\,[Q]\,A$

Table 5.1: Uniform Notation for PDL

PDL's semantics is based on Kripke structures [95]: a model is a pair $\mathcal{M} = \langle W, \mathcal{I}\rangle$ where W is a non empty set of states and $()^{\mathcal{I}}$ an interpretation s.t. for every direct atomic program a it is $(a)^{\mathcal{I}} \subseteq W \times W$ and for every propositional letter P it is $(P)^{\mathcal{I}} \subseteq W$. The relation $(a)^{\mathcal{I}}$ corresponds to the relation $\mathcal{R}_a$ of modal logics (§3.2). The interpretation $()^{\mathcal{I}}$ is extended as follows:

$$
\begin{aligned}
(A \wedge B)^{\mathcal{I}} &= (A)^{\mathcal{I}} \cap (B)^{\mathcal{I}} \qquad (\neg A)^{\mathcal{I}} = W - (A)^{\mathcal{I}} \\
(\langle P\rangle A)^{\mathcal{I}} &= \left\{ w \mid \exists v \in W \text{ s.t. } \langle w,v\rangle \in (P)^{\mathcal{I}} \text{ and } v \in (A)^{\mathcal{I}} \right\} \\
(P;Q)^{\mathcal{I}} &= \left\{ \langle w,v\rangle \mid \exists u\, \langle w,u\rangle \in (P)^{\mathcal{I}} \text{ and } \langle u,v\rangle \in (Q)^{\mathcal{I}} \right\} \\
(P \cup Q)^{\mathcal{I}} &= (P)^{\mathcal{I}} \cup (Q)^{\mathcal{I}} \\
(P^*)^{\mathcal{I}} &= \text{reflexive and transitive closure of } (P)^{\mathcal{I}} \\
(P^-)^{\mathcal{I}} &= \left\{ \langle v,w\rangle \mid \langle w,v\rangle \in (P)^{\mathcal{I}} \right\} \\
(A?)^{\mathcal{I}} &= \left\{ \langle w,w\rangle \mid s \in (A)^{\mathcal{I}} \right\}
\end{aligned}
$$

In the sequel, we write $\mathcal{M}, w \Vdash A$ for $w \in (A)^{\mathcal{I}}$ and extend the notion of validity to signed formulae in the usual way:

$$
\begin{aligned}
\mathcal{M}, w \Vdash \mathtt{t}.A &\quad \text{iff} \quad \mathcal{M}, w \Vdash A \\
\mathcal{M}, w \Vdash \mathtt{f}.A &\quad \text{iff} \quad \mathcal{M}, w \nVdash A
\end{aligned}
$$

The notion of validity and logical consequence can be defined in the same way of modal logics of knowledge and belief (§3.2). When $\mathcal{M}, w \Vdash \varphi$ holds we say that w satisfies φ in $\mathcal{M}$. If S is a set of formulae then a world w satisfies S iff for all $A \in S$ one has $\mathcal{M}, w \Vdash \mathtt{t}.A$. A model $\langle W, \mathcal{I}\rangle$ *satisfies* S iff every world in W satisfies S.

Definition 5.1 *It is* logical consequence *of global assumptions G and local assumptions U iff in every model $\langle W, \mathcal{I}\rangle$ which satisfies G if a world w satisfies U then w satisfies A.*

The *Fischer-Ladner closure* of a formula C [60, 95] is an extension of the subformula closure we have already defined for propositional and modal logics (S2.2 and §3.2) taking into account the iteration operator:

- if $\mathtt{t}.\langle P^*\rangle A \in \text{Subfml}(\varphi)$ then $\mathtt{t}.\langle P\rangle\langle P^*\rangle A, \mathtt{t}.A \in \text{Subfml}(\varphi)$.
- if $\mathtt{f}.\langle P^*\rangle A \in \text{Subfml}(\varphi)$ then $\mathtt{f}.\langle P\rangle\langle P^*\rangle A, \mathtt{f}.A \in \text{Subfml}(\varphi)$.

The cases for $[P^*]\,A$ are substantially identical. For completeness we must also add the following condition:

- if $\psi \in \text{Subfml}(\varphi)$ then $\overline{\psi} \in \text{Subfml}(\varphi)$.

To establish the truth value of a formula φ in a model it is sufficient to check the value of the formulae in $\text{Subfml}(\varphi) \cup \text{Subfml}(\overline{\varphi})$ for every state of the model [60, 95]. Both number and size of the formulae in $\text{Subfml}(\varphi)$ are linearly bounded by the size of φ.

In the case of CPDL, the notion of logical consequence can be incorporated into the notion of validity by a simple translation which uses the Fischer-Ladner closure:

$$G \models U \Rightarrow A \quad \text{iff} \quad \models \{\, [(\cup_a (a \cup a^-))^*]\, B | B \in G \,\} \cup U \Rightarrow A$$

where a are the atomic programs occurring in the subformula closure of G, U and A. It is worth noting that this operation can be performed in polynomial time in the size of the of G, U, and A. As a consequence we restrict ourselves to validity in the rest of the chapter (i.e. $G = U = \emptyset$).

5.4 Prefixed Tableaux Naming Eventualities

We extend the notion of prefixed formulae from modal to CPDL formulae in the obvious way. The difference is that σ is an alternating sequence of integers and *atomic (direct or converse) programs* defined as $\sigma ::= 1 \mid \sigma.a.n \mid \sigma.a^-.n$.

Intuitively, σ denotes the *sequence of two-way transitions* of states and atomic programs **a** that starts from the initial state 1 and reach the state

$\dfrac{\sigma : \pi(\mathbf{a})}{\sigma.\mathbf{a}.m : \pi_0}\ \pi(\mathbf{a})$	with $\sigma.\mathbf{a}.m$ new in the branch
$\dfrac{\sigma : \nu(\mathbf{a})}{\sigma.\mathbf{a}.n : \nu_0}\ \nu(\mathbf{a})^F$	with $\sigma.\mathbf{a}.n$ already present in the branch
$\dfrac{\sigma.\mathbf{a}^{-}.n : \nu(\mathbf{a})}{\sigma : \nu_0}\ \nu(\mathbf{a})^B$	with σ already present in the branch

Figure 5.1: Transitional Rules for CPDL

where φ holds. For instance, the prefix $1.a^{-}.2.b.3.a.5.c^{-}.4$ corresponds to $w_1 \overset{a}{\leftarrow} w_2 \overset{b}{\rightarrow} w_3 \overset{a}{\rightarrow} w_5 \overset{c}{\leftarrow} w_4$. PDL has only one-way transitions.

We use the standard initial subsequence ordering $\sqsubseteq$ – i.e. impose $\sigma \sqsubset \sigma.\mathbf{a}.n$ for every σ, $\mathbf{a}$ and n, and take the reflexive-transitive closure.

5.4.1 Prefixed tableau Rules

The definition of branch and tableau is substantially identical to modal logics (§3.3): a *tableau* $\mathcal{T}$ is a binary tree where nodes are labelled with formulae; a *branch* $\mathcal{B}$ is a path from the root to a leaf; a *segment* $\mathcal{S}$ a path from the root to a node of the tree. If a segment terminates into a branching due to a disjunctive rule (β,X, $LB(\mathbf{a})$) we denote the left-hand extension of $\mathcal{S}$ (the left 'branch') with $\mathcal{S}_l$ and the right-hand with $\mathcal{S}_r$.

Again, a prefix is present in a segment, if there is a prefixed formula with that prefix already in the segment, and it is new if it is not already present. In the sequel we say that the pair $\langle \sigma_0, \mathcal{S}_0 \rangle$ is *shorter* than $\langle \sigma, \mathcal{S} \rangle$ if

- either σ_0 is a proper initial subsequence of σ and $\mathcal{S}_0$ is an initial subsegment of $\mathcal{S}$,
- or $\sigma = \sigma_0$ and $\mathcal{S}_0$ is a proper initial subsegment of $\mathcal{S}$.

We avoid ad hoc rules for sequential combination and nondeterministic choiche and use only the general format of $\alpha, \beta, dneg$ rules which we have defined for modal logics (Fig. 3.1 and Fig. 3.1 in §3.3).

The rules for CPDL formulae starting with an atomic program $\mathbf{a}$ are shown in Fig. 5.1 and are the usual π and ν-rules augmented with special rules for the converse. We need a rule which is the analogous of the (B) rule for symmetric modal logics (Chapter 3). In a nutshell we use *both forward (F) and backward (B) rules* for necessities.

$$\frac{\sigma : \mathtt{f}.\langle P^*\rangle A}{\begin{array}{c}\sigma : \mathtt{f}.A \\ \sigma : \mathtt{f}.\langle P\rangle\langle P^*\rangle A\end{array}}\,[*] \qquad \frac{\sigma : \mathtt{t}.\langle P^*\rangle A}{\begin{array}{c}\sigma : \mathtt{t}.X \\ X \doteq \mathtt{t}.\langle P^*\rangle A\end{array}}\,\langle * \rangle \quad X \text{ new}$$

$$\frac{\sigma : \mathtt{t}.X}{\sigma : \mathtt{t}.A \quad \begin{array}{c}\sigma : \mathtt{f}.A \\ \sigma : \mathtt{t}.\langle P\rangle X\end{array}}\,X$$

Figure 5.2: Rules for $*$-iteration Operator

The rules for iteration combines prefixed tableaux with the ideas of Stirling & Walker [166, 165] for model checking fix points in the modal μ-calculus[4] (Fig. 5.2). In practice when an iterated eventuality $\langle P^*\rangle A$ is found, we introduce a *new* propositional variable $\mathtt{t}.X$ (possibly with indices), set a side condition $X \doteq \langle P^*\rangle A$, and use the X-rule for further reductions. The set $\mathcal{X}$ of propositional variables introduced in this way is distinct from the set $\mathcal{P}rop$ of propositional letters which are used for formulae.

The use of $\sigma : \mathtt{f}.A$ in the right part of the X rule is motivated by the definition of $\langle P^*\rangle A$ as a *least* fixpoint: P-steps are performed while A is false, stopping as soon as A becomes true. Indeed $\langle P^*\rangle A \equiv \langle(\neg A?; P)^*\rangle A \equiv \langle \mathtt{while}\ \neg A\ \mathtt{do}\ P\rangle \top$ is valid in CPDL [60].

The rules for $[P^*]\,A$ are identical, simply swap the signs of the formulae.

As mentioned in section 5.2, these variables are introduced to detect the presence of P loops which do not fulfill $\langle P^*\rangle A$. Using such variables, we can eliminate the $\Rightarrow$ (and its transitive closure) introduced by Pratt [144].

Remark. The combination of converse with iteration is harder than both of them in isolation: although $\nu(\mathbf{a})^B$ is enough for CPDL without iteration and X rules are enough for PDL, their combination is not enough for full CPDL.

Iteration constructs properties with *unbounded delays*, like $\langle a^*\rangle P$, while converse can be used for *late discoveries*, such as $\langle a\rangle\,[a^-]\,\neg P$, which impose a property on the current state *after* the execution of a program. Their combination creates *mines*, like $P \wedge \langle a^*\rangle\,[(a^-)^*]\,\neg P$, which after an unbounded number of iterations, make the initial state inconsistent.

So, we use a restricted *analytic cut* LB ("look behind"), shown in Fig. 5.3.

Since the cut is analytic and its application strongly restricted, its introduction does not destroy the decidability of the calculus (although its unnecessary

[4] A formula $\langle P^*\rangle A$ can be expressed in the modal mu-calculus as $\mu X.A \vee \langle P\rangle X$ where $\mu X.B(X)$ denote the least fix point of the open formula $B(X)$.

$$\frac{\vdots}{\sigma.\mathbf{a}.n : \mathtt{f}.\langle \mathbf{a}^- \rangle B \qquad \sigma.\mathbf{a}.n : \mathtt{t}.\langle \mathbf{a}^- \rangle B} LB(\mathbf{a})$$

provided $\sigma.\mathbf{a}.n$ is already present, and $\mathtt{t}.B$ (or $\mathtt{f}.B$) is a formula of Subfml(φ), where φ is the formula to be proved valid or satisfiable.

Figure 5.3: Look Behind Analytic Cut

application may choke the proof search).

5.4.2 Tableau proofs and ignorable branches

In the sequel, if $\mathcal{S}$ is a segment (possibly a branch) of a tableau we indicate with $\mathcal{S}/\sigma$ the set of prefixed formulae in $\mathcal{S}$ labelled with the prefix σ, i.e.:

$$\mathcal{S}/\sigma = \{\varphi \mid \langle \sigma : \varphi \rangle \in \mathcal{S}\}.$$

Definition 5.2 *A prefix σ is* reduced *in $\mathcal{S}$ if π-rules are the only rules not yet applied to formulae of $\mathcal{S}/\sigma$. It is* fully reduced *if all rules have been applied.*

Two prefixes are a different name for the same state in CPDL (where the past matters) if they

1. have the same properties (dynamic formulae) and
2. are reachable by the same atomic program.

An identical requirement is used for automata [174]. The second requirement is not necessary for PDL.

Definition 5.3 *A prefix σ in the segment $\mathcal{S}$ is a* copy *of a prefix σ_0 in $\mathcal{S}_0$ if*

1. *$\mathcal{S}/\sigma = \mathcal{S}_0/\sigma_0$, and*
2. *both have the form $\sigma'.\mathbf{a}.n$ and $\sigma'_0.\mathbf{a}.m$ for the same atomic program $\mathbf{a}$.*

If distinct X_i, X_j are present in σ_0 and σ we consider them equal if they represent the same iterated eventuality i.e. $X_i \doteq \mathtt{t}.\langle P^* \rangle A \doteq X_j$.

Definition 5.4 *A branch $\mathcal{B}$ is $\pi/$*completed *if*

1. *all prefixes are reduced in $\mathcal{B}$, and*

2. *for every σ which is not fully reduced there is a pair $\langle\sigma_0, \mathcal{S}_0\rangle$ shorter than $\langle\sigma, \mathcal{B}\rangle$ such that σ_0 is fully reduced in the segment $\mathcal{S}_0$ and σ in $\mathcal{B}$ is a copy of σ_0 in $\S_0$.*

If $\sigma : \mathtt{t}.X$ has been considered equal to a $\sigma_0 : \mathtt{t}.X_0$ we say that X *collapses* into X_0. By π/completenes, collapsed X *cannot* generate new $\sigma' : \mathtt{t}.X$ with longer σ'. The intuition for π/completeness is that $\pi_{\mathbf{a}}$-rules are not applied to formulae belonging to copies.

Definition 5.5 *An eventuality $X \doteq \mathtt{t}.\langle P^*\rangle A$ is* fulfilled *in segment $\mathcal{S}$ iff either there is a $\sigma : \mathtt{t}.X$ in S such that $\sigma : \mathtt{t}.A$ is also in $\mathcal{S}$ or X has been collapsed into an X_0 which is fulfilled.*

Definition 5.6 *A branch $\mathcal{B}$ is* contradictory *iff it contains both $\sigma : \mathtt{t}.A$ and $\sigma : \mathtt{f}.A$, for some A and some σ. A π/completed branch $\mathcal{B}$ is* ignorable *iff there is an eventuality X which is not fulfilled.*

In a π/completed branch $\mathcal{B}$, an eventuality X cannot collapse (directly or transitively) into itself and be fulfilled at the same time, since we have either $\sigma : \mathtt{t}.A$ or $\sigma : \mathtt{f}.A$ in the X-rule. Suppose we tried to collapse an eventuality X from $\langle\sigma, \mathcal{B}\rangle$ into a shorter $\langle\sigma_0, \mathcal{S}_0\rangle$ also with $\sigma_0 : \mathtt{t}.X$. Obviously, σ_0 cannot fulfil X otherwise the branch would only have had $\sigma_0 : \mathtt{t}.A$ and a longer $\langle\sigma, \mathcal{B}\rangle$ with X could not have been generated at all, because other occurences of X can only be generated by reducing $\sigma_0 : \mathtt{t}.\langle P\rangle X$ which only occurs together with $\sigma_0 : \mathtt{f}.A$. Thus the only possibility is the right-hand branch $\mathcal{S}_{0r}$ which has another instance of X i.e. $\sigma_0 : \mathtt{t}.\langle P\rangle X$ together with $\sigma_0 : \mathtt{f}.A$. It can generate the longer $\langle\sigma, \mathcal{B}\rangle$ by reducing $\mathtt{t}.\langle P\rangle X$. However if σ must be a copy of σ_0 this means that also $\sigma : \mathtt{f}.A$ must be there and σ does not fulfill X.

In a nutshell we found an unsuccessful loop [144]: we try to fulfill an eventuality $\mathtt{t}.\langle P^*\rangle A$ with X-rules; the left-hand branches with $\sigma : \mathtt{t}.A$ are always discarded; finally we meet a prefix σ_ω with the same formulae of an already seen prefix; so we conclude that we could never fulfill it and give up. We can express this fact with the following sufficient condition:

Proposition 5.7 *A branch $\mathcal{B}$ is ignorable if there is an X which collapses (directly or transitively) into itself provided the corresponding shorter $\langle\sigma_0, \mathcal{S}_0\rangle$ is fully reduced.*

Fully reduced means also that cut has been applied in all possible ways: we must be sure that two prefixes are identical for what regard the past, at least w.r.t. the Fischer-Ladner closure. Cut is only needed here.

Technique 5.1 *Apply $LB(\mathbf{a})$ only if the prefixed formulae $\sigma.\mathbf{a}.n : \mathtt{t}.X$ and $\sigma.\mathbf{a}.n : \mathtt{f}.A$ occurs in the branch and we are trying to collapse X into an X_0 (also itself).*

(a)	$1 : \mathtt{t}.\neg p \wedge \langle a \rangle \left[a^-\right] \langle b^- \rangle q \wedge \left[b^-; b^*\right] p$	Intial formula
(b)	$1 : \mathtt{t}.\neg p$	with α-rules from (a)
(c)	$1 : \mathtt{t}.\langle a \rangle \left[a^-\right] \langle b^- \rangle q$	
(d)	$1 : \mathtt{t}. \left[b^-; b^*\right] p$	
(e)	$1 : \mathtt{f}.p$	with *dneg*-rules from (b)
(f)	$1 : \mathtt{t}. \left[b^-\right] [b^*] p$	*dneg* from (d)
(g)	$1.a.2 : \mathtt{t}. \left[a^-\right] \langle b^- \rangle q$	π_a from (c)
(h)	$1 : \langle b^- \rangle q$	$\nu^B_{a^-}$ from (g)
(i)	$1.b^-.3 : \mathtt{t}.q$	π_{b^-} from (h)
(l)	$1.b^-.3 : \mathtt{t}. [b^*] p$	ν^F_b from (i)
(m)	$1.b^-.3 : \mathtt{t}.p$	$[*]$ from (l)
(n)	$1.b^-.3 : \mathtt{t}. [b] [b^*] p$	
(o)	$1 : \mathtt{t}. [b^*] p$	ν^B_b from (i)
(p)	$1 : \mathtt{t}.p$	$[*]$ from (n)
(q)	$1 : [b] [b^*] p$	
	$\perp$	

Figure 5.4: Tableau Proof for CPDL

Definition 5.8 *A tableau is* closed *if all branches are either contradictory or ignorable. A tableau is* open *if at least one branch is open (π/completed and neither contradictory nor ignorable).*

Definition 5.9 *A* tableau validity proof *for the formula A in the logic* CPDL *is the closed tableau starting with $1 : \mathtt{f}.A$.*

For satisfiable formulae a model can be built from an open branch.

5.5 Examples and Questions

A simple example where converse forces us to move back and forth among states is given in Fig. 5.4. An example of ignorable branch is given in Fig. 5.5. In the second example we do not need to use analytic cut (there is no converse).

Some interesting questions are better clarified by examples.

Question 5.1 *Why different X are introduced each time the same $\langle P^* \rangle A$ is met with a different prefix, if later on we identify them in the loop checking?*

The propositional variables $X \doteq \langle P^* \rangle A$ are an automatic bookkeeping system: if we introduce $\sigma_0 : \mathtt{t}.X$ at a certain stage and, later on, we find a longer σ with $\sigma : \mathtt{t}.X$, then we can infer, *without further checks*, that there are some $\langle P \rangle$-steps from σ_0 to σ, no matter how complicated P is. Hence, if $\sigma : \mathtt{t}.A$ is present in the branch, we can conclude that the initial occurrence of $\langle P^* \rangle A$ and all subsequent occurrences (represented by X) are fulfilled.

$$
\begin{array}{c}
1 : \mathtt{t}.\langle b; a^* \rangle p \wedge [b^*; a^*] \neg p \\
1 : \mathtt{t}.\langle b; a^* \rangle p \\
1 : \mathtt{t}.[b^*; a^*] \neg p \\
1 : \mathtt{t}.\langle b \rangle \langle a^* \rangle p \\
1 : \mathtt{t}.[b^*][a^*] \neg p \\
1 : \mathtt{t}.[a^*] \neg p \\
1 : \mathtt{t}.[b][b^*][a^*] \neg p \\
1.b.2 : \mathtt{t}.\langle a^* \rangle p \\
1.b.2 : \mathtt{t}.[b^*][a^*] \neg p \\
1.b.2 : \mathtt{t}.[a^*] \neg p \\
1.b.2 : \mathtt{t}.[b][b^*][a^*] \neg p \\
1.b.2 : [a^*] \neg P \\
1.b.2 : \mathtt{t}.\neg p \\
1.b.2 : \mathtt{t}.[a][a^*] \neg p \\
1.b.2 : \mathtt{f}.p \\
1.b.2 : \mathtt{t}.X \doteq \mathtt{t}.\langle a^* \rangle p \\
\begin{array}{cc}
\begin{array}{c} 1.b.2 : \mathtt{t}.p \\ \hline \times \end{array} &
\begin{array}{c} 1.b.2 : \mathtt{t}.\neg p \\ 1.b.2 : \mathtt{t}.\langle a \rangle X \\ (+) \end{array}
\end{array}
\end{array}
$$

$$
\begin{array}{c}
(+) \\
1.b.2.a.3 : \mathtt{t}.X \\
1.b.2.a.3 : \mathtt{t}.[a^*] \neg p \\
1.b.2.a.3 : \mathtt{t}.\neg p \\
1.b.2.a.3 : \mathtt{t}.[a][a^*] \neg p \\
1.b.2.a.3 : \mathtt{f}.p \\
\begin{array}{cc}
\begin{array}{c} 1.b.2.a.3 : \mathtt{t}.p \\ \hline \times \end{array} &
\begin{array}{c}
1.b.2.a.3 : \mathtt{t}.\neg p \\
1.b.2.a.3 : \mathtt{t}.\langle a \rangle X \\
1.b.2.a.3 : \mathtt{f}.p \\
1.b.2.a.3.a.4 : \mathtt{t}.X \\
1.b.2.a.3.a.4 : \mathtt{t}.[a^*] \neg p \\
1.b.2.a.3.a.4 : \mathtt{t}.\neg p \\
1.b.2.a.3.a.4 : \mathtt{t}.[a][a^*] \neg p \\
1.b.2.a.3.a.4 : \mathtt{f}.p \\
\begin{array}{cc}
\begin{array}{c} 1.b.2.a.3.a.4 : \mathtt{t}.p \\ \hline \times \end{array} &
\begin{array}{c} 1.b.2.a.3.a.4 : \mathtt{t}.\neg p \\ 1.b.2.a.3.a.4 : \mathtt{t}.\langle a \rangle X \\ \hline 1.b.2.a.3 = 1.b.2.a.3.a.4 \end{array}
\end{array}
\end{array}
\end{array}
\end{array}
$$

Figure 5.5: Another Tableau Proof for CPDL

If we reused the same X for a different prefix $\sigma_1 : \mathtt{t}.\langle P^* \rangle A$ in the branch, then we would not know anymore whether σ follows from σ_0 or from σ_1. We would have to wait for the completion of the (pseudo)model and then modelcheck it, which is exactly what we are trying to avoid.

Yet, X_i are just names for the corresponding $\langle P^* \rangle A$ and we want to fulfill the formula and not a name. If we find out that $X_i \doteq \langle P^* \rangle A \doteq X_j$ this means that they are just different names for the same property: if a state σ_0 fulfils the same formulae of σ plus X_i then it clearly fulfils the X_j occurring in σ and thus we can identify the two formulae and hence the states.

For instance try the following formulae (without $LB(\mathbf{a})$, since there is no converse) starting a tableau for $\mathtt{t}.A_{SAT}$.

$$
\begin{array}{rcl}
A_{SAT} & \doteq & p \wedge [b^*] (\langle b \rangle p \wedge \langle a^* \rangle \neg p) \\
A_{UNSAT} & \doteq & A_{SAT} \wedge [b^*; a^*] \, p
\end{array}
$$

the consequences of the wrong usage of X are shown in Fig. 5.6 for A_{SAT}.

5.5.1 Cut elimination

Question 5.2 *Is cut really necessary?*

Cut is eliminable if formulae do not contain both converse and iteration operators since the soundness proof does not require its use. The difficult formula

$$
\begin{array}{c}
1 : \mathtt{t}.\neg p \wedge [b^*]\langle b\rangle\neg p \wedge [b^*]\langle a^*\rangle p \\[6pt]
1 : \mathtt{t}.\neg p \\
1 : \mathtt{t}.\langle b\rangle\neg p \\
1 : \mathtt{t}.[b][b^*]\langle b\rangle\neg p \\
1 : \mathtt{t}.\langle a^*\rangle p \\
1 : \mathtt{f}.p \\
1 : \mathtt{t}.X \doteq \mathtt{t}.\langle a^*\rangle p \\
\begin{array}{c|c}
\begin{array}{c} \underline{1 : \mathtt{t}.p} \\ \times \end{array} &
\begin{array}{c}
1 : \mathtt{f}.p \\
1 : \mathtt{t}.\langle a\rangle X \\[4pt]
1.a.2 : \mathtt{t}.X \\
1.a.2 : \mathtt{t}.p \\[4pt]
1.b.3 : \mathtt{t}.\neg p \\[4pt]
1.b.3 : \mathtt{f}.p \\[8pt]
1.b.3 : \mathtt{t}.\langle b\rangle\neg p \\[4pt]
1.b.3 : \mathtt{t}.[b][b^*]\langle b\rangle\neg p \\[4pt]
1.b.3 : \mathtt{t}.\langle a^*\rangle p \\[4pt]
1.b.3 : \mathtt{t}.X \\
\begin{array}{c|c}
\begin{array}{c} \underline{1.b.3 : \mathtt{t}.p} \\ \times \end{array} &
\begin{array}{c} 1.b.3 : \mathtt{f}.p \\ 1.b.3 : \mathtt{t}.\langle a\rangle X \\[8pt] \text{ignorab???} \end{array}
\end{array}
\end{array}
\end{array}
\end{array}
$$

$$
\begin{array}{c}
1 : \mathtt{t}.\neg p \wedge [b^*]\langle b\rangle\neg p \wedge [b^*]\langle a^*\rangle p \\[6pt]
1 : \mathtt{f}.p \\
1 : \mathtt{t}.\langle b\rangle\neg p \\
1 : \mathtt{t}.\langle a^*\rangle p \\
1 : \mathtt{t}.[b][b^*]\langle b\rangle\neg P \\
1 : \mathtt{t}.[b][b^*]\langle a^*\rangle p \\[4pt]
1 : \mathtt{t}.X_1 \doteq \mathtt{t}.\langle a^*\rangle p \\
1 : \mathtt{f}.p \\
1 : \mathtt{t}.\langle a\rangle X_1 \\[4pt]
1.a.2 : \mathtt{t}.X_1 \\
1.a.2 : \mathtt{t}.p \\[4pt]
1.b.3 : \mathtt{f}.p \\
1.b.3 : \mathtt{t}.\langle b\rangle\neg p \\
1.b.3 : \mathtt{t}.[b][b^*]\langle b\rangle\neg P \\
1.b.3 : \mathtt{t}.\langle a^*\rangle p \\
1.b.3 : \mathtt{t}.[b][b^*]\langle a^*\rangle p \\
1.b.3 : \mathtt{t}.X_3 \doteq \mathtt{t}.\langle a^*\rangle p \\
1.b.3 : \mathtt{f}.p \\
1.b.3 : \mathtt{t}.\langle a\rangle X_3 \\[4pt]
1.b.3.a.4 : \mathtt{t}.X_3 \\
1.b.3.a.4 : \mathtt{t}.p \\[4pt]
1.b.3.b.5 : \mathtt{f}.p \\
1.b.3.b.5 : \mathtt{t}.\langle b\rangle\neg p \\
1.b.3.b.5 : \mathtt{t}.[b][b^*]\langle b\rangle\neg P \\
1.b.3.b.5 : \mathtt{t}.\langle a^*\rangle p \\
1.b.3.b.5 : \mathtt{t}.[b][b^*]\langle a^*\rangle p \\
1.b.3.b.5 : \mathtt{t}.X_5 \doteq \mathtt{t}.\langle a^*\rangle p \\
1.b.3.b.5 : \mathtt{f}.p \\
1.b.3.b.5 : \mathtt{t}.\langle a\rangle X_5 \\
\vdots
\end{array}
$$

The left tableau is wrongly closed by the ignorability condition because the same X is recycled. The right one never terminates because different X_i for the same eventualities are not identified.

Figure 5.6: Erroneous Tableau 'Proofs'

is $[(P^-)^*]\,A$ which imposes unbounded constraints on the past. Check the following formulae without cut and with a (modified) X-rule without the $\sigma : \mathtt{f}.A$ on the right.

$$
\begin{aligned}
B_{UNSAT} &\doteq p \wedge \langle a^*\rangle \left[(a^-)^*\right] \neg p \\
B_{SAT} &\doteq p \wedge \langle a^*\rangle(\neg p \wedge \bigwedge_{i=1}^{n} \left[a^{-i}\right] \neg p)
\end{aligned}
$$

where we abbreviate $a^-; \ldots ; a^-$ for i times with a^{-i}. The second formula is satisfiable, while the first is not. Without cut and without the $\mathtt{f}.A$-branch in

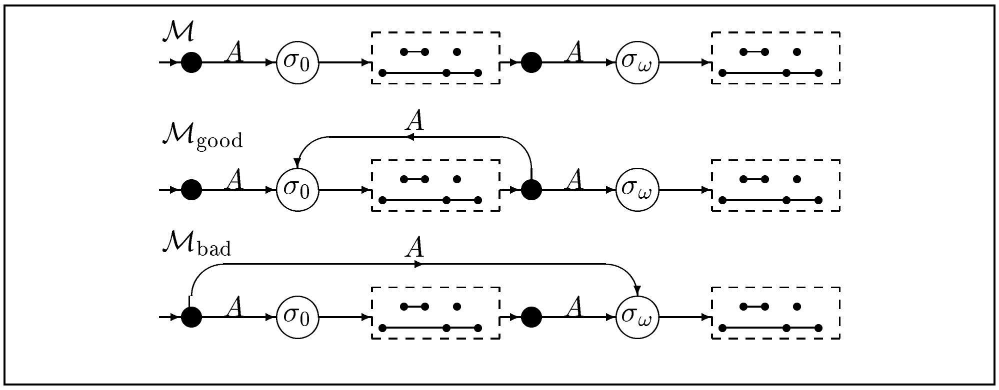


Figure 5.7: Bad and Good Models

the X rule, after n applications of the X rule both formulae will have only contradictory or ignorable branches. After $n+1$ steps, the tableau for B_{SAT} has one non-ignorable branch whereas the one for B_{UNSAT} remains ignorable.

5.6 Soundness and Completeness

The intuitions behind soundness and completeness can be explained by examining the concepts of copied prefixes and ignorable branches from a model theoretic perspective. In particular by comparing tableau rules to visiting steps of a model and prefixes to booking devices (names for states).

Whenever we find two prefixes σ_0 and σ_ω which have the same properties we may conclude that they are essentially identical (model $\mathcal{M}$ in Fig. 5.7). Thus, there is no need to expand the formulae of σ_ω: we have already done it for σ_0 and, if we didn't find a contradiction, we will not find it now. We can avoid the visit of the infinite path from σ_ω by changing the model.

If a branch is open then we introduce a loop back to σ_0, thus dropping the in-principle-infinite path starting from σ_ω (model $\mathcal{M}_{\text{good}}$ in Fig. 5.7). This is the completeness theorem. When a branch is ignorable there is an eventuality $\langle P^* \rangle A$ on σ that, after some $\langle P \rangle$-steps where A is always false, arrives to an identical state σ_ω. So we change the model to $\mathcal{M}_{\text{bad}}$ (Fig. 5.7), and conclude that we cannot fulfill the eventuality in any number of P-steps. This is the soundness theorem. Pratt [145, 144] called these cases successful and unsuccessful loop.

5.6.1 Soundness

The formal soundness proof follows our well established pattern:

1. devise a mapping between prefixes in a tableau and states in a model;
2. prove a safe extension lemma (any tableau rule applied to a satisfiable formula preserves satisfiability with this mapping);
3. prove a safe closure lemma (no satisfiable branch is ignored).

Remark. For modal logics safe closure is immediate (a branch must only be non contradictory) whereas it is the hardest part for PDL/CPDL: we have to verify, with a *finite* computation, that an eventuality will *never* be fulfilled.

The definition of a mapping between prefixes and worlds of a model and the definition of SAT branches are identical to modal logics (§3.6).

The following result can be proved by induction on the number of rules applied to a tableau.

Theorem 5.10 *If $\mathcal{T}$ is a SAT tableau, then the tableau $\mathcal{T}'$ obtained by an application of a tableau rule is also SAT.*

What remains to be done is the most difficult part: proving that ignorable branches can be discarded. We need a preliminary result telling that

- either if an eventuality X is not collapsed then its reduction terminates into a fulfilling $\sigma : \mathtt{t}.A$ i.e. no more X are generated,
- or if we follow the path from an eventuality that is collapsed into itself (directly or indirectly) then we always meet $\langle \sigma : \mathtt{t}.X \rangle$ together with $\langle \sigma : \mathtt{f}.A \rangle$.

To this extent we introduce the following relation $\prec$ between prefixed formulae for each $X \doteq \mathtt{t}.\langle P^* \rangle A$ in a branch $\mathcal{B}$:

- $\langle \sigma_0 : \mathtt{t}.X \rangle \prec \langle \sigma : \mathtt{t}.X \rangle$ iff $\sigma_0 : \mathtt{t}.X$ appears in the segment $\mathcal{S}_0$, the prefixed formula $\sigma : \mathtt{t}.X$ properly appears again[5] in $\mathcal{S}$, and $\langle \sigma_0, \mathcal{S}_0 \rangle$ is shorter than $\langle \sigma, \mathcal{S} \rangle$;
- $\langle \sigma : \mathtt{t}.X \rangle \prec \langle \sigma_0 : \mathtt{t}.X_0 \rangle$ iff X has been collapsed in X_0 and σ_0 is the fully reduced copy of σ.

The transitive (but not reflexive) closure of $\prec$ is $\prec\!\!\prec$. We need to use segments to take tests into accounts. In a nutshell, we are reconstructing (only for the proof and only for iterated eventualities) the $\Rightarrow$ relation of Pratt [144].

Then we can prove the following result, whose intuition is that either an eventuality is fulfilled (and $\prec$ forms a well-ordering with maximum), or after a certain stage all X fall into a final cluster (and $\prec$ is not well-founded).

[5] When $\sigma = \sigma_0$ this means that $\sigma_0 : \mathtt{t}.X$ must have been introduced again in the part of $\mathcal{S}$ properly extending $\mathcal{S}_0$. Otherwise we would always have $\sigma_0 : \mathtt{t}.X \prec \sigma_0 : \mathtt{t}.X$.

Lemma 5.11 *Let $\mathcal{B}$ be a π-completed branch, $\sigma : \mathtt{t}.X$ be a prefixed eventuality, with $X \doteq \langle P^*\rangle A$, and path($\sigma : \mathtt{t}.X$) be the set of $\langle \sigma_i : \mathtt{t}.X_i\rangle$ present in $\mathcal{B}$, such that $\langle \sigma : \mathtt{t}.X\rangle \prec\!\!\prec \langle \sigma_i : \mathtt{t}.X_i\rangle$. Then there is a $\langle \sigma_\omega, X_\omega\rangle$ in path($\sigma : \mathtt{t}.X$) such that:*

- *either $\langle \sigma_\omega : \mathtt{t}.A\rangle$ is in $\mathcal{B}$ and then path($\sigma : \mathtt{t}.X$) is well-ordered by $\prec$ and $\langle \sigma_\omega, X_\omega\rangle$ is the maximum element.*
- *or $\langle \sigma_\omega : \mathtt{t}.X_\omega\rangle \prec\!\!\prec \langle \sigma_\omega : \mathtt{t}.X_\omega\rangle$ and then for all $\langle \sigma_i : \mathtt{t}.X_i\rangle \in$ path($\sigma : \mathtt{t}.X$) the prefixed formula $\sigma_i : \mathtt{f}.A$ is in $\mathcal{B}$ and $\langle \sigma_i : \mathtt{t}.X_i\rangle \prec\!\!\prec \langle \sigma_\omega : \mathtt{t}.X_\omega\rangle$.*

Again the proof is by induction on the applied rules, using the argument for Prop. 5.7 and the fact that collapsed eventualities are not further reduced. Notice that the assignment of a different X for each occurrence of the eventuality $\langle P^*\rangle A$ with a different prefix is necessary for the proof. With this provision the only way to obtain some $\sigma' : \mathtt{t}.X$ is by reducing some previously generated $\sigma : \langle P\rangle X$.

Theorem 5.12 *If $\mathcal{T}$ is a SAT tableau, then one SAT branch is not ignorable.*

The proof is by contradiction: suppose that a SAT branch is ignorable and derive a contradiction. Details are in appendix A.3, we sketch the main points.

Since every SAT branch is ignorable we can prove that an eventuality $\langle P^*\rangle A$ collapsed into itself cannot be fulfilled in less than r P-step (for some $r \geq 1$) on every model. This part of the proof uses in essential way the presence of the $\mathtt{f}.A$ on the right side of the branching rule for $\langle *\rangle$.

Since such branches are SAT then we know, by definition of forcing that the eventuality $\langle P^*\rangle A$ can be satisfied after n P-steps, for some n. Then we change the model, using the technique of Fig.5.7. We show that the two models are almost identical — this is the step where cut is needed for converse — and therefore there is path of length $n - r$ P-steps which fulfils the eventuality. Iterating this reasoning we arrive at a path shorter than r P-steps, contradiction.

The *correctness theorem* follows with a standard argument.

Theorem 5.13 (Soundness) *If A has a validity proof then A is valid for* CPDL.

5.6.2 Completeness

For completeness, we follow the same path of standard modal logic (§3.6): apply a systematic and fair procedure to the tableau and if it does not close, choose an open branch to build a model for the initial formula $1 : \mathtt{f}.A$ i.e. a counter-model for A. The key is the following *model existence theorem.*

Theorem 5.14 *If $\mathcal{B}$ is an open branch then there is $\langle W, \mathcal{I}\rangle$ where it is SAT.*

The proof (details in appendix A.3) has the same overall structure of the corresponding theorem for modal logics (§3.6).

We associate a world $\langle\sigma\rangle$ to every prefix occurring in $\mathcal{B}$ and then prove by induction that $\sigma : \varphi \in \mathcal{B}$ implies $\langle\sigma\rangle \Vdash \varphi$.

There are two important differences: we need to perform a double induction on the structure of programs and on the structure of formulae; the construction of the relation $\mathcal{R}_a$ (here represented by the valuation $(a)^{\mathcal{I}}$) changes because of the presence of the converse operator.

Here we have that $\langle\sigma\rangle \, \mathcal{R}_a \, \langle\sigma.a.n\rangle$ holds[6] as in modal logic K, *and* the relation $\langle\sigma.a^{-}.n\rangle \, \mathcal{R}_a \, \langle\sigma\rangle$ holds as well.

A *completeness theorem* follows with a standard argument:

Theorem 5.15 (Completeness) *If A is valid then A has a validity proof for* CPDL.

5.7 Design of EXPTIME-efficient Algorithms

Tableaux calculi can be used with the following naive algorithm:

- select a formula from the branch and reduce it;
- if the reduction requires branching, then choose one branch and add the other to the stack;
- repeat until the branch is contradictory, ignorable, or open;
- in the first two cases, discard the branch and backtrack.

This algorithm computes each time from scratch without keeping track of discarded branches, i.e. *it does not learn from failures.* This makes sense for logics in PSPACE, but not for PDLs. In fact the naive algorithm works in NEXPTIME, while CPDL is EXPTIME-complete [60, 146].

5.7.1 Traditional and "on-the-fly" EXPTIME-algorithms

A worst case EXPTIME-algorithm can be developed with the techniques of Pratt [144] or Manna & Pnueli [107, Sec.5.1-5.4]: use a data structure where all possible subsets of the formulae that may appear in the tableau are listed. As soon as our expansion procedure introduces a new formula with a certain prefix, we collect the formulae with the same prefix and look in our database: if this set is already marked then we do not expand it further, otherwise we

[6] In equivalent terms one has $(\langle\sigma\rangle, \langle\sigma.a.n\rangle) \in (a)^{\mathcal{I}}$.

mark it. We use pairs, each composed by an arriving program and a set of formulae, as elements of the database.

When this construction terminates, we start a marking algorithm which deletes bad pairs. A key difference with Pratt [144] is that we discard at once all prefixes with an X which makes the branch ignorable. Already for PDL, this is more effective than [144] since we do not compute the transitive closure of $\Rightarrow$ but just look for Xs locally.

Since we mark each set with the arriving program and use the cut-rule then, for each atomic programs $\mathbf{a}$, our database could contain all propositionally consistent subsets of $\left\{ \nu_0, \nu(\mathbf{a}), \overline{\nu(\mathbf{a})} \mid \nu_0 \in \mathrm{Subfml}(\mathtt{f}.A) \right\}$, where A is the formula we want to prove. This gives an upper bound for the database size exponential in $O(|\, Act(A) \,| \times |\, A \,|^2)$, where $Act(A)$ are the direct or converse atomic programs in A, and hence the desired EXPTIME bound.

We can transform this algorithm into an *on-the-fly method* by creating the data structure incrementally as proposed by Bhat et al. [16] and Manna & Pnueli [107, Sec. 5.5] for temporal logics or in this dissertation itself for modal logics (§3.3). Each time we find a a new set of formulae $\mathcal{S}/\sigma$ we add the pair $\langle \mathcal{S}/\sigma, last(\sigma) \rangle$ to the database. Before expanding new formulae we check whether the corresponding pair is not already among the *visited* ones.

Again, our bookkeeping system anticipates the closure of branches due to unfulfilled eventualities. By using X variables, the check for closure is substantially identical to the methods used for temporal logics: the duplication of an eventuality in a visited set is enough to close the branch.

5.7.2 Depth-first incremental EXPTIME-algorithm for PDL

Still the flavor of the algorithm is *breadth first*. Its transformation into a *depth-first* algorithm is similar to the techniques of Bhat et al. [16] for temporal logic. We first present the algorithm for PDL to clarify the intuitions and afterward we extend it to CPDL.

For a segment $\mathcal{S}$ we say that $\mathcal{S}/\sigma.\mathbf{a}.n$ is a *successor* [107] of $\mathcal{S}/\sigma$ if the prefix $\sigma.\mathbf{a}.n$ has been introduced by an application of a $\pi(\mathbf{a})$-rule to a prefixed formula $\sigma : \pi(\mathbf{a})$ in $\mathcal{S}$. Next we can define a 'bad'-set as follows:

Definition 5.16 *Let $\mathcal{T}$ be a tableau and $\mathcal{S}$ a segment and σ a prefix occurring in $\mathcal{T}$, then the set $\mathcal{S}/\sigma$ is an* inconsistent set *($\perp$-set) for $\mathcal{T}$ iff either*

1. $\mathtt{t}.A, \mathtt{f}.A \in \mathcal{S}/\sigma$ *for some formula A (contradiction);*
2. $\mathtt{t}.X, \mathtt{f}.A \in \mathcal{S}/\sigma$ *for some $X \doteq \mathtt{t}.\langle P^* \rangle A$ and $\mathcal{S}/\sigma$ is a copy of a fully reduced σ_0 for a $\langle \sigma_0, \mathcal{S}_0 \rangle$ shorter than $\langle \sigma, \mathcal{S} \rangle$ (unfulfilled eventuality);*
3. $S \subseteq \mathcal{S}/\sigma$ *for some $\perp$-set S;*

4. *$\mathcal{S}$ terminates into a disjunction and both $\mathcal{S}_l/\sigma$ and $\mathcal{S}_r/\sigma$ are $\perp$-sets;*

5. *$\mathcal{S}/\sigma$ has a $\perp$-set successor $\mathcal{S}/\sigma.\mathbf{a}.n$.*

The depth-first algorithm is then immediate:

- select a segment $\mathcal{S}$ and a prefix σ which have some unreduced formulae;
- if $\mathcal{S}/\sigma$ is a $\perp$-set (already seen or by using the two local conditions) then mark all its prefixed formulae as reduced *and* apply Def. 5.16 to generate new $\perp$-sets;
- elseif $\mathcal{S}/\sigma$ is a copy (among the visited nodes) then mark all its prefixed formulae as reduced;
- else select a formula and reduce it; if any of the resulting segment $\mathcal{S}'$ is such that $\mathcal{S}'/\sigma$ is fully reduced then add $\mathcal{S}'/\sigma$ among the visited nodes;
- loop until either the root has become a $\perp$-set (UNSAT) or an open branch can be found (SAT).

Not all selection rules may work properly with this algorithm. It suffices that they are *copy preserving*: if, at the certain stage of the computation, $\mathcal{S}/\sigma$ has been detected as a copy of another $\mathcal{S}_0/\sigma_0$ it should remain so for the rest of the computation. For instance applying first α, ν-like rules (which do not introduce branching or new prefixes) preserves copies for PDL.

The correctness of the algorithm can be proved by a cut and paste argument: whenever we closed a branch due to a bad set we just replace down the previously generated subtree for the bad set, and so on inductively.

For its complexity observe that the number of prefixes of an open branch is bounded by the number of visited sets (less $\perp$-sets) and these are exponentially bounded by the size of the Fischer-Ladner closure. Each closed branch introduces at least one new bad set and those are also bounded by the Fischer-Ladner closure.

For instance suppose we closed $\mathcal{S}_l/\sigma$ and then $\mathcal{S}_r/\sigma$. These two may be equal or both already seen but $\mathcal{S}/\sigma$ is surely added as a new $\perp$-set (otherwise we would not have applied the β-rule but directly closed the branch).

5.7.3 Extension to CPDL

For CPDL things are more involved. The first modification, already mentioned, is to use pairs $\langle \mathcal{S}/\sigma, last(\sigma)\rangle$ for visited nodes. The second step is to introduce the notion of *temporary copy* in a similar fashion of [79, 113] for symmetric and euclidean modal logics. When $\mathcal{S}/\sigma$ is found to be a copy its unreduced formulae are simply frozen and eventually reactivated when a new prefixed

Tableau Proof	Propagation of $\bot$ − sets
$1 : \mathtt{t}.p \wedge \langle a \rangle\, [a^{-};]\, \neg p$	$\{\, \mathtt{t}.p \wedge \langle a \rangle\, [a^{-}]\, \neg p \,\}$
$\Downarrow\ \alpha$	$\uparrow$
$1 : \mathtt{t}.p$	$\uparrow$
$1 : \mathtt{t}.\langle a \rangle\, [a^{-}]\, \neg p$	$\{\, \mathtt{t}.p, \mathtt{t}.\langle a \rangle\, [a^{-}]\, \neg p \,\}$
$\Downarrow\ \pi_a$	$\uparrow$
$1.a.2 : \mathtt{t}.\, [a^{-}]\, \neg p$	$1.a.2$
$\Downarrow\ \nu^{B}_{a^{-}}$	$\downarrow$
$1 : \mathtt{t}.\neg p$	$\{\, \mathtt{t}.p, \mathtt{t}.\neg p \,\}$
$\Downarrow\ dneg$	$\uparrow$
$1 : \mathtt{f}.p$	$\{\, \mathtt{t}.p, \mathtt{f}.p \,\}$
$\bot$	

Figure 5.8: Back-propagation of Bad Sets

formula with prefix σ or σ_0 is introduced (note that a set may be re-activated at most $|\, A \,|$-times). Moreover, in the $\bot$-set condition (2), cut-formulae must have been applied in all possible ways.

Finally we must also change the notion of successor and add the notion of a *predecessor*. For instance the formula $p \wedge \langle a \rangle\, [a^{-}]\, \neg p$ has the tableau described in Fig.5.8: With the PDL-definition the $\bot$-set would only have been $\{\, \mathtt{t}.p, \mathtt{f}.p \,\}$ while also $\{\, \mathtt{t}.p, \mathtt{t}.\langle a \rangle\, [a^{-}]\, \neg p \,\}$ is clearly a $\bot$-set.

In practice, beside rewinding downward from $\mathcal{S}/\sigma.\mathbf{a}.n$ to $\mathcal{S}/\sigma$ through the application of the $\pi(\mathbf{a})$-rule, we can also rewind the deduction upward through the application of the $\nu^B(\mathbf{a})$-rule. So, if $\mathcal{S}/\sigma$ is found to be a $\bot$-set as in the example above then we must rewind upward from $\mathcal{S}/\sigma$ to $\mathcal{S}/\sigma.\mathbf{a}.n$ through the $\nu^B(\mathbf{a}^{-})$-rules until we have rewinded them all and then downward again with the $\pi(\mathbf{a})$-rules. At the end of this process, we arrive at the same prefix σ in a shorter segment $\mathcal{S}_0$ and $\mathcal{S}_0/\sigma$ is the new $\bot$-set.

Part II

Approximate Tableau Calculi

Chapter 6

Approximate Propositional Tableaux

In this chapter we shift from exact deduction methods to the notion of approximate proof for propositional log ic. After an overview of different approaches, we introduce an approximate semantics for propositional logic and present an approximating tableau calculus for which anytime deduction is possible.

We discuss the computational and semantical 'quality guarantees' on the approximation of the anytime deduction mechanism.

Two other methods, by Schaerf & Cadoli [155] and by Dalal [37], for the design of anytime reasoners are considered and we show that they are instances of our general method.

6.1 An Overview of Approximation Methods

The objective of tractability of logical inference has received a major attention in artificial intelligence (at least in the logicist approaches to AI).

At an intuitive level, the obstacle to tractability of reasoning is *unbounded logical omniscience*: given a propositional knowledge base, we can always derive every logical consequence. As pointed out by Fagin, Halpern & Vardi [56], this problem has been solved in two possible ways: weakening the "omniscience" or weakening the "logical" to make inference tractable.

The first approach is based on *tractable but incomplete classical inference procedures.* For instance, we can use unit resolution as advocated by del Val [46, 47] or boolean constraints propagation by Dalal [36, 37]. An alternative approach is restricting inferences to a subset of formulae as proposed by Rantala [147] or Dalal & Etherington [39]. Anyhow, we have a complete procedure for a subset of the language and a sound but incomplete (yet tractable) system for the whole language.

Completeness for the whole language can be recovered by *knowledge compilation (off-line) or analytic cut (on-line).* With knowledge compilation we compute all possible answers with the tractable inference mechanism and store them for future retrieval. Intuitively this is simply a form of lemmaizing, where lemmata are computed once-off and then stored.

The original technique of Selman & Kautz [161] only included Horn lemmas and was therefore complete only for Horn relaxation of the original problems. The new systems proposed by del Val [46, 47] and Dalal [37] use *general clauses as lemmas.* In this way, we can start an incremental knowledge compilation, imposing incremental bounds on the size of the clauses used as lemmata. So we can construct anytime reasoning procedures using the techniques of Russel & Zilberstein [150].

The problem, well known also in the automated reasoning community, is that lemmata tend to grow in exponential number and the result of knowledge compilation (incremental or not) may be an exponential blow-up of the knowledge base [22, 161].

An incremental construction of anytime reasoning procedures can be done by lemmaizing on-line i.e. by using analytic cut as done by Crawford and Kuipers [32] and Dalal [37]. In AI, the use of analytic cut is sometimes called "Socratic completeness". Here the problem is to identify the formulae that must be used with the cut rule.

The dual approach is to weaken the "logical" by *accepting the omniscience in a weaker logic than classical logic.* The (now standard) approach is to use a variant of Belnap's four valued logic [14]. This approach has been initially proposed by Levesque [104], has been refined by a number of researchers such as Delgrande [48] Lakemeyer [99, 100], Fagin et al. [56].

This technique yields (sometimes) tractable, sound & incomplete (or unsound & complete) inference procedures for classical logics. Nevertheless, we still face the problem of computational complexity since the tractability result can only be proved for very limited normal forms. In the general case, the 'approximate' classical logical consequence proposed by Levesque [104] or Fagin et al. [56] is equivalent to the 'exact' logical consequence in the relevance logic of Anderson & Belnap [3] and this is CO-NPTIME-complete [56, 155].

We can construct *incremental approximations* with the techniques proposed by Fagin & Halpern [55] and Schaerf & Cadoli [155]: we give a classical interpretation only to a subset of the propositional variables (the 'interesting' variables) while other propositions are given a three-valued interpretation. By increasing the set of interesting variables we can regain completeness in an incremental fashion.

Schaerf & Cadoli [155] provide sound & incomplete and complete & unsound reasoning while Fagin & Halpern [55] only used sound & incomplete reasoning. Again, the tractability threshold of the overall process is guaranteed only for particular normal forms. The problem is that entailment in their three-valued logic is co-NPtime-complete and therefore we cannot simply ignore the uninteresting parts in the computation (unless they have a special form).

6.2 An Approximate Semantics

The propositional approximation we propose combines in a uniform way a number of these ideas. In particular, we extend the various propositional semantics proposed by Dalal [38], Levesque [105], Schaerf & Cadoli [155] and modify the logic for awareness of Fagin & Halpern [55].

We start with an *approximate partition* (for short ap) of the set of propositional atoms $\mathcal{P}rop$ into three sets: *unknown* propositions, which are denoted by $unk(\mathcal{P}rop)$, *interesting* propositions denoted by $int(\mathcal{P}rop)$, and *incoherent* propositions $inc(\mathcal{P}rop)$.

The intuition is that some atoms describe *interesting* properties and deserve a classical interpretation. They appear in the formulae of an "interesting" (yet approximate) proof or in the truth conditions of a model. Other variables represent *incoherent* properties and so are the formula composed with them. They cannot be used for "interesting proofs". Other atoms corresponds to *unknown* or meaningless properties. They cannot give "interesting models" (for satisfiability or countermodels for validity).

For instance, suppose that we have a propositional knowledge base representing two circuits (see the book of Claesen [27] or section 2.6). The outputs of the subcircuits of level i are equivalent to formulae composed by variables of level $i-1$ and so on. We may want to prove that two circuits are equivalent by using only formulae of level $i-1$ without getting bogged down in the implementation details of level $i-2$, $i-3$, $\ldots 0$. Thus, at a first approximation, we may consider the propositional variables of levels lower than $i-1$ as expressing some potentially incoherent (unverified) properties and disregarding them in the search for an approximate proof.

Dually, suppose that we have coded a planning problem into propositional logic as done by Kautz & Selman [94]. Then, we may start looking for a plan, i.e. a satisfiable assignment for the knowledge base. During the search, we process interesting formulae as usual. Each time we find an unknown formula we conclude this branch of the search with failure: at this level of approximation, interesting models (plans) should not require to give truth values to unknown propositions. When we meet incoherent formulae we simply ignore them; we assume that we can fix them later. Thus we do not consider incoherent formulae in the (approximate) proof that a plan does not exist; only contradictions between interesting formulae matter.

From the viewpoint of inference, the presence of incoherent propositions leads to more models for a KB (these formulae are always approximately satisfied) and thus fewer proofs than classical logics (sound approximation). Unknown formulae yield fewer models for a KB (these formulae cannot be approximately satisfied) and thus more proofs (complete approximations).

The tricky part is to cast these intuitions into a precise deductive framework. The key advantage of the general machinery of section 2.2, is that is sufficient to revise the forcing relation $\Vdash$ for propositional variables, leaving the rest unchanged.

Given a model $\mathcal{I}$, and an approximate partition ap, an approximate forcing relation is defined as in §2.2 with the exception of the conditions on propositional atoms $p \in \mathcal{P}rop$:

$$\begin{array}{lll} \mathcal{I} \Vdash \mathtt{t}.p & \text{if} & p \in int(\mathcal{P}rop) \text{ and } (p)^{\mathcal{I}} = \top \\ & \text{elseif} & p \in inc(\mathcal{P}rop) \\ \mathcal{I} \Vdash \mathtt{f}.p & \text{if} & p \in int(\mathcal{P}rop) \text{ and } (p)^{\mathcal{I}} = \bot \\ & \text{elseif} & p \in inc(\mathcal{P}rop) \end{array}$$

The definition of $\Vdash$ for the other connectives remains unchanged wrt section 2.2. For instance $\mathcal{I} \Vdash \beta$ if and only if $\mathcal{I} \Vdash \beta_1$ or $\mathcal{I} \Vdash \beta_2$. The notions of interesting, unknown and incoherent variables play no role in the definition of connectives (although they play a major role in the quick evaluation of formulae).

From now on, the operator $\Vdash$ represents an approximate forcing. We use the superscript ap only when we need to distinguish it from the classical forcing relation (§2.2). All definitions and notions we have introduced in section 2.2 are unchanged. We only prefix the word "approximate" to each notion. For instance we say that $\mathcal{I}$ is an *approximate model* for A if $\mathcal{I} \Vdash \mathtt{t}.A$ and it is a *approximate counter model* for A if $\mathcal{I} \Vdash \mathtt{f}.A$.

The only thing we revise is the definition of logical consequence (Def. 2.2). We adopt the definition used by Dalal [38] because it is more in the spirit of approximate forcing:

Definition 6.1 *Given an approximate partition, a formula A is an* approximate logical consequence *of assumptions U (in symbols $\models^{ap}_{\mathsf{PC}} U \Rightarrow A$) iff every model for U is not a counter-model for A.*

Remark. This is not a multi-valued logic although it looks like Belnap's 4-valued logic [3, 14] or like the multi-valued assignment for non-S variables in Schaerf & Cadoli [155].

In Belnap's logic and in any multi-valued logic a variable $p \in \mathcal{P}rop$ can assume every value of the domain: $\{\top, \bot, inc, unk\}$. Here, variables in $int(\mathcal{P}rop)$

α_1	α_2	α
unk	X	*unk*
X	*unk*	*unk*
inc	X	X
int	*int*	*int*

dir	*neg*
unk	*unk*
inc	*inc*
int	*int*

β_1	β_2	β
unk	X	X
inc	X	*inc*
X	*inc*	*inc*
int	*int*	*int*

Table 6.1: Superficial Meaning of Formulae

can only assume the values $\top$ or $\bot$; they are classical variables, they can never assume the value *unk*. Variables in $unk(\mathcal{P}rop)$ cannot assume any other value beside *unk*.

This difference is such that our approach is more robust than multi-valued logic: we do not lose the good properties of classical logic at once. For instance Levesque's approach [104, 105] retains only the tautologies of Belnap's relevance logic [14]; the weakening of logical omniscience by Fagin et al. [56] brings the loss of all valid formulae. Here, interesting formulae behave exactly as in classical logic:

Proposition 6.2 *If $vars(A) \cup vars(U) \subseteq int(\mathcal{P}rop)$ then A is an approximate logical consequence of U iff A is a classical logical consequence of U.*

The proof follows from the simple observation that if all propositional variables are interesting then the approximate forcing is a classical forcing.

The interesting aspect of this formalism is the combination of interesting, unknown and incoherent propositions. We shall see in section 6.3 how this fact can be exploited to design an anytime algorithm. In the meanwhile, we would like to evaluate the overall nature of a formula when all three kinds of propositions are present. This can be done in a quick (polynomial time) "superficial" way using a function $\texttt{meaning}(\varphi)$, from signed formulae to the set $\{\, inc, unk, int \,\}$. The base cases depend on the approximate partition at hand and are specified as follows:

$$\texttt{meaning}(\texttt{t}.p) \quad = \quad \texttt{meaning}(\texttt{f}.p) \quad = \quad \begin{cases} int & \text{if } p \in int(\mathcal{P}rop) \\ unk & \text{if } p \in unk(\mathcal{P}rop) \\ inc & \text{if } p \in inc(\mathcal{P}rop) \end{cases}$$

The function is defined inductively in Table 6.1. We use a variable X to represent the table in compact form. For example, in the third row of the α-table we have that if $\texttt{meaning}(\alpha_2) = X$ then $\texttt{meaning}(\alpha) = X$.

The intuition behind this superficial evaluation of a formula is simple. For example, consider negation. If a formula is interesting, then also its negation is interesting. Equally, if a formula represents a unknown concept also its negation is an unknown concept. In the case of disjunction, if a disjunct of a formula is unknown then the overall status of the formula is given by the remaining disjunct. For conjunction, it works in the other way: if a conjunct is unknown, then the conjunction is unknown.

Then unknown and incoherent formulae do not behave as in classical logic and the following properties can be proved by a simple induction:

Proposition 6.3 *If* `meaning`(φ) *is incoherent then every model satisfies* φ.

Proposition 6.4 *If* `meaning`(φ) *is unknown then no model satisfies* φ.

So, reasoning with unknown and incoherent formulae is simple:

Proposition 6.5 *If* $B \in U$ *and* `meaning(t.`B`)` *is unknown then* U *is not approximately consistent.*

In other words, there is no model for unknown properties.

Proposition 6.6 *If* `meaning(f.`A`)` *is incoherent and* U *is approximately consistent then* A *is not an approximate logical consequence of* U.

There is always a counter-model where an incoherent formula A is false. Note that an approximate inconsistency in the knowledge base U overrules the approximate incoherence of the query A.

Still, approximate reasoning does not hinder the semantical equivalent of the *deduction theorem* which can be proved by a simple semantical argument:

Theorem 6.7 *(Approximate deduction theorem) A formula* A *is an approximate logical consequence of* $U \cup \{B\}$ *if and only if* $B \supset A$ *is an approximate logical consequence of* U. *In simbols:*

$$\models^{ap}_{\mathsf{PC}} U \cup \{B\} \Rightarrow A \quad iff \quad \models^{ap}_{\mathsf{PC}} U \Rightarrow B \supset A$$

This is definitely a good property since this means that in our logic there is *not* a difference between validity and logical consequence. So, the complexity of the deductive task cannot be changed by a simple reformulation of the problem. In contrast, with the approaches of Fagin et al. [56] or Levesque and Lakemeyer [104, 99, 100] one can guarantee a tractability threshold only for formulae in a restricted normal form. In particular in [56] formalizing a problem as $\models \emptyset \Rightarrow B \supset A$ requires constant time (the answer is always no) whereas the formalization as $\models \{B\} \Rightarrow A$ jumps to co-NPTIME-complete. Similar considerations hold for the approach of Cadoli & Schaerf [155].

Some examples may be useful, in particular the behavior of our formalism with respect to logical omniscience. At first modus ponens may fail. In particular we cannot trigger modus ponens on a chain of implications involving incoherent formulae:

$$\begin{array}{rclcrcl} U_1 & = & \{p, p \supset q, q \supset r, r \supset s\} & \quad & inc(\mathcal{P}rop) & = & \{r\} \\ int(\mathcal{P}rop) & = & \{p, q, s\} & & unk(\mathcal{P}rop) & = & \{\ \} \end{array} \tag{6.1}$$

We can conclude that $\models^{ap}_{\mathsf{PC}} U_1 \Rightarrow q$ but $\not\models^{ap}_{\mathsf{PC}} U_1 \Rightarrow s$: we cannot use r, an incoherent property for modus ponens. Hence, with incoherent concepts our reasoning may be incomplete and therefore unbounded logical omniscience only holds for completely interesting formulae.

Case analysis may also fail. Consider the following KB:

$$\begin{array}{rclcrcl} U_2 & = & \{p \supset q, \neg p \supset q, r \supset s, \neg r \supset s\} & \quad & inc(\mathcal{P}rop) & = & \{r\} \\ int(\mathcal{P}rop) & = & \{p, q, s\} & & unk(\mathcal{P}rop) & = & \{\ \} \end{array}$$

The formula q is an approximate logical consequence of U_2 but s is not.

With unknown concepts we may draw *unsound conclusions*. Consider the following example:

$$\begin{array}{rclcrcl} U_3 & = & \{p, p \supset q, \neg q \vee \neg r \vee s\} & \quad & inc(\mathcal{P}rop) & = & \{\ \} \\ int(\mathcal{P}rop) & = & \{p, q, s\} & & unk(\mathcal{P}rop) & = & \{r\} \end{array}$$

Then q is an approximate consequence of U_3. Also s is an approximate consequence of U_3. This happens because, in search of countermodels, we skip any model requiring to assign a truth value to r, an unknown proposition.

Unsound conclusions are not a 'big' problem is they are not too many, and our approach constrain their number. We want to stress the robustness of this approach in comparison with others forms of complete approximation which use abstraction (see e.g. Giunchiglia & Walsh [73]): in the example above $r \wedge \neg r$ is an approximate consequence of U_3 but the formula $s \wedge \neg s$ is *not* an approximate consequence of U_3. On the contrary, the use of abstraction inevitably leads to completely inconsistent abstract spaces: from $r \wedge \neg r$ we derive that $A \wedge \neg A$ is a logical consequence for every A.

Therefore we are able to *tolerate inconsistency* (limited to unknown and incoherent variables) in our framework.

In summary, when some propositions are considered unknown we obtain incomplete but possibly unsound approximations while considering propositions incoherent leads to sound but incomplete approximations. We can have both forms of reasoning.

Remark. An important observation is that our approximate calculus is not closed under all possible substitutions.

As a consequence, we cannot always consider $\top$ and $\perp$ as abbreviations for arbitrary tautologies and contradictions. Suppose that A is very complex tautology whose atomic components are considered (at a certain stage of approximation) as incoherent propositions, i.e. $\mathrm{vars}(A) \subseteq inc(\mathcal{P}rop)$. Then we have that $\not\models^{ap}_{\mathsf{PC}} \neg A \Rightarrow \perp$. This is also justified from an intuitive point of view: we can immediately recognize $\perp$ as contradictory, but why should we bother to spend our (scarce) computational resources to recognize that $\neg A$ is contradictory? After all A is an incoherent sentence.

6.3 Anytime Tableaux

The general machinery we have introduced will make it possible to devise extremely general anytime algorithms without substantial changes in the overall deduction procedures.

6.3.1 Approximate Rules

Once we have fixed an approximate partition of the set $\mathcal{P}rop$, we can change the definition of reduced formulae and closed branches:

Definition 6.8 *Let ap be an approximate partition; the formula φ it is* approximately reduced *in the branch $\mathcal{B}$ iff*

- *either* $\mathtt{meaning}(\varphi) = int$ *and it is reduced in $\mathcal{B}$ according the classical Definition 2.4,*
- *or* $\mathtt{meaning}(\varphi) = inc$.

Definition 6.9 *Let ap be an approximate partition; a branch $\mathcal{B}$ is* approximately contradictory *iff*

- *either $\mathcal{B}$ is contradictory according the classical Definition 2.5 for some interesting formulae ψ and $\overline{\psi}$,*
- *or* $\mathtt{meaning}(\varphi) = unk$ *for some $\varphi \in \mathcal{B}$.*

Tableaux rules, definition of closed and open tableau etc. are identical to those of section 2.3; we simply prefix each notion with the word "approximate". We say that a tableau proof is a *classical proof* if the classical conditions for closure have been used (i.e. without appeal to unknown formulae). In a similar way we say that an open branch is a *classical counter-model* if it is reduced according the standard notion (i.e. without incoherent formulae).

Recall that we always apply techniques T.2.1 and T.2.2 (see page 28 and 28) which prohibit the reduction of already reduced formulae or the reduction of contradictory branches. After the change in the definition of contradictory branches and reduced formulae, the outcome is that we only apply tableau rules to interesting formulae:

Technique 6.1 *Reduce a formula* φ *only if* $\texttt{meaning}(\varphi) = int$.

This restriction has a twofold use:

- for the β rule, avoids a useless branching;
- for the *simp* rule preserve the soundness and the completeness of the rule with respect to the approximate semantics.

Since we are not interested in approximate proofs for their own sake it makes sense to apply simplification also to incoherent and unknown formulae. We know that it is sound and complete for classical logic (§2.3) and that it can be performed in polynomial time. Such applications of simplification to an incoherent formula may lead directly to a classical proof. The trade-off is that simplification is no longer sound for the approximate semantics.

Consider example 6.1. The formulae in the proof for $U_2 \Rightarrow s$ have the following meaning:

$$\begin{aligned} \texttt{meaning}(\texttt{t}.p) = \texttt{meaning}(\texttt{t}.p \supset q) = \texttt{meaning}(\texttt{f}.s) &= int \\ \texttt{meaning}(\texttt{t}.q \supset r) = \texttt{meaning}(\texttt{t}.r \supset s) &= inc \end{aligned}$$

In Fig. 6.1 we show two tableaux: in the left one simplification is not applied to incoherent formuale whereas in the right it is applied to every formula. The left tableau yields an approximate counter-model for s and is therefore the correct 'sound' answer from the viewpoint of the approximate semantics, because s is not an approximate consequence of U_2 with the chosen partition. However, the right tableau is sound with respect to the classical semantics which is the one we really care about.

So, the application of simplification to incoherent or unknown formulae is a matter of philosophical taste: don't use it, if you care for the approximate semantics; use it, if you care only for the classical semantics and look at the approximation only as an intermediate result when you have not enough time.

Then, we use the following techniques (2.6):

Technique 6.2 *Apply rules* α *and dneg before other rules.*

Technique 6.3 *Apply the simplification rule to (interesting) propositional atoms before any other rule.*

Remark. The presence of simplification, its precedence over other rules and its application (at least) to interesting propositional atoms will be *essential* to guarantee the performance of the anytime algorithm.

	$\mathtt{f}.s$
	Loc
$\mathtt{f}.s$	$\mathtt{t}.p$
Loc	$\mathtt{t}.p \supset q$
$\mathtt{t}.p$	$\mathtt{t}.q \supset r$
$\mathtt{t}.p \supset q$	$\mathtt{t}.r \supset s$
$\mathtt{t}.q \supset r$	$simp_{p,p \supset q}$
$\mathtt{t}.r \supset s$	$\mathtt{t}.q$
$simp_{p,p \supset q}$	$simp_{q,q \supset r}$
$\mathtt{t}.q$	$\mathtt{t}.r$
open	$simp_{r,r \supset s}$
	$\mathtt{t}.s$
	closed

In the left derivation we start the tableau with $\mathtt{f}.s$ and apply rule Loc. Beside $\mathtt{t}.p$, $\mathtt{t}.p \supset q$, and $\mathtt{f}.s$, all formulae are incoherent. So we are only left with the simplification of $(\mathtt{t}.p \supset q)[\mathtt{t}.p]$ which yields $\mathtt{t}.q$. Now all formulae are approximately reduced and we are done.

The right derivation starts as the left derivation except that we simplify with every atomic proposition we find in the proof, no matter whether interesting or not. Notice that the proof still take polynomial time because simplification does not introduce branching.

Figure 6.1: Simplification of non-interesting formulae

6.3.2 An Anytime Proof Procedure

We have now the formal machinery to construct an approximating sequence of proofs of increasing precision as in Fig. 1.1. The algorithm we use for the proof search is shown in Fig. 6.2. If we time out after $\mathcal{T}_k$, we have available either an approximate proof (all branches of $\mathcal{T}_k$ are approximately closed) or an approximate counter model (an open branch of $\mathcal{T}_k$). With sufficient resources we can always arrive at a stage that $int_k(\mathcal{P}rop) = \mathcal{P}rop$ to find a classical proof (or counter-model).

In the sequel we use the symbol $\vdash^{ap_k}_{\mathsf{PC}} U \Rightarrow A$ when this algorithm find a tableau proof for the k-th approximate partition and $\not\vdash^{ap_k}_{\mathsf{PC}} U \Rightarrow A$ when an approximate counter-model is returned.

Let U be a set of assumptions and A a formula then

1. **choose** an approximate partition so that the set of interesting variables $I_0 = int(\mathcal{P}rop)$ is small (e.g. logarithmic in the size of U and A);

2. **compute** an approximate tableau $\mathcal{T}_k$;

3. **if** the approximate tableau $\mathcal{T}_k$ is a classical proof or a classical counter-model **then** stop;

4. **else if** not timeout

   - **find** the part of $\mathcal{T}_k$ where non classical reduction or closure rules are used;
   - **increase** the precision so that at least one approximately reduced or contradictory formula change status by enlarging the set of interesting variables from I_k to I_{k+1};
   - **return to** (2) for a new $\mathcal{T}_{k+1}$;

Figure 6.2: Anytime Algorithm for Propositional Deduction

6.3.3 Finding a "good" partition

Among the many sequences of approximate partitions that we can generate there are two simple sequence which corresponds to "only sound" and "only complete" anytime algorithms.

Only sound approximation: let $unk_k(\mathcal{P}rop) = \emptyset$ for all k. This means that we start with few interesting variables and many incoherent ones which are gradually moved into the set of interesting propositions.

Only complete approximation: let $inc_k(\mathcal{P}rop) = \emptyset$ for all k. This means that we start with few interesting variables and many unknown ones which are gradually considered interesting propositions.

Finding a good initial partition or deciding how to improve the current partition is difficult. We devise one particular choice in the section about Dalal anytime family of reasoner (§6.6).

Since we have an approximate proof, we can also use the "approximate steps" of the proof as a guide for passing from the partition of stage k to the partition of stage $k+1$. In particular, if a branch in $\mathcal{T}_k$ has been closed because it contains a formula φ such that $\texttt{meaning}(\varphi) = unk$ then we can revise some variables of φ and consider them interesting. For instance, if φ is of type α

and only α_1 is of type unknown, then we can select one or more variables until α_1 change its status.

6.4 Quality and Complexity of Approximations

Now we must prove the following properties:

- for every approximate partition, the approximate tableau method is sound and complete wrt the approximate semantics;
- the 'semantical' quality of approximation increases when we increase the set of interesting propositions (See again Fig. 1.1);
- the cost of obtaining a better approximation increases according the requirements of anytime algorithms.

6.4.1 Approximate soundness and completeness

Soundness and completeness results follow from the soundness and completeness result for standard propositional logic with simplification.

Definition 6.10 *A tableau branch $\mathcal{B}$ is SAT for the model $\mathcal{I}$ with local assumptions U iff for every signed formula φ in $\mathcal{B}$, one has $\mathcal{I} \Vdash^{ap} \varphi$. A tableau is SAT if at least a branch is SAT.*

The key step for the soundness proof is a safe extension lemma:

Lemma 6.11 *Let $\mathcal{T}$ be an SAT tableau, the tableau $\mathcal{T}'$ obtained by an application of an approximate rule for is also SAT.*

The proof is a standard proof by induction on the applied rules as in the classical case.

We can close a tableau without having reached a contradiction (i.e. when we found an unknown formula), so we also need a *safe closure lemma* as we have done for dynamic logic:

Lemma 6.12 *If $\mathcal{T}$ is a SAT tableau, then one SAT branch is not approximately contradictory.*

The proof is by contradiction (see appendix B.1 for details): we assume that a SAT branch is approximately contradictory and we derive a contradiction. The tricky part of the proof is when both φ and $\overline{\varphi}$ are present in $\mathcal{B}$ and $\texttt{meaning}(\varphi) = \texttt{meaning}(\overline{\varphi}) = int$.

The standard proof would proceed by induction showing that is they are contradictory then we can unwind the contradiction down to atomic level.

The problem is that here "contradictory" incoherent formulae are satisfied by every model. So we must use the conditions in Table 6.1 and reason by cases to rule out such possibility.

Then we can prove that the system is sound with a standard argument:

Theorem 6.13 (Approximate Soundness) *Given an approximate partition, if the formula A has a approximate tableau proof with assumptions U then it is an approximate logical consequence of U.*

For the completeness the hard part is the model existence theorem:

Theorem 6.14 *If $\mathcal{B}$ is an approximately open branch then there is an approximate model $\mathcal{I}$ where it is SAT.*

In comparison with the standard construction of a model (§2.3) we must use a little trick: we assign the truth value $\top$ to a proposition p only if p is interesting. So we have that $(p)^{\mathcal{I}} = \top$ only if $\mathtt{t}.p \in \mathcal{B}$ and $p \in int(\mathcal{P}rop)$.

In this way we avoid the potential conflict caused by incoherent propositions. Recall that a branch can be approximately open even if $\varphi, \overline{\varphi} \in \mathcal{B}$ provided $\mathtt{meaning}(\varphi) = inc$. The key observation is that we do not have to worry about them. We know by §6.2 that $\mathcal{I}$ will satisfy them anyway.

We need some careful reasoning on the α and β formulae when they are not incoherent, because one of their subformulae may be unknown and therefore not satisfiable. So, all possible cases in Table 6.1 must be considered to rule out this possibility.

Now we can prove that the system is complete with a standard argument: Suppose A is a logical consequence of U but there is no closed tableau for A. We start the tableau with $\mathtt{f}.A$ and then apply any systematic procedure for reducing formulae. We end up with only reduced branches. Since at least a branch is not closed we apply the model existence theorem to the branch and we derive a counter-model for A obtaining a contradiction.

Theorem 6.15 (Approximate Completeness) *Given an approximate partition, if the formula A is an approximate logical consequence of assumptions U then it has a approximate tableau proof with U.*

A corollary of the soundness and completeness theorems is that each step of the anytime proof procedure is sound and complete for the current approximation of the anytime algorithm:

Corollary 6.16 *For every approximate partition $ap_0, \ldots ap_k$ chosen by the anytime algorithm, $\vdash^{ap_k}_{\mathsf{PC}} U \Rightarrow A$ if and only if A is an approximate logical consequence of assumptions U for the partition ap_k.*

6.4.2 The semantical quality of approximation

We have just proved that every step of the algorithm is sound and complete wrt the particular partition chosen at each stage. The next result shows that, given enough time and motivation we are guaranteed to find *the* correct (classical) answer:

Theorem 6.17 (Convergence Guarantee) *If $k \geq \mid vars(A) \cup vars(U) \mid$ then $\vdash_{\mathsf{PC}}^{ap_k} U \Rightarrow A$ if and only if A is classical logical consequence of U.*

It is a consequence of Prop.6.2 together with the observation that the set of interesting variables $int_k(\mathcal{P}rop)$ increases after each iteration.

The algorithm may also terminate with a classical proof or classical counter-model *before* reaching the stage where $\mathrm{vars}(A) \cup \mathrm{vars}(U) \subseteq int(\mathcal{P}rop)$. Then we need no further iteration:

Proposition 6.18 (Soundness Guarantee) *Let ap_k be an approximate partition such that $\vdash_{\mathsf{PC}}^{ap_k} U \Rightarrow A$ and no formula φ with unknown type has been used to close a branch of $\mathcal{T}_k$. Then A is a classical logical consequence of U.*

Proposition 6.19 (Completeness Guarantee) *Let ap_k be an approximate partition such that $\not\vdash_{\mathsf{PC}}^{ap_k} U \Rightarrow A$ and classically unreduced no formula φ with incoherent type is present in an open branch of $\mathcal{T}_k$. Then A is not a classical logical consequence of U.*

Both results are an immediate consequence of the classical soundness and completeness theorem for propositional tableau with simplification (§2.3).

We would also like to know the *semantical direction* of approximation: after each stage of iteration are going towards a "more complete" reasoning or a "more sound" reasoning?

Recall that we use unknown proposition for unsound reasoning and incoherent proposition for incomplete reasoning. Therefore, from an intuitive point of view, we can increase the soundness of the procedure by decreasing the number of unknown proposition. An unsound approximation deduces more consequence than it should and therefore increasing soundness means deducing less consequences.

This can be characterized precisely (proofs in appendix B.1):

Theorem 6.20 (Soundness Increase) *Let ap_k and ap_{k+1} be two approximate partition such that $unk_k(\mathcal{P}rop) \supseteq unk_{k+1}(\mathcal{P}rop)$ and $inc_k(\mathcal{P}rop) = inc_{k+1}(\mathcal{P}rop)$. Then $\not\vdash_{\mathsf{PC}}^{ap_k} U \Rightarrow A$ implies $\not\vdash_{\mathsf{PC}}^{ap_{k+1}} U \Rightarrow A$.*

Theorem 6.21 (Completeness Increase) *Let ap_k and ap_{k+1} be two approximate partition such that $inc_k(\mathcal{P}rop) \supseteq inc_{k+1}(\mathcal{P}rop)$ and $unk_k(\mathcal{P}rop) = unk_{k+1}(\mathcal{P}rop)$. Then $\vdash_{\mathsf{PC}}^{ap_k} U \Rightarrow A$ implies $\vdash_{\mathsf{PC}}^{ap_{k+1}} U \Rightarrow A$.*

It is immediate to prove that the two sequences of only sound and only complete approximations in section 6.3 can be used to provide anytime algorithms with the required guarantees.

6.4.3 Computational Complexity

The last step is proving that our proof search algorithm can offer the anytime complexity guarantee requested by Russel & Zilberstein [150]. In particular, this implies that the time requested to find out $\mathcal{T}_k$ is bounded by a polynomial in the size of U and A and an exponential in k. See again page 12 for a definition of anytime algorithms.

At first we need some preliminary properties whose proof is immediate.

Proposition 6.22 *The verification of* `meaning`(φ) *requires polynomial time in the size of* φ *and* $vars(A) \cup vars(U)$.

Without further restrictions, we cannot bound a priori the time needed to pass from ap_k to ap_{k+1}, since it is always possible to design particularly inefficient algorithms. For instance, one could call a theorem prover to verify whether $\models_{\mathsf{PC}} U \Rightarrow A$ and then use for ap_k exactly the propositions used by the prover in the proof.

Technique 6.4 *For the generation of* ap_{k+1}*, apply a polynomial time algorithm to the non interesting formulae of* $\mathcal{T}_k$.

The algorithm may take constant time by starting from a fixed enumeration of propositions and adding them in sequence to ap_0, ap_1 etc.

From these preliminary results, and the observation that simplification can be done in polynomial time (§2.3), we derive the following property:

Lemma 6.23 *For every sequence of approximate partitions* $ap_0, \ldots ap_k$ *and every algorithm which employs techniques T.6.1-6.4, both the time necessary to construct* $\mathcal{T}_k$ *and the size of* $\mathcal{T}_k$ *are proportional to the product of a polynomial in the size of* U *and* A *for the number of branches of* $\mathcal{T}_k$.

So, the only thing we need for an anytime guarantee is a bound on the number of branches of $\mathcal{T}_k$. A careful observation of the completeness proof reveals the following property:

Proposition 6.24 *Let* ap_k *be an approximate partition. If* $\mathcal{I}$ *is an approximate counter-model for* A *with assumptions* U *then every* $\mathcal{I}_0$ *which agrees with* $\mathcal{I}$ *on interesting propositions is also a counter-model for* A.

Thus, during the proof search, we can search for a partial assignment of the interesting variables, disregarding the others. It can be done using the cut rule (§2.3) on literals in $int(\mathcal{P}rop)$, generating in this way an analogue of the Davis-Putnam procedure (§2.5).

Consider the following additional technique:

Technique 6.5 *Let $p \in int_k(\mathcal{P}rop)$. If p also occurs in a branch $\mathcal{B}$ of the tableau then apply the cut rule to $\mathtt{t}.p$ and $\mathtt{f}.p$ in $\mathcal{B}$.*

Then we can prove the following result (see appendix B.1)

Theorem 6.25 (Anytime Guarantee using the cut rule) *Let ap_k be an approximate partition. The algorithm which employs techniques T.6.1-6.4 and T.6.5 always returns a tableau $\mathcal{T}_k$ with at most $2^{|int_k(\mathcal{P}rop)|}$ branches.*

The key observations is that the cut rules introduce a signed atomic proposition and that the systematic application of simplification *before* other rules eliminates the atom from the branch (§2.3). So, a simple induction on the number of propositions does the job.

We can give an anytime guarantee (although slightly worse[1]) in the absence of the cut rule, using only the β-rule. To this extent we need to introduce the concept of *branching depth* $\mathrm{bdepth}(\varphi)$ of a formula. At first we set $\mathrm{bdepth}(\varphi) = 0$ whenever $\mathtt{meaning}(\varphi) \neq int$ and then for the case where φ is interesting we proceed as follows:

$$
\begin{aligned}
\mathrm{bdepth}(\mathtt{t}.p) &= 0 \\
\mathrm{bdepth}(\mathtt{f}.p) &= 0 \\
\mathrm{bdepth}(\alpha) &= \max\{\,\mathrm{bdepth}(\alpha_1), \mathrm{bdepth}(\alpha_2)\,\} \\
\mathrm{bdepth}(\beta) &= 1 + \max\{\,\mathrm{bdepth}(\beta_1), \mathrm{bdepth}(\beta_2)\,\}
\end{aligned}
$$

Notice that if α_1 is incoherent then obviously $\mathrm{bdepth}(\alpha_1) = 0$ If $\mathcal{B}$ is a set of signed formulae we have $\mathrm{bdepth}(\mathcal{B}) = \max\{\,\mathrm{bdepth}(\varphi) \mid \varphi \in \mathcal{B}\,\}$.

Remark. The branching depth of a set of signed formulae is bounded by the number of occurrences of interesting propositions.

Then we use the following technique:

Technique 6.6 *Always select the formulae with the lowest branching depth in the branch.*

Then we need the following lemma:

[1] It is known that using cut makes a computational difference [171].

Lemma 6.26 *Let ap be an approximate partition, and let $\mathcal{T}$ be a tableau for a signed formula φ which does not contain the constants $\top$ and $\bot$. If the tableau has been generated using technique T.6.1 and bdepth(φ) ≥ 1 then in every reduced branch $\mathcal{B}$*

- *either there is an interesting signed atom* $\mathtt{t}.p$ *or* $\mathtt{f}.p$ *in* $\mathcal{B}$,
- *or there is an unknown formula in* $\mathcal{B}$.

Intuitively the only possibility to violate this lemma is a branching which introduced incoherent formulae. For instance suppose that φ was a β formula. If any of the β_i is incoherent then also the generating β would be incoherent (see Table 6.1) and therefore it would have been bdepth(φ) $= 0$.

Theorem 6.27 (Anytime Guarantee using the β rule) *Let ap_k be an approximate partition and let bd_k be the branching depth of interesting propositions in $\{\mathtt{t}.B | B \in U\} \cup \{\mathtt{f}.A\}$. The algorithm which employs techniques T.6.1-6.4 and T.6.6 always return a tableau $\mathcal{T}_k$ with at most $2^{bd_k \times |int_k(\mathcal{P}rop)|}$ branches.*

If we see the tableau as a binary tree (only counting β-reductions) the theorem requires that the height of the tree is less or equal to $bd_k \times |int_k(\mathcal{P}rop)|$.

The difference from the previous theorem is that we may have to break connectives at least bd_k times before finding an interesting variable to whom we can apply the induction principle.

This result is not entirely satisfactory in the general case, since bd_k depends on the *number of occurrences* of interesting variables and not on the number of interesting variables tout court. However, for clausal normal form it is satisfactory since it is the proportional to the length of the longest interesting clause. If the length of clauses is bounded, this is a constant.

6.5 Schaerf and Cadoli Approximate Entailment

The approximate entailment defined by Schaerf & Cadoli [155] builds upon the previous work of Levesque [104, 105]. The main idea is close to the one we have just used: identify a set S of interesting propositions and then define a form of approximate reasoning where only interesting propositions do matter.

Two entailments are defined: $U \models_S^3 A$ and $U \models_S^1 A$. The first corresponds to a sound approximation of logical consequence whereas the second is a complete approximations. S-1 entailment corresponds to our superficial entailment when all variables not in S are unknown. The S-3 entailment is based on Levesque's 3-valued logic [104] a variant of Belnap's logic [3, 14].

The semantics of S-1 and S-3 entailments in [155] is only defined for formulae in negation normal form. We generalize it to our formalism by revising the definition of $\mathcal{I}$ and using *signed interpretations*, i.e. valuations V on signed literals rather than literals.

Definition 6.28 *An S-1 signed interpretation is a function V from signed propositions to the set $\{\top, \bot\}$ such that $V(\mathtt{t}.p) \neq V(\mathtt{f}.p)$ if $p \in S$ and $V(\mathtt{t}.q) = V(\mathtt{f}.q) = \bot$ if $q \in \mathcal{P}rop \setminus S$.*

Definition 6.29 *An S-3 signed interpretation is a function V from signed propositions to the set $\{\top, \bot\}$ such that $V(\mathtt{t}.p) \neq V(\mathtt{f}.p)$ if $p \in S$ and $V(\mathtt{t}.q) \neq \bot$ and $V(\mathtt{f}.q) \neq \bot$ if $q \in \mathcal{P}rop \setminus S$.*

We revise the notion of forcing of signed propositions (§2.2 and §6.2):

$$\begin{array}{lll} V \Vdash \mathtt{t}.p & \text{iff} & V(\mathtt{t}.p) = \top \\ V \Vdash \mathtt{f}.p & \text{iff} & V(\mathtt{f}.p) = \bot \end{array}$$

The forcing for β, α and double negation formulae is unchanged. We only need to change the definition of logical consequence:

Definition 6.30 *Given a set of variables S, a formula A is an S-3 (S-1) logical consequence of assumptions U iff every S-3 (S-1) model for U is a model for A.*

Notice the difference with our Def. 6.1.

Remark. Notice that S-3 signed interpretations are 3-valued interpretation on formulae in $\mathcal{P}rop \setminus S$. Therefore, with this definition S-3 logical consequence when $S = \emptyset$ is close to entailment in relevance logic.

The tractability results of Schaerf & Cadoli [155] are restricted to clausal normal form for both U and A. In the general case, even when $S = \emptyset$, the notion of S-3 entailment coincides with Levesque notion of entailment [104] and this has been proved to be CO-NPTIME-complete by Fagin et al. [56]. We do not gain from a three-valued semantics in the general case.

For the comparison to make sense, we restrict ourselves to clausal normal form. So U is a *set of clauses* and A is a *clause*. To show the equivalence with our system we need some preliminary results proved by Schaerf & Cadoli [155, page 265]. For sake of readability we reformulate them using our notation:

Lemma 6.31 *Let S' be the set $S \cup vars(A)$. Then A is an S-3 consequence of U iff A is an S'-3 consequence of U.*

This lemma is a reformulation of Lemma.5.2 in [155].

Lemma 6.32 *Let $A = B \vee C$ where $vars(B) \subseteq S$ and $vars(C) \cap S = \emptyset$. Then A is an S-1 consequence of U iff B is an S-1 consequence of U.*

This lemma follows immediately from Theorem.5.3 in [155].

The intuition behind both lemmata is that to prove S-1 or S-3 logical consequence we can restrict ourselves to the case when all variables in the query A are interesting.

The equivalence with our system is described by the following properties:

Theorem 6.33 *Let $U \cup \{ A \}$ be a set of formulae in clausal normal form and let $S \subseteq \mathcal{P}rop$ be a set of propositional atoms such that $vars(A) \subseteq S$. Choose an approximate partition such that $unk(\mathcal{P}rop) = \emptyset$ and $int(\mathcal{P}rop) = S$. Then A is an S-3 consequence of U iff A is an approximate logical consequence of U.*

Theorem 6.34 *Let $U \cup \{ A \}$ be a set of formulae in clausal normal form and let $S \subseteq \mathcal{P}rop$ be a set of propositional atoms such that $vars(A) \subseteq S$. Choose an approximate partition such that $inc(\mathcal{P}rop) = \emptyset$ and $int(\mathcal{P}rop) = S$. Then A is an S-1 consequence of U iff A is an approximate logical consequence of U.*

Both theorems can be easily proved by using the following observations:

- if $\mathrm{vars}(A) \subseteq int(\mathcal{P}rop)$, then A is a 'classical' formula and therefore our definition and Schaerf and Cadoli definition of logical consequence coincide.

- we can transform an approximate counter-model for A into an S-3 or an S-1 counter-model for A, mapping interesting variables in the obvious way, incoherent literals $\mathtt{f}.q$ and $\mathtt{t}.q$ both in $\top$ and unknown literals both in $\bot$. The other direction is also trivial.

Notice that Theorem 6.34 does not hold if A is an arbitrary formula (but then S-3 entailment is not tractable anymore).

We can also prove that the ad-hoc algorithm given by Schaerf & Cadoli [155, page 269] is just a traversal of a tableau proof.

Consider the following proving techniques:

Technique 6.7 *Let $\{ p_1, \ldots p_m \}$ be an an enumeration of the propositions in $int(\mathcal{P}rop)$. Following the enumeration, apply the cut-rule (§2.3) to $\mathtt{f}.p_i$ and $\mathtt{t}.p_i$ then simplify the resulting branches with p_i.*

Technique 6.8 *Visit the tableau proof in* depth first, *discarding branches which are classically contradictory.*

$$\frac{\vdash_{\mathtt{BCP}} U \Rightarrow A}{\vdash^k_{\mathtt{BCP}} U \Rightarrow A}\,BCP \qquad \frac{\vdash^k_{\mathtt{BCP}} U \Rightarrow B_k \qquad \vdash^k_{BCP} U \cup \{\, B_k \,\} \Rightarrow A}{\vdash^k_{BCP} U \Rightarrow A}\,k-cut$$

where B_k is a clause with at most k distinct literals.

Figure 6.3: Rules for k-BCPfamily of Reasoners

The algorithm of [155] uses essentially those two techniques for the construction of a tableau proof, choosing the set of interesting variables and postponing the decision whether other variables are incoherent or unknown.

We use a preliminary definition which is not in [155] but which makes the description of the algorithm simpler. Let $S = int(\mathcal{P}rop)$, we say that the assumptions U are *potentially unsatisfiable* if setting $unk(\mathcal{P}rop) = \mathcal{P}rop - S$ implies that $\mathtt{meaning}(\mathtt{t}.B) = unk$ for some $B \in U$. Intuitively, when this happens we cannot find an approximate model for U (nor an S-1 interpretation) because of proposition 6.4.

Fact 6.35 *The $S-1/S-3$-algorithm in [155] constructs a tableau proof for $int(\mathcal{P}rop) = S$ using techniques T.6.7 and T.6.8 and terminating as follows:*

- *it stores the first branch (with choice points) which would be approximately open by setting $inc(\mathcal{P}rop) = \mathcal{P}rop - S$ (if one exists);*
- *if U is potentially inconsistent it stops the search;*
- *otherwise it continues until a classically open branch is found (if one exists).*

In subsequent calls we can restart from the last branch returned.

6.6 Dalal Family of Anytime Reasoners

A family of anytime reasoners based on boolean constraint propagation BCP have been proposed by Dalal [38, 37]. Also this formalism is restricted to clausal normal form, both for the assumptions U and the theorem to be proved A. The main intuition is to use a basic tractable inference mechanism (indeed BCP) and make it complete by using cut [37] or compilation [37, 47].

The family of sound but possibly incomplete anytime reasoners $\vdash^k_{\mathtt{BCP}} U \Rightarrow A$ make uses of the rules specified in Fig. 6.3. Notice that U is a set of clauses and A and B_k are clauses.

We have already discussed in §2.5 the basic inference mechanism of BCP and noted that it is identical to the unit rule of DPLL, or unit resolution, and therefore subsumed by simplification with the *dneg* rule. So we need simply to apply the cut rule (§2.3) in the following way:

$$\frac{}{\texttt{f}.B_k \qquad \texttt{t}.B_k}\, k - cut \qquad \begin{array}{l}\text{where } B_k \text{ has at most } k \text{ distinct literals} \\ \\ \text{and } B_k \text{ is a subformula of } U\end{array}$$

Then it is easy to prove the following result:

Theorem 6.36 *The clause A is a logical consequence of the set of clauses U for the reasoner k-BCP iff there is a tableau proof for A with U which only uses dneg, simp and $k - cut$.*

If we do not want to use cut then the transformation of transformation of an (equivalent) $\vdash^k_{BCP}$ derivation into a derivation without cut is simply an application of cut-elimination [70, 163].

We can also map this proof into an approximate proof in the following way: chose an approximate partition by setting $unk_k(\mathcal{P}rop) \doteq \emptyset$ and $int_k(\mathcal{P}rop)$ as the set of variables such that the corresponding literals occurs along the branches corresponding to BCP-parts of the proof. Dually, for non valid formulae, take for $int_k(\mathcal{P}rop)$ the literals occurring in the equivalents classes defined by $\vdash^k_{BCP}$ in [38].

Another interesting construction is possible which

- is independent from the particular derivation $\vdash^k_{BCP} U \Rightarrow A$ used,
- gives an example of a possible construction of the $int_k(\mathcal{P}rop)$ sets of the anytime algorithm described in §6.3.

$$\begin{aligned} I_k^{(0)} &= \text{vars}(A) \\ I_k^{(n+1)} &= I_k^{(n)} \cup \{\text{vars}(B) \mid B \in U \text{ and } \mid \text{vars}(B) \setminus I_k^{(n)} \mid \leq k+1\} \\ I_k &= \cup_n I_k^{(n)} \end{aligned}$$

Once int_k is computed we do not need to restart from scratch and can simply put $I_{k+1}^{(0)} = I_k$. The computation of $int_k(\mathcal{P}rop) = I_k$ is clearly polynomial in $|U \cup \{A\}|$.

Theorem 6.37 *Let $U \cup \{A\}$ be a set of clauses and for every integer k let AP_k be the approximate partition with $unk_k(\mathcal{P}rop) = \emptyset$ and $int_k(\mathcal{P}rop) = I_k$, then $\vdash^k_{\mathsf{BCP}} U \to A$ implies $\vdash^{apx_k}_{\mathsf{PC}} U \Rightarrow A$.*

The proof (Appendix B.1) uses cut-elimination: derive a `BCP`-proof, eliminate cut and show that theliterals occuring in the proof are contained in I_k.

It is also interesting to note that there is an alternative proof based on the semantics for `BCP` which has been proposed by Dalal [38]. In this case we must revert the direction of the proof and show that we can transform an approximate model for our tableaux calculus into a model for `BCP`.

Chapter 7

Approximate Modal Tableaux

We lift the approximation method to modal and dynamic logics after a brief review of the approximations methods for PSPACE *problems.*

Modal logics have two independent sources of complexity: logical omniscience and introspection. We discuss an approximate semantics to tame both of them, by merging the propositional approximation with a new method tailored for modal logics.

The design of anytime tableau methods makes use of the general machinery we have used for classical modal tableau and is followed by the analysis of the semantical and computational guarantee our anytime procedure can offer on the quality of the solution.

7.1 Approximation Methods for PSPACE-hard Problems

The study of approximation methods for modal logics, for deduction or satisfiability, has not received the same attention given to propositional logic.

In theoretical computer science the study of approximating satisfiability for PSPACE-complete problems (such as modal logics) is relatively recent, and very few algorithms exists such as those developed by Marathe et al. [108, 109]. An introductory review can be found in the paper by Condon [29].

In a similar fashion of propositional approximations, the algorithmic approach has focused on satisfiability. Reporting back these methods to model theoretic or proof theoretic intuitions on modal logics and Kripke-models is rather difficult. Loosely speaking, given a formula and a description of a world of a Kripke model the approximation algorithms try to determine whether the world really belongs to the model of the formula.

The study of approximating deduction for multi modal logics has received more, but still limited, attention in artificial intelligence.

If we use modal logics for modeling knowledge and beliefs, we found that two independent sources of complexity are present. One is *unbounded logical omniscience*, which has been tackled already in the chapter about propositional approximation. The other is a new characteristic of modal logics: *unbounded logical introspection*. At an intuitive level, it boils down to the ability of an agent of reasoning about his beliefs of someone else's beliefs about somebody's else beliefs about ....

A large number of the works cited in the chapter on propositional approximation (§6.1) also use modal operators. For instance Levesque [104], Lakemeyer [99] and Fagin et al. [56] have modal operators for explicit (and tractable) beliefs and implicit (but intractable) beliefs. Yet, these operators are not used and the tractability results are always restricted to sentences of the form $[a]\, A \supset [b]\, B$ where A and B are propositional sentences. In practice, modal operators are just 'place-holders' to distinguish the query from the knowledge base.

Also when full introspection is allowed, as in the work of Lakemeyer [100], we are given logics of belief with the axioms of $\mathsf{S5}$ (for one agent) which we known to be in NPTIME.

So, these works do not use the full expressivity of modal operators and provide us only with a propositional approximation. It is not by chance that in these works we always find mentioned the problem of unbounded logical omniscience but never the problem of unbounded logical introspection: the problem is simply not considered.

Fully tractable inference systems for the description logic $\mathcal{ALC}$, a variant of multi modal logic K, has been proposed by Patel-Schneider [140] where a full fledged four-valued semantics is given to achieve tractability.

The study of incremental modal approximations has also been made by Schaerf & Cadoli [155] where no semantics is given and approximation is essentially a syntactic rewriting procedure. The major idea of that work was to place a limit on the nesting of existential (possibility-like) and universal (necessity-like) quantifiers. We extend this idea further in a multi-directional approximations with a clear semantics based on Kripke models.

7.2 An Approximate Semantics

We introduce three levels of approximations: we upgrade the propositional approximation to modal logics, introduce an approximation tailored to modal logic, and finally combine the two approximations.

7.2.1 Upgrading the propositional approximation

As in §6, we start with an *approximate partition*. Again, we only revise the forcing relation $\Vdash$ for propositional variables.

ν_0	$\nu(a)$
unk	*int*
inc	*inc*
int	*int*

π_0	$\pi(a)$
unk	*unk*
inc	*inc*
int	*int*

Table 7.1: Superficial Meaning of Modal Formulae

Given an L-model $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$, and an approximate partition ap, an approximate forcing relation is defined as in section 3.2 with the exception of the conditions on propositional atoms $p \in \mathcal{P}rop$:

$$\begin{array}{lll} \mathcal{M}, w \Vdash \mathtt{t}.p & \text{iff} & \text{either } w \in (p)^{\mathcal{I}} \text{ and } p \in int(\mathcal{P}rop) \\ & & \text{or } p \in inc(\mathcal{P}rop) \\ \mathcal{M}, w \Vdash \mathtt{f}.p & \text{iff} & w \notin (p)^{\mathcal{I}} \text{ and } p \in int(\mathcal{P}rop) \\ & & \text{or } p \in inc(\mathcal{P}rop) \end{array}$$

Then we upgrade also the notion of type of a formula defined in Table 6.1 to the modal connectives. This is shown in Table 7.1.

The superficial meaning of a formula can be determined in polynomial time in the size of the formulae and in the size of $\mathcal{P}rop$.

The equivalent of Prop. 6.3 and Prop. 6.4 are also proved by induction.

Proposition 7.1 *If* `meaning`(φ) *is incoherent then in every model, every world w satisfies φ.*

Proposition 7.2 *If* `meaning`(φ) *is unknown then in every model, no world w satisfies φ.*

The definition of logical consequence remains the same of section 6.2.

A consequence of these properties is that *logical omniscience* only holds for interesting formulae. With incoherent concepts our reasoning may be incomplete and therefore

$$\not\models^{apx}_{L} [a]\,(p \supset q) \wedge [a]\,p \supset [a]\,q$$

if $q \in int(\mathcal{P}rop)$ but $p \in inc(\mathcal{P}rop)$. On the contrary, with unknown concepts we may draw unsound conclusions. For instance, switching p from incoherent to $p \in unk(\mathcal{P}rop)$ implies that

$$\models^{apx}_{\mathsf{L}} [a]\,(p \vee q) \supset [a]\,q.$$

When some propositions are considered unknown than we can obtain incomplete but possibly unsound approximations while considering propositions incoherent leads to sound but incomplete approximations. Here we can have both. For instance if $p \in unk(\mathcal{P}rop)$ and $q \in inc(\mathcal{P}rop)$ then we obtain that

$$\not\models^{apx}_{\mathsf{L}} [a]\,(p \vee q) \supset [a]\,q.$$

7.2.2 Modal approximation

The partitions of propositional letters is not sufficient to tame the complexity of modal logics. Indeed it has been proved that validity checking remains PSPACE-complete even if we have only one interesting atom:

Theorem 7.3 ([81]) *Deciding* K*-validity of* A *is* PSPACE*-hard if there is at least a variable in* $vars(A)$*.*

The problem, pointed out by Rantala [147] and recently by Levesque [104], Delgrande [48], and Lakemeyer [99], is the presence of a (possibly exponential) number of possible worlds where all laws of classical logic hold.

As we have already noted, a common approach has been the weakening of the laws of (classicam) logic, i.e. using a logic weaker than classical logic (§6.1). We have already argued that this is not what we want: in the phrase "unbounded logical introspection" we are satisfied by the word "logical", and the word "introspection". It is the "unbounded" that creates problem.

Intuition. Given a Kripke model $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ we introduce a partition of worlds into three sets. Again we have incoherent, unknown and interesting (normal) worlds. We limit the "unbounded" by imposing that "too many" steps of introspection always lead to outside interesting worlds.

Remark. The notion of incoherent and unknown worlds is substantially different from the notion of "impossible" world used in the literature by Goranko [76] and Levesque [106].

Traditional impossible world are "inaccessible" worlds, i.e. worlds which are not reachable via the accessibility relation $\mathcal{R}_a$. On the contrary, our incoherent and unknown worlds are reachable, and the whole point of our approximation mechanism is indeed that, after too many introspection steps, an agents always reaches one of those worlds.

Our notion of non-interesting worlds is therefore closer to the notion of "non-normal" worlds introduced by Kripke [97] (see also Fitting [62]) and later utilized and perfectioned by a number of researchers [100, 55, 56]. In particular, the intuition behind incoherent and unknown worlds is close to the similar notion of "incoherent" and "incomplete" worlds of Fagin, Halpern & Vardi [56][page 212], although our use and formalization is different.

Intuition. At incoherent worlds we admit any kind of statement, including contradictory statements, while at an unknown worlds we do no admit any statement whatsoever.

For sake of simplicity[1], we *restrict the treatment to multi modal logic* K. We discuss the extension to other logics and PDL at the end of the chapter.

We say that an $\langle W, \{ \mathcal{R}_a \}_{a \in \mathcal{A}}, \mathcal{I} \rangle$ is a K-*approximate model* if there is a partition of the worlds such that from an incoherent (unknown) world we only access incoherent (unknown) worlds. Formally,

$$\begin{array}{lll} w\mathcal{R}_a v \text{ and } w \in inc(W) & \text{implies} & v \in inc(W) \\ w\mathcal{R}_a v \text{ and } w \in unk(W) & \text{implies} & v \in unk(W) \end{array}$$

We revise the assignment to propositions, leaving rest is unchanged (§3.2):

$$\begin{array}{lll} \mathcal{M}, w \Vdash \mathtt{t}.p & \text{if} & w \in int(W) \text{ and } w \in (p)^{\mathcal{I}} \\ & \text{elseif} & w \in inc(W) \\ \mathcal{M}, w \Vdash \mathtt{f}.p & \text{if} & w \in int(W) \text{ and } w \notin (p)^{\mathcal{I}} \\ & \text{elseif} & w \in inc(W) \end{array}$$

The following properties are valid with the above definitions:

Proposition 7.4 *If $w\mathcal{R}_a v$ implies $v \in inc(W)$ and there is at least a v such that $w\mathcal{R}_a v$ then $w \Vdash \nu(a)$ and $w \Vdash \pi(a)$ for all $\nu(a)$ and $\pi(a)$.*

Proposition 7.5 *If $w\mathcal{R}_a v$ implies $v \in unk(W)$ and there is at least a v such that $w\mathcal{R}_a v$ then $w \nVdash \nu(a)$ and $w \nVdash \pi(a)$ for all $\nu(a)$ and $\pi(a)$.*

Notice that w may also be an interesting world.

The intuition is that at the "border" of the cluster of incoherent worlds everything is possible and everything is necessary[2]. If all worlds nearby are unknown (and there is at least one of such worlds) then nothing is necessary and nothing is possible.

[1] Also because this is the only thing we need for reasoning about the knowledge representation language $\mathcal{ALC}$ or the DEC-SRC calculus for access control.

[2] In traditional non-normal worlds [97, 62] we would have had that everything was possible and nothing was necessary.

7.2.3 Bounding self-introspection

For sake of simplicity we consider first the single agent case.

To approximate the introspection capabilities of an agent, we introduce the notion of a *bounded agent* as a triple $\langle a, k_a, type \rangle$ where k_a is an integer and *type* is either c (for credulous) or s (for skeptical). We say that agent a is *k_a-skeptical* in the former case and k_a-credulous in the latter.

Intuition. With a bounded agent we specify the limit of introspection that she may have and her credulous or skeptical attitude attitude towards her beliefs once the limit of introspection is reached.

Definition 7.6 *An approximate model* $\langle W, \{ \mathcal{R}_a \}_{a \in \mathcal{A}}, \mathcal{I} \rangle$ *is a model for a bounded agent* $\langle a, k_a, type \rangle$ *iff*

- *if a is* k_a-skeptical *then every simple $\mathcal{R}_a$ path with more than $k_a + 1$ worlds terminates into an incoherent world.*
- *if a is* k_a-credulous *then every simple $\mathcal{R}_a$ path with more than $k_a + 1$ worlds terminates into an unknown world.*

The intuition is that a k_a-skeptical agent is logical omniscient and able of introspective reasoning but only up to k_a levels of introspection. Afterwards, she will give up and assume that an incoherent state is reached.

Unfortunately, this definition is too strong: in a similar way of [56], we lose too many valid formulae (with skeptical agents) or we get too many (with credulous agent). Moreover there are many ways in which a possible world can be reached from another world. For instance, consider the transitive model $W = \{ w, v, u \}$ with $w \mathcal{R} v \mathcal{R} u$ and $w \mathcal{R} u$. We can reach the world u with 1 or 2 steps of introspection.

The problem is that we have only specified the maximal distance from the real world (where we try to verify our properties) and the incoherent and unknown worlds. Since we have not specified a minimal distance, our agent may well be k_a-skeptical but it may find himself immediately at the border with an incoherent world. On the contrary, we are only interested in worlds which are at *exactly* k_a steps of distance from incoherent (or unknown) worlds.

To avoid this problem we restrict ourselves to *tree-like* models of interesting worlds and introduce the notion of *realistic world*:

Definition 7.7 *Let* $\langle W, \{ \mathcal{R}_a \}_{a \in \mathcal{A}}, \mathcal{I} \rangle$ *be a tree-like approximate model for a bounded agent* $\langle a, k_a, type \rangle$*. A world* $w \in W$ *is* k_a-realistic *if in the graph* $\langle W, \{ \mathcal{R}_a \}_{a \in \mathcal{A}} \rangle$ *the world w it the root of a tree of $\mathcal{R}_a$-arcs such that every simple $\mathcal{R}_a$ path with no more than than $k_a + 1$ worlds only traverse interesting worlds.*

In a nutshell, from a k_a-realistic world we can go down the tree-like Kripke model for at least k steps only traversing interesting worlds and then we end up in one or more uninteresting worlds.

Note that there is no loss of generality for logic K: we known that we can always "unwind" a model into a tree-like model [89, 172]. The model we obtain from our sound and complete tableau procedure for K in chapter 3 is a tree-like model.

We can upgrade the definition of forcing for π formulae:

$$\mathcal{M}, w \Vdash \pi(a) \quad \text{iff} \quad \exists v \in W : w \mathcal{R}_a v \text{ and } \mathcal{M}, v \Vdash \pi_0$$

Now, if w is a k_a-realistic world then v must be a $k_a - 1$-realistic world.

Intuition. Each time we try to satisfy a new π formula we get closer and closer to unknown and incoherent worlds. So we have not an unbounded introspection.

We upgrade the notion of logical consequence (§3.2) because we are only interested in realistic worlds: given a bounded agent $\langle a, k_a, type\rangle$ we say that a K-model $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ for the agent is an *approximate countermodel* for A with local assumptions U if there is k_a-realistic world $w \in W$ which satisfies U and $\mathtt{f}.A$.

Definition 7.8 *Let $\langle a, k_a, type\rangle$ be bounded agent, a formula A is a* K-approximate logical consequence *of local assumptions U (in symbols $\models_{\mathsf{K}}^{k_a - type} U \Rightarrow A$) iff every K-model $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ is not a countermodel for A with local assumptions U.*

For sake of readability we drop $\Rightarrow$ whenever $U = \emptyset$.

Some example can be helpful to clarify the semantics.

Consider a 1-skeptical agent, we have that *plain logical omniscience holds*:

$$\models_{\mathsf{K}}^{1-s} [a]\,(p \supset q) \wedge [a]\,p \supset [a]\,q \tag{7.1}$$

although *unbounded logical omniscience may not hold*:

$$\not\models_{\mathsf{K}}^{1-s} [a]\,([a]\,(p \supset q) \wedge [a]\,p) \supset [a]\,[a]\,q$$

The situation is reversed for 1-credulous agents: (7.1) holds as well but *unsound formulae may become approximately valid.*

$$\models_{\mathsf{K}}^{1-c} [a]\,([a]\,p \vee [a]\,q) \supset [a]\,[a]\,q$$

Still, 1-*credulous agent remain sound* if the reasoning does not require introspection over 1 level of modal nesting. For instance

$$\not\models_{\mathsf{K}}^{1-c} [a]\,[a]\,p \vee [a]\,q \supset [a]\,q$$

We can use k-skeptical agents for an sound, but possibly incomplete, approximation and an k-credulous agents for complete, but possibly unsound, approximations.

7.2.4 Bounding mutual introspection

With many agents around, the picture is more complicated. At first because, even with 1-skeptical (or credulous) agents, reasoning remains hard.

Theorem 7.9 *Let $\langle a, k_a, type\rangle$ and $\langle b, k_b, type\rangle$ be two bounded agents such that $k_a \geq 1$ and $k_b \geq 1$; deciding approximate* K*-validity is* PSPACE*-hard.*

This result can be easily derived by generalizing the construction used by Halpern & Moses [82]: use the same hardness examples and simply replace every connective $[a]$ with $[a]\,[b]\,A$ (we have at least two agents). So we never have a simple $\mathcal{R}_a$-path with more than one consecutive $\mathcal{R}_a$-step and we keep traversing interesting worlds.

Intuition. The problem with many agents is that it is not sufficient to bound the self-introspection capabilities of an an agent. We must also bond its ability of reasoning about somebody else's beliefs.

So we introduce the notion of *family of bounded agents* as the set $\mathcal{F} = \{\,\langle a, k_a, type\rangle\,\}_{a\in\mathcal{A}}$. We assume that there is one and only one triple in $\mathcal{F}$ for every agent a and we say that agent a is k_a-*skeptical* if $\langle a, k_a, s\rangle \in \mathcal{F}$ and it is k_a-credulous if $\langle a, k_a, c\rangle \in \mathcal{F}$.

All definitions that we have introduced for one agent can be easily generalized to this more general notion.

Definition 7.10 *An approximate model $\langle W, \{\,\mathcal{R}_a\,\}_{a\in\mathcal{A}}\,, \mathcal{I}\rangle$ is a model for family of bounded agents $\{\,\langle a, k_a, type\rangle\,\}_{a\in\mathcal{A}}$ iff*

- *for every k_a-*skeptical *agent a every simple path with an initial $\mathcal{R}_a$-arc and with more than $k_a + 1$ worlds terminates into an incoherent world.*

- *for every k_a-*credulous *agent a every simple path with an initial $\mathcal{R}_a$-arc and with more than $k_a + 1$ worlds terminates into an unknown world.*

We revise the notion of $\mathcal{F}$-realistic world in a similar way. It only affects the definition of forcing for π-formulae: we converge towards incoherent or unknown worlds no matter the path we follow i.e. no matter the agent responsible for the last introspection step. If we have the following forcing relation:

$$\mathcal{M}, w \Vdash \pi(a) \quad \text{iff} \quad \exists v \in W : w\mathcal{R}_a v \text{ and } \mathcal{M}, v \Vdash \pi_0$$

the world w is a $\mathcal{F}$-realistic world, and we have $\langle a, k_a, type_a \rangle$, then v is a $\mathcal{F}/a$-realistic world where the family $\mathcal{F}/a$ is obtained by $\mathcal{F}$ by setting *for all agents* the bound to $k_a - 1$ and the type to $type_a$.

Beside technical reasons, this definition has a intuitive motivation: when passing from w to v we are performing an introspection step for the agent a. Loosely speaking it is a who is now reasoning about the beliefs of other agents in his own head. If a has a skeptical attitude so he will reason skeptically when reasoning about the belief of other agents.

The definition of countermodel, logical consequence and all other notions remains unchanged.

In this way we *bound the overall introspection capabilities* of an agent. If an agent is either k_a-skeptical or k_a-credulous we say that he is k_a-*bounded.* If $k = \infty$ (or a sufficiently large integer) than we say that a is *unbounded.* Indeed, we do not lose all properties of the good old modal logics for bounded agents with a sufficiently high bound.

Recall the lower bound hb_{K} that we have given in section 4.5 and let n be the cardinality of $\mathrm{Subfml}(\{\, \mathtt{t}.B | B \in U \,\} \cup \{\, \mathtt{f}.A \,\})$.

We can prove that if *every* agent a of the family $\mathcal{F}$ has an introspection bound $k_a > hb_{\mathsf{K}}$, or even better $hb_{\mathsf{K}} > 2^n$ then approximate entailment coincides with classical entailment.

7.2.5 Primary and secondary approximations

At present we have defined two independent approximations: one for propositional part of the logic and an approximation for the modal part.

These approximations can be combined in many ways. The major design decision regards their relative importance. This can be better clarified by an example. We known that incoherent formulae always hold in interesting worlds and interesting formulae never hold in unknown worlds. So what happens to incoherent formulae in unknown worlds?

One choice is to make the propositional part predominant: incoherent proposition always hold in a world, no matter its type. The opposite choice gives more importance to worlds: unknown worlds never satisfy a formula, no matter its type. In the former case the propositional approximation is a primary approximation and the modal one is a secondary approximation.

The choice is mostly a matter of taste, since there is no clear semantical motivation behind one case or the other. Hence, we choose the modal approximation as a primary approximation because it makes computation simpler.

Once this decision has been made, we still have a number of degrees of freedom in particular we can fix the propositional partitions for the whole model or make them vary from world to world.

The choice of changing interesting formulae from world to world is made by Fagin & Halpern [55] for the logic of awareness. This may not have much sense with only one agent, but add a degree of flexibility for multi-agent systems: not all agents may consider the same propositions equally interesting.

Deduction mechanism for this form of approximation can be found in Massacci [115] for sound approximations only, with a very preliminary version of the propositional approximation developed in this dissertation.

To keep exposition simple we assume that the approximate partition of propositional letter is the same for all agents. We need only a small revision in the definition of forcing:

$$
\begin{array}{lcl}
\mathcal{M}, w \Vdash \mathtt{t}.p & \text{if} & w \in int(W) \text{ and } p \in int(\mathcal{P}rop) \text{ and } w \in (p)^{\mathcal{I}} \\
 & \text{elseif} & w \in int(W) \text{ and } p \in inc(\mathcal{P}rop) \\
 & \text{elseif} & w \in inc(W) \\
\mathcal{M}, w \Vdash \mathtt{f}.p & \text{if} & w \in int(W) \text{ and } p \in int(\mathcal{P}rop) \text{ and } w \notin (p)^{\mathcal{I}} \\
 & \text{elseif} & w \in int(W) \text{ and } p \in inc(\mathcal{P}rop) \\
 & \text{elseif} & w \in inc(W)
\end{array}
$$

We need also to modify the *meaning* of a formula to take into account the fact that we have bounded (credulous or skeptical) agents. Now we have three arguments $\mathtt{meaning}(a, k, \varphi)$: the first is the agent a for which the formula is evaluated, the second the current bound of introspection and the last is the signed formula. If the agent is not specified we use the wild-card $*$.

The function is unchanged over propositional connectives (see Table 6.1), it only changes over modal connectives. In the initial realistic world we may have not yet specified the agent nor the bound. So, when we find a modal operatorduring a quick evaluation for a family $\{\, \langle a, k_a, type\rangle \,\}_{a\in\mathcal{A}}$ we work as follows:

$$
\begin{array}{rcl}
\mathtt{meaning}(*, *, \pi(a)) & = & \mathtt{meaning}(a, k_a, \pi(a)) \\
\mathtt{meaning}(*, *, \nu(a)) & = & \mathtt{meaning}(a, k_a, \nu(a))
\end{array}
$$

Once we have fixed the basic case we proceed by induction using the values of Table 7.1 and decreasing the value of k in the recursive call. This is shown in Fig. 7.1

Intuitively, we are just capturing in another way what we have already stated in the previous paragraph with Prop. 7.1 and Prop. 7.2: at the bor-

$$\begin{aligned}
\mathtt{meaning}(a,0,\pi(a)) &= \begin{cases} inc & \text{if } a \text{ is skeptical} \\ unk & \text{if } a \text{ is credulous} \end{cases} \\
\mathtt{meaning}(a,k+1,\pi(a)) &= \begin{cases} unk & \text{if } \mathtt{meaning}(a,k,\pi_0) = unk \\ inc & \text{if } \mathtt{meaning}(a,k-1,\pi_0) = inc \\ int & \text{otherwise} \end{cases} \\
\mathtt{meaning}(a,0,\nu(a)) &= \begin{cases} inc & \text{if } a \text{ is skeptical} \\ int & \text{if } a \text{ is credulous} \end{cases} \\
\mathtt{meaning}(a,k+1,\nu(a)) &= \begin{cases} inc & \text{if } \mathtt{meaning}(a,k,\nu(a)) = inc \\ int & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 7.1: Meaning of Formulae for Bounded Agents

der with incoherent worlds ($k = 0$) everything is possible and everything is necessary. The opposite happens with unknown worlds.

7.3 Approximate Modal Tableaux

Again, our general deductive machinery makes it possible to devise anytime algorithms with only minor changes to the rules devised in section 3.3.

7.3.1 Approximate rules

In the rest of the section we assume that we have fixed an approximate partition of the set $\mathcal{P}rop$ and, for each agent $a \in \mathcal{A}$ decided whether it is k_a-credulous or k_a-skeptical for some prefixed constant k_a.

We say that a prefix σ *belongs to an agent* a whenever $\mathrm{first}(\sigma) = a$ (see §3.3). Intuitively, this means that the first introspection step is due to a, or, in other words that a is the "main" agent responsible for this chain of reasoning which lead to the possible world σ.

We change the definition of reduced formulae and of closed branches:

Definition 7.11 *Let $\mathcal{F}$ be a family of bounded agents and apx an approximate partition. The prefixed formula $\sigma : \varphi$ it is* approximately reduced *iff*

- *either it is reduced according the classical Definition 3.5,*
- *or $\sigma = 1$ and* $\mathtt{meaning}(*,*,\varphi) = inc$*;*
- *or σ belongs to a and* $\mathtt{meaning}(a, k_a + 1 - |\sigma|, \varphi) = inc$*.*

The intuition behind the last clause of the definition is given by the following observation: each time we pass from σ to $\sigma.b.n$ it is like performing an introspection step. So the length of σ is an indicator of the number of introspection steps that we have already done: we have made $|\sigma| -1$ steps (we have the initial prefix 1 to account for, see §3.3). Since the overall bound of the agent is k_a this means that we are only left with $k_a + 1 - |\sigma|$ introspection steps. Then we use this shorter bound for the approximate valuation of φ.

Definition 7.12 *Let $\mathcal{F}$ be a family of bounded agents and apx an approximate partition. A* branch $\mathcal{B}$ *is* approximately contradictory *iff*

- *either $\mathcal{B}$ is contradictory according the classical Def. 3.3 due to two interesting formulae φ, $\overline{\varphi}$,*
- *or $\sigma = 1$ and* $\texttt{meaning}(*, *, \varphi) = unk$*;*
- *or σ belongs to a and* $\texttt{meaning}(a, k_a + 1 - |\sigma|, \varphi) = unk$*.*

Technique 7.1 *Reduce a formula $\sigma : \varphi$ only if* $\texttt{meaning}(\mathit{first}(\sigma), k_{\mathbf{first}(\sigma)} + 1 - |\sigma|, \varphi) = int$*.*

The same remarks about the applicability of simplification (either propositional or modal) that we have made in §6.3 can be made here. We may need to constraint simplification to interesting formulae if we care about the approximate semantics and use the unbounded (but still polynomial version) if we only care about speed and the classical semantics.

The definition of closed and open tableau remain unchanged (beside prefixing them with the word "approximate"). We say that a tableau proof is a *classical proof* if only the standard condition for closing it have been used. In a similar way we say that an open branch is a *classical countermodel* if it has been considered reduced only according the standard notion.

7.3.2 Design of an anytime proof procedure

We have now the formal machinery to construct an approximating sequence of proofs of increasing precision as in Fig. 1.1. The anytime algorithm is Alg. 7.2.

If we time out after $\mathcal{T}_k$, we have available either an approximate proof (all branches of $\mathcal{T}_k$ are approximately closed) or an approximate counter model (an open leaf of $\mathcal{T}_k$). On the contrary, with sufficient resources we can always arrive at a stage where every agents in $\mathcal{F}_k$ has a bound k_a sufficiently large to guarantee that a classical proof (or countermodel) is found.

Remark. For the single agent case, we have shown that there is a polynomial bound on the maximum length of prefixes if $G = \emptyset$. When we reach this bound we stop increasing k_a and use the termination conditions in section 4.4.

Figure 7.2: Algorithm

Let U be a set of modal assumptions and A a formula in a multi-modal logic then

1. **choose** an approximate partition so that the set of interesting variables $I_0 = int(\mathcal{P}rop)$ is small (e.g. logarithmic in the size of U and A);
2. **choose** a family of bounded agents $\mathcal{F}_0 = \{ \langle a, k_a, type \rangle \}_{a \in \mathcal{A}}$ where k_a is a small prefixed constant, and decide whether $type_a$ is $c$redulous or $s$keptical;
3. **compute** the an approximate K-tableau $\mathcal{T}_k$;
4. **if** the approximate tableau $\mathcal{T}_k$ is a classical proof or a classical counter-model **then** stop;
5. **else if** not timeout
   - **find** the part of $\mathcal{T}_k$ where non classical reduction or closure rules
   - **increase** the precision so that at least one approximately reduced or contradictory formula change status by
     - **either** enlarging the set of interesting variables from I_k to I_{k+1};
     - **or** increasing the k_a for an agent a in the family $\mathcal{F}_k$ generating a new family $\mathcal{F}_{k+1}$;
   - **return to** (3) for a new $\mathcal{T}_{k+1}$;

7.4 Quality and Complexity of Approximation

Also in this case we are able to prove the following properties:

- the approximate tableau method is sound and complete wrt the approximate semantics;
- the "semantical" quality of approximation increases when the set of interesting propositions or the bound k_a for an agent a are increased;
- the complexity of obtaining a better approximation increases according the requirements of anytime algorithms.

7.4.1 Approximate soundness and completeness

For sake of readability we only prove the case for the modal approximation without propositional approximation. This means that in the sequel of this section we assume that $int(\mathcal{P}rop) = \mathcal{P}rop$

The soundness and completeness of the *whole* procedure is a straightforward combination of this proof with the approximate soundness and completeness proofs in §6.4.

The soundness proof follows the same lines of the modal proof:

- set a mapping between prefixes and worlds on a putative Kripke model;
- prove a safe extension lemma;
- prove a safe closure lemma.

We re-use all definitions about soundness and completeness we have introduced in §3.6 without repeating them. We simply prefix to every definition the word "approximate".

Definition 7.13 *Let $\mathcal{F} = \{\langle a, k_a, type\rangle\}_{a\in\mathcal{A}}$ be a family of bounded agent, and $\mathcal{M} = \langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ be an* K*-model for this family. A tableau branch $\mathcal{B}$ is approximately* K*-SAT for $\mathcal{M}$ with local assumptions U iff there is an SST-interpretation $\imath()$ such that*

1. *the world $\imath(1)$ is F-realistic and satisfies U in $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$;*
2. *for every prefixed formula $\sigma : \varphi$ in $\mathcal{B}$, the world $\imath(\sigma)$ satisfies φ.*

A tableau is K*-SAT if at least a branch is such.*

The key new step is the safe closure lemma (see appendix B.2):

Lemma 7.14 *If $\mathcal{T}$ is a SAT tableau, then one SAT branch is not approximately contradictory.*

The proof (by induction) uses the same arguments we have used for the propositional approximation plus a new argument for the modal approximation. The only difficult case is the case of a $\pi(b)$-formula which has closed a branch.

Here we must use in essential way the restriction on the forcing relation for π-formulae caused by the particular nature of approximate models we use: after each introspection step we get closer to an unknown world. So we can use induction on the proof and we are done.

Without this restriction the proof would fail. It may be that a π formula is satisfied just because the fulfilling introspection step brings us farther away rather nearer to unkown worlds. This would not be a big loss since we we could lose the soundness of the approximate procedure wrt the approximate semantics but we could still prove the anytime guarantees and the quality of the approximations of the next subsection.

Then we can prove that the system is sound wit a standard argument

Theorem 7.15 (Approximate Soundness) *Given an approximate partition, if the formula A has a approximate tableau proof with assumptions U then it is an approximate logical consequence of U.*

For the completeness we need to prove the model existence theorem:

Theorem 7.16 *If $\mathcal{B}$ is an approximately open branch for a family of bounded agents then there is a model $\mathcal{M}$ for the family such that $\mathcal{B}$ is SAT for $\mathcal{M}$.*

The construction of the model is substantially identical to the construction used in modal logic (Thm. 3.21). We only have *two extra worlds* one subsuming all unknown worlds and one for all incoherent worlds and construct the model as follows:

$$\begin{array}{rcl} W & \doteq & \{ \langle\sigma\rangle \ : \ \sigma \text{ is present in } \mathcal{B} \} \cup \{ \langle unk\rangle , \langle inc\rangle \} \\ \langle\sigma\rangle \, \mathcal{R}_a \, \langle\sigma^*\rangle & \text{iff} & \sigma \rhd_a \sigma^* \\ \langle\sigma\rangle \, \mathcal{R}_a \, \langle inc\rangle & \text{iff} & \text{first}(\sigma) = b \text{ and } |\sigma| = k_b \text{ and } b \text{ is skeptical} \\ \langle\sigma\rangle \, \mathcal{R}_a \, \langle unk\rangle & \text{iff} & \text{first}(\sigma) = b \text{ and } |\sigma| = k_b \text{ and } b \text{ is credulous} \\ & & \\ (p)^{\mathcal{I}} & \doteq & \{ \sigma \mid \sigma : \mathtt{t}.p \in \mathcal{B} \} \end{array}$$

Moreover we have that $\langle unk\rangle \, \mathcal{R}_a \, \langle unk\rangle$ and $\langle inc\rangle \, \mathcal{R}_a \, \langle inc\rangle$ for all a. The proof then follows exactly the argument of Thm. 3.21 for classical modal logic.

Now we can prove that the system is complete with a standard argument.

Theorem 7.17 (Approximate Completeness) *Given an approximate partition, if the formula A is an approximate logical consequence of assumptions U then it has a approximate tableau proof with U.*

A corollary of the these theorems is that each step of the anytime proof procedure is sound and complete for the approximation

Corollary 7.18 *For every sequence of families of bounded agents $\mathcal{F}_k$ chosen by the anytime algorithm, $\vdash^{\mathcal{F}_k}_{\mathsf{PC}} U \Rightarrow A$ if and only if A is an approximate logical consequence of assumptions U for the kth family of bounded agents $\mathcal{F}_k$.*

7.4.2 The semantical quality of approximation

The next steps shows that, given enough time and motivation we are guaranteed to find an *the* correct (classical) answer. Let d be maximum nesting of modal operators in $\text{Subfml}(\{ \mathtt{t}.B | B \in U \} \cup \{ \mathtt{f}.A \})$ then we have:

Theorem 7.19 (Convergence Guarantee) *If $int_k(\mathcal{P}rop) = \mathcal{P}rop$ and for every agent a in the family $\mathcal{F}_k$ it is $k_a \geq d + 2$, then $\vdash^{k}_{\mathsf{K}} U \Rightarrow A$ if and only if A is classical consequence of U.*

If $int_k(\mathcal{P}rop) = \mathcal{P}rop$ then all propositional letters are considered as classical propositional letters and we have already shown in section 4.5 the decidability result for such a bound which can be easily generalized to the multi-agent case.

The algorithm may also terminate with a classical proof or classical countermodel *before* reaching this stage. In some cases we can prove that we do not have to iterate further.

Proposition 7.20 (Sound Approximation) *Let $\mathcal{F}_k$ be a family of bounded agents and AP_k an approximate partition. If $\vdash_{\mathsf{K}}^{\{\langle a,k_a,type\rangle\}_{a\in\mathcal{A}}} U \Rightarrow A$ and no formula φ with unknown type has been used to close a branch of $\mathcal{T}_k$, then A is a (classical) logical consequence of U.*

Proposition 7.21 (Complete Approximation) *Let $\mathcal{F}_k$ be a family of bounded agents and AP_k an approximate partition. If $\not\vdash_{\mathsf{PC}}^{F-k} U \Rightarrow A$ and there is an open branch of $\mathcal{T}_k$where no formula φ with incoherent type is present, then A is not a (classical) logical consequence of U.*

Both results are an immediate consequence of the classical soundness and completeness theorem for modal tableau with simplification (§3.6).

Establishing the *sematical direction* of approximation is more complicated. At first we assume that *the propositional approximation is fixed*, so that we need only to reason about the modal aspects of approximation.

Theorem 7.22 (Modal Completeness Increase) *Let $\mathcal{F}_k$ and $\mathcal{F}_{k+1}$ be two families of bounded agents such that the bounds on credulous agents are identical between the two families and that if $\langle a, k_a, s\rangle \in \mathcal{F}_k$ then $\langle a, k'_a, s\rangle \in \mathcal{F}_k$ and $k_a \leq k'_a$. If $\not\vdash_{\mathsf{K}}^{\mathcal{F}_k} U \Rightarrow A$ implies $\not\vdash_{\mathsf{K}}^{\mathcal{F}_{k+1}} U \Rightarrow A$.*

Theorem 7.23 (Modal Soundness Increase) *Let $\mathcal{F}_k$ and $\mathcal{F}_{k+1}$ be two families of bounded agents such that the bounds on skeptical agents are identical between the two families and that if $\langle a, k_a, c\rangle \in \mathcal{F}_k$ then $\langle a, k'_a, c\rangle \in \mathcal{F}_k$ and $k_a \leq k'_a$. If $\not\vdash_{\mathsf{K}}^{\mathcal{F}_k} U \Rightarrow A$ implies $\not\vdash_{\mathsf{K}}^{\mathcal{F}_{k+1}} U \Rightarrow A$.*

Both arguments are identical to the arguments we have used for propositional approximations in §6.4: credulous agents are responsible for the approximate closure of a branch and they do not change for the first theorem. Skeptical agents are responsible for approximate countermodels and they do not change in the statement of the second theorem.

In the *general case* we have *four possible directions* in which we can increase the soundness and the completeness of our procedure:

An increase in soundness can be achieved by setting $\langle a, k_a, s\rangle \in \mathcal{F}_k$ iff $\langle a, k_a, s\rangle \in \mathcal{F}_{k+1}$ and $inc_k(\mathcal{P}rop) = inc_{k+1}(\mathcal{P}rop)$ and then

- $\langle a, k_a, c\rangle \in \mathcal{F}_k$ and $\langle a, k'_a, c\rangle \in \mathcal{F}_{k+1}$ with $k_a \geq k'_a$;
- $unk_k(\mathcal{P}rop) \supseteq unk_{k+1}(\mathcal{P}rop)$;

An increase in completeness can be achieved by setting $\langle a, k_a, c\rangle \in \mathcal{F}_k$ iff $\langle a, k_a, c\rangle \in \mathcal{F}_{k+1}$ and $unk_k(\mathcal{P}rop) = unk_{k+1}(\mathcal{P}rop)$ and then

- $\langle a, k_a, s\rangle \in \mathcal{F}_k$ and $\langle a, k'_a, s\rangle \in \mathcal{F}_{k+1}$ with $k_a \geq k'_a$;
- $inc_k(\mathcal{P}rop) \supseteq inc_{k+1}(\mathcal{P}rop)$;

7.4.3 Computational complexity

We need a preliminary property whose proof is immediate

Proposition 7.24 *The verification of* `meaning`(a, k, φ) *requires polynomial time in the size of* φ*, the value of* k *and* $|\mathcal{A}|$.

Then we restrict the algorithm used to generate the family $\mathcal{F}_{k+1}$ from the family $\mathcal{F}_k$.

Technique 7.2 *For the generation of* $\mathcal{F}_{k+1}$*, apply a polynomial time algorithm to the non interesting formulae of* $\mathcal{T}_k$.

Notice that the algorithm may also take constant time: it may start form a fixed enumeration of agents and increase their introspection bounds in turn to generate $\mathcal{F}_0$, $\mathcal{F}_1$ and so on.

A careful observation of the completeness proof reveals the following property:

Lemma 7.25 *Let* b_k *be the maximum bound for every agent in the family* $\mathcal{F}_k$ *bounded agents, if* A *has countermodel for* U *then it has a countermodel with at most* $2^{b_k} + 3$ *worlds.*

Then we can prove the following

Theorem 7.26 (Anytime Modal Guarantee) *Let* $\mathcal{F}_k$ *be a family of bounded agents and let* b_k *be the maximum bound for every agent in* $\mathcal{F}_k$*. The algorithm which employs techniques T.7.1, T.7.2 always return a tableau* $\mathcal{T}_k$ *where each branch has at most* 2^{b_k} *prefixes.*

Since b_k is fixed, this means that we have reduced the complexity of validity checking from PSPACE to CO-NPTIME.

We can combine the results about propositional and modal approximations into one obtaining an overall anytime guarantee. It is a multiplication of the propositional bound (considering also modal atoms) for every possible prefix.

7.5 Approximation for Modal Logics Extending K

All properties of the accessibility relation we have defined in Table 3.2 must now be restricted to normal, interesting worlds. For instance we say that $\mathcal{R}_a$ is approximately transitive iff $w\mathcal{R}_a v$ and $v\mathcal{R}_a u$ implies $w\mathcal{R}_a u$ for all interesting worlds $w, u, u \in int(W)$.

As expected, the bound k of skeptical and credulous agents has an impact on the positive and negative introspection capabilities of agents. Consider for instance the axiom 4 (Table 3.2) corresponding to positive introspection. Here we only have approximate positive introspection. Positive introspection requires reasoning with formulae with nested $[a]\,A$ and therefore 1-skeptical agents avoid (correctly) any kind of introspection. We need to have at least 2-skeptical agents to perform introspection.

For instance we have the following properties:

$$\not\models^{s1}_{\mathsf{K4}} [a]\,p \supset [a]\,[a]\,p \qquad \models^{s2}_{\mathsf{K4}} [a]\,p \supset [a]\,[a]\,p \qquad \not\models^{s2}_{\mathsf{K4}} [a]\,p \supset [a]\,[a]\,[a]\,p$$

In the case of logics with introspection axioms such as K4, K5 and its deontic and reflexive extensions we have also some small changes in the rules.

At first *for all logics extending* K4*, the rule* (4) *cannot be applied to prefixes of length* $\mid \sigma \mid= k_a$ where σ belongs to k_a-bounded agent. Intuitively, this happens because the (4) rule corresponds to the property of transitivity and we have restricted transitivity to interesting worlds in section 7.2.

In contrast with the classical case (§3.3), rule (K) is not always derivable from rules (4) and (T) for k-bounded agents because (4) fails at the border with incoherent or unknown world. Far away from not interesting worlds, the rule (K) is still derivable from those rules.

Second, *for the logics extending* K5*, rule* (4^R) *cannot be applied to prefixes of length* $\mid \sigma.b.n \mid= k_a$ where $\sigma.b.n$ belongs to k_a-bounded agent. Again this property fails because we have restricted of euclidity to interesting worlds.

In this case, to retain completeness we need to *add a rule for delayed reflexivity* in the following form (see also §3.3 for a definition):

Added Rule	Simulation
$\dfrac{\sigma.b.n : \nu(b)}{\sigma.b.n : \nu_0}\,T^D$	$\dfrac{\dfrac{\sigma.b.n : \nu(b)}{\sigma : \nu(b)}\,4^R}{\sigma.b.n : \nu_0}\,K$

The simulation with rules (4^R) and K fails at the border with not interesting worlds because rule (4^R) cannot be applied. Therefore we need to introduce directly the corresponding valid rule.

7.6 Approximating Eventualities in PDL

The extension to PDL is immediate for the modal operators. Here modality are programs and the intuition behind k-skeptical and k-credulous agents can be explained as their attitude towards the termination of programs.

Indeed recall that $\langle P \rangle A$ means, in the original intentions of Fischer and Ladner [60], that the program P terminates in a state where the property A holds.

Intuition. Bounding the introspection capabilities to k means that we expect every program longer than k-steps always crashes. It only makes a difference whether it crashes leaving us into an unknown state (credulous reasoning) or an incoherent state (skeptical reasoning).

We can also bound *iterated eventualities* without introducing any new machinery: we simply need to limit the maximum number of occurrences of the propositional letters X_i introduced by each $\langle P^* \rangle A$ in a branch.

By limiting the number of occurrences to k_i we are simply saying that we approximate P^* with $\bigcup_{n=1}^{k_i} P^n$. The choice of a skeptical or credulous approach simply tells in which kind of world leads the application of P^{k_i+1}. it can be respectively an incoherent or an unknown world.

Part III

An Application to Computer Security: Access Control

Chapter 8

Access Control and the DEC-SRC Calculus

In this chapter we single out an application of modal logics, namely the analysis of computer security and access control.

We introduce the topic of access control for the non-expert and present an overview of the various logics which have been used for reasoning and formal analysis in this field.

We choose a particular modal logic for the analysis of access control which have been introduced by Abadi, Lampson et. al [1] at the Digital System Research Center. We give the intuition and the semantics of the logic, clarify its computational features and gives a number of examples.

8.1 What is Access Control

Access control is a key issue for multi-agent distributed systems (see [153] for an introduction). Its main purpose is to restrain the actions which legitimate or malicious agents (humans, programs, agents, etc.) may perform, either directly or indirectly (via other agents). It should be clearly distinguished from other aspects of multi-agent distributed systems such as concurrency and coordination. These try to maximize the proper use of (public) resources while access control is focussed on the minimization of the improper use of (private or sensitive) resources.

Loosely speaking we may say that access control is the distributed systems counterpart of some aspects of human normative systems: who can do what. A human logging on a workstation, or a web-spider asking for permission to visit a restricted Internet site can be easily compared with a human crossing the passport control in an airport. In all cases, an access control decision is made. Indeed access control is just a problem of jurisdiction and security policy in distributed systems where many agents may have different goals and are able to make autonomous decisions.

Different access control policies may be devised to answer the needs of different systems: the well known Bell La Padula model deals with the military [12], banks and commercial organizations have been studied by Clarks & Wilson [28], and the access control requirements of health care services are discussed by Anderson [4]. A comprehensive survey of policies employed and needed in different organizations in the U.S. has been made by Ferraiolo et al. [59].

Distributed systems face additional challenges (e.g. large scale, insecure communications, delegation of management etc.) which make the access control decision more complex. For instance, access control has been combined with authentication by Wobber et al. [180], and policies have been refined on-line at the various levels of delegation by Moffet & Sloman [129].

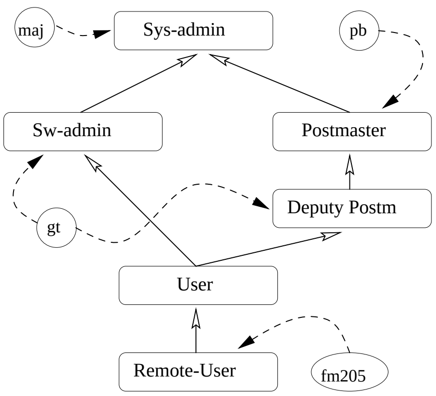
Moreover, the agents (principals in the security terminology) responsible for the decision making are likely to be sophisticate software agents, which must be able to take autonomous decisions in a sensible way.

It is also important to notice that the relations between agents in a real distributed system are seldom flat (all agents being equal). On the contrary, agents may be members of different groups or play different roles as shown by Ferraiolo et al. [59, 58]; the privileges of an agent usually depend on the groups they are members of; groups and roles are organized along hierarchies. Access control systems and policies becomes more complex [125, 74, 154].

A very simple example is shown in Fig. 8.1. The right hand side is the text from an information leaflet distribute to the users of the Computer Laboratory at the university of Cambridge (UK). It describes the roles, privileges and tasks of various principals (in this case humans but they need not be). On the left side we show a pictorial representation of the same structure. We represent roles as ovals and the hierarchies of privileges by straight line arrows. Principals having those roles are represented as circles and the "has role" relation is represented by a dotted arrow. We have described only the major, traditional roles of every distributed information system and we can notice how we have already to take into account hierarchies, membership relations and other issues such as delegation (*maj* may delegate *pb* when he is away).

As systems become more complex, human (and informal) verification becomes infeasible. Hence formal methods, logics and automated reasoning techniques can be useful tools for the verification of security policies and access control procedures.

Our target is to apply the deductive techniques developed in this dissertation as an automated reasoning tool for access control in multi-agent distributed systems.




```
maj:  Titan Room Manager, knows
      about UNIX, DEC stations,
      Alphas, Ethernet, ATM, printer
      systems, backup systems and TEX.
gt:  Knows about UNIX, Suns, HP
     bobcats and snakes, GNU emacs,
     X, Ethernet, Lisp and similar
     languages.  Deputy Postmaster.
pb:  Postmaster, knows about UNIX,
     Suns, X, mail, news, and wide
     area communications.

[...]  If you anticipate a need to
login from outside Cambridge, you
should consult maj or pb.
```

Figure 8.1: From "Computing Facilities at the Computer Laboratory"

8.1.1 Principles, techniques and requirements

The principles of access control in distributed systems can be described with few abstractions: *subjects* (humans, programs etc.), *objects* (data, other programs etc.) and *privileges* which subjects detain on objects (e.g. read, write and execute in `UNIX`). The use of these abstractions is the basis of most formal models proposed in the literature, starting from the traditional access matrix by Lamport [102, 152] to more advanced systems [58, 101, 154].

Using an access matrix is relatively simple: we put subjects along the rows, objects along the columns and privileges in the cells. Then checking what a subject can do is just a table look-up[1].

The problem becomes more complicated if we admit operations over such a matrix and are interested in *transition analysis*. We study the evolution of the system and its safety properties: given some privileges and some operations to change them, prove that the system will not evolve in an undesirable state. To this extent see the work of Harrison et al. [85] or Sandhu [152].

In the verification of multi-agent distributed systems we are often interested in *policy analysis*: do the actual low level access control procedures (and the decisions of an agent based on them) respect the overall policies of the organization?

With an access matrix this is immediate: all privileges of every agents are explicitly written down. We can just check them one by one.

The problem is that security policies are often not represented (and not

[1]This is technically complicated by the fact that we represent the matrix in compact form by lists along rows (capabilities) or along columns (access control lists) [153].

understood) at this level of details. For instance, the extensive field study of Ferraiolo et al. [59] pointed out that permissions are assigned to a user according his/her roles in the organization. How do we know that the low level privileges assigned to an agent are really the logical consequence of the structured policy based on roles and groups used and understood in the organization? The access matrix is extremely poor and it does not capture explicitly the richness of the underlying security policies. A key feature of the new approaches [58, 74, 154] is the attempt to model more closely the (hierarchical) relationships between the various subjects, such as those described in Fig. 8.1.

We can revise the abstractions in use and consider *agents* (or principals) and *operations* over objects as primitives as done by Abadi et al. [1], Ferraiolo et al. [58], Sandhu et al. [154] and use a logic able of expressing the following properties:

- *complex agents* which could be users, proper agents but also groups, roles and also agents using roles[2] or acting as delegates of other agents;
- *membership* relations between agents and groups or roles; agents can belong to different groups and to play different roles;
- *hierarchies* between groups and roles, where permissions are inherited (the more powerful has at least all privileges of the less powerful);
- *delegation certificates* so that agents may pass their privileges to other agents on-the-fly;
- *privileges attributions* which link agents with the operation that they can accomplish.
- *imperative statements*, representing the requests coming from other agents or groups, and which must be considered by the agent making the access control decision.
- *indirect requests* by an agent on behalf of somebody else (for instance A must be able to say that B made a request).

8.2 Logics for Reasoning About Access Control

The use of these abstractions leads naturally towards a formalization of the problem with multi modal logics (alternative approaches can be found in [85, 152, 129, 74]). The requests, actions and obligations of agents are mapped

[2]For instance $Alice$ and $Alice\ \mathsf{as}\ professor$ are not the same agent for the principal $Bob\ \mathsf{as}\ student$.

into modalities, atomic imperative or objective statements into propositions; other modal operators represent hierarchies, knowledge, permission etc.

The use of modal logics for modeling obligations, jurisdiction of agents has a long tradition dating back to Kanger [93] and new formalizations and applications can be found in the works of Cuppens & Demolombe [33] Glasgow et al. [75], and van der Meyden [173]. Traditional logics of knowledge and belief have been extended to deal with security problems by Burrows et al. [20], Glosgow et al. [75], and [169] and to multi-agent systems with beliefs, desires and intentions [71].

8.2.1 Logics for ability vs logics for obligations

In systems for reasoning about obligations and knowledge, atomic propositions typically represent state of affairs. In this framework our main concern is confidentiality, as for Cuppens & Demolombe [33] or Glasgow et al. [75] and we are looking for properties like

$$\text{“}P \text{ knows } A\text{”} \;\supset\; \text{“}P \text{ is allowed to know } A\text{”} \tag{8.1}$$

For a theory of ability, focusing on actions and obligations as advocated by Kanger [93] or van der Meyden [173], we look for other properties:

$$\text{“}P \text{ sees to it that } A\text{”} \;\supset\; \text{“}A \text{ is permitted”} \tag{8.2}$$

If atomic propositions are imperative statements [1] we may look for properties like the following one:

$$\text{“}P \text{ says } A\text{”} \wedge \text{“}P \text{ controls } A\text{”} \;\supset\; A \tag{8.3}$$

In logics for authentication such as the BAN logic [20, 169], a formula of this form (and the corresponding high-level property) is usually called the *jurisdiction* of a principal. Loosely speaking we may say that if P controls A (or equally P has privileges or jurisdiction over A) then P can (is allowed to) make A happen.

Still, if A is an imperative sentence, we may wonder what it means to say that A is true. The intuition is that A is a request directed to the agent in charge of the access control decision. With this provision, we can interpret it as propositional letters which can be true in a particular state (granted by the agent) or false (not granted).

To visualize the intuition behind the formulae we may compare a request such as $fm205\,\mathtt{says}\,rm(test.c)$ to a command typed by the user on a terminal:

`fm205{ely}: rm test.c`. As we press enter, it is evaluated according our privileges and either executed or not (i.e. $rm(test.c)$ may be true or false).

From this perspective, the modal operator $P\,\texttt{says}\,A$ is closer to the Do operator of Kanger [93] or "P sees to it that A". If A is not a sentence describing a state of affairs but rather an imperative statement we need to revise such definition. The key observation, from the viewpoint of access control, is that we should rather use an operator for "P tries A", since P may not be allowed to do A.

Anyhow, most logics for obligations and multiagent systems are also flat: agents are usually on equal footing. This may be acceptable for modelling laws (in front of which we are supposed to be equal), but it may be less so for an access control policy.

Hence our aim is to replace the reasoning based on schema (8.3) by the following one (more complex yet more realistic):

$$\text{“}P \text{ says } A\text{”} \wedge \text{“}Q \text{ controls } A\text{”} \wedge \text{ “}P \text{ is more powerful than } Q\text{”} \supset A \qquad (8.4)$$

where the relation between P and Q is specified by an access control policy.

8.2.2 The Reasoning Services

Once we have formalized the information needed by an agent for an access control decision (e.g. group hierarchies, privileges, etc.) we also want to do something with it. We expect a number of reasoning services:

Consistency Check: check whether the assumptions have a model. Typically verify the compatibility of role hierarchies, users attributions etc. with the constraints.

Logical Consequence: decide whether a property A logically follows from the assumptions. It could be used to decide whether a request should be granted or whether some delegation certificate is entailed even if is not explicitly written.

Model Generation: if a property A does not follows from the assumptions, derive a counter model for it.

Compatibility: liberal systems may decide to grant a request provided that it is not inconsistent with the policy (even if is not logical consequence of our policy).

From this perspective tableau methods are extremely suitable since they provide a deductive machinery where all these services are available.

On the contrary, axiom systems, which are often used to define formal models based on logics [1, 33, 75, 93, 173], are inadequate: axioms systems require people to be good logicians and cannot be automated at all. The exclusive focus on axiomatization, without calculi for automated reasoning, may condemn logics for practical reasoning to be unusable.

8.2.3 The choice of the DEC-SRC logic

We focus on the expressive calculus developed at Digital-SRC by Abadi, Lampson et al. [1] for a number of reasons:

- it provides a uniform and expressive framework for reasoning about access control in presence of delegation and hierarchies between agents;
- it has a simple and natural semantics based on Kripke models;
- it constitutes the basis of a real system [101, 180] whose protocols have been checked (by hand) using the calculus;
- its features pose interesting technical challenges for deduction.

One of the characteristics which challenge standard modal tableau calculi is the presence of formulae used for modeling delegation certificates and hierarchical relationships between subjects (i.e. modalities). Those relations have the same force of axiom schemas and are close to role-value-map constructs of AI languages analyzed by Schild [158].

The key difficulty is that we cannot compile these axioms into tableau rules (or axiom schemas) since their presence depends on the particular non logical axioms *and* the particular theorem we want to prove. Different axioms correspond to different security policies and we do not want to hardwire the policy in the logic. Moreover, access privileges may be passed on-the-fly by the agents. Thus a formula may contain a delegation of certain privileges from A to B (and thus be a valid request) another may not (and thus be invalid). Again we don't want to fix the possible delegations in the calculus.

Another characteristic is the absence of the tree model property which also hinders a straightforward extension of tableau calculi. Finally it shares some properties of the universal modality [76, 77] which is not axiomatizable[3].

In a nutshell, some global properties of the underlying Kripke models can be discovered only *on-line* during the deduction process.

We describe these semantical properties in the remaining part of the chapter, while the extension of prefixed tableaux to a sound and complete deduction method for a large fragment of the calculus and a correct one for the whole language (the calculus is undecidable in general) is delayed until chapter 9.

[3] This may also explain why only a sound axiomatization has been devised in [1].

8.3 The DEC-SRC Calculus

To make the paper self-contained we present the syntax and the semantics of the calculus and refer to Abadi et al. [1] for further discussions and to Lampson et al. [101, 180] for its applications. We only clarify some of its semantical features which are not discussed in [1]. We modify slightly the notation so that it is consistent with the notation used throughout the dissertation.

8.3.1 Notation and Intuitions

The two building blocks of the calculus are operations and agents.

Operations over objects are represented by atomic imperative *statements* denoted by r (for request). Once again, they are propositional letters which can be true in a particular state of the system (request granted) or false (not granted). Complex statements are built with boolean connectives $\wedge, \neg, \supset$ etc.: for instance $\mathit{login}(\mathit{telnet}) \supset \mathit{login}(\mathit{ftp})$ whose intuitive interpretation is "if telnet access has been granted so has ftp access".

The names of agents, groups and cryptographic keys are represented by atomic *principals*, and denoted by a, b, K_a etc. Complex principals (P or Q) are built by conjunction "$\&$" and quoting "$|$". The intuition is that $P \,\&\, Q$ is a principal with the privileges of both P and Q, whereas $P \,|\, Q$ corresponds to the principal P claiming to quote a request from Q. Notice that P may claim to quote Q even when Q never said anything.

An example may better clarify the quoting construct. Suppose we received an email from $fm205$ asking for a request r. We often react as r had come directly from $fm205$ but, to be precise, the request came from the mail agent *sendmail* which claimed to have received a mail from $fm205$ i.e. the mail is coming from the "composite" agent $\mathit{sendmail} \,|\, fm205$. We usually trust *sendmail* to forward all requests by $fm205$ and not to forge fake requests but we cannot (nor we should) automatically rely on this assumption in every cas.

Other operators are possible i.e. $P \,\mathtt{for}\, Q$ and $P \,\mathtt{as}\, R$ [1]: the former is used when P is claiming to act as a delegate for Q; the latter when Q speaks using a role[4] R. They can be encoded using "$|$" and "$\&$" (see [1] for a discussion), so we do not use them. For instance $P \;\mathtt{for}\; Q \doteq (P \,\&\, d) \,|\, Q$ where d is an agent entrusted with delegation certificates. The incorporation of roles may be trickier since roles may have particular properties. We discuss their incorporation into this framework in section 8.5.

To represent *agent requests* we use the modal statement "$P \,\mathtt{says}\, s$": principal P requests s to be granted. If P is a group then we follow Abadi et al. [1, 101] and interpret it as "somebody in group P says s". When P is a cryptographic key we mean that the request has been signed or encrypted with

[4] For a distinction between the security concepts of role and group see [58, 154].

the corresponding key. So, following a common interpretation in computer security, we may identify a principal with a key[5].

Hierarchical relations between principals are constructed with the "speaks for" operator $P \Rightarrow Q$. The intuition is that P has at least all privileges of Q i.e. P can speak for Q. If P says s this would be as Q itself said s. Back to the mail agent example, we often assume that $\mathit{sendmail} \mid fm205 \Rightarrow fm205$.

The operator $\Rightarrow$ is also used for group membership: $P \Rightarrow G$ means that P has at least all privileges of group G. We also speak of a "certificate" when we have the formula $K \Rightarrow P$. Intuitively this formula binds K to P, so that all messages signed with K can be seen as coming from P.

Principals and statements are linked by *privileges attributions* [154]: the statement "$P\,\mathtt{controls}\,s$" captures the intuition that principal P has access control over s. This is the *jurisdiction* of a principal (see §8.2).

In the DEC-SRC calculus `controls` is a defined symbol:

$$P\,\mathtt{controls}\,s \;\equiv\; (P\,\mathtt{says}\,s) \supset s$$

The intuition may become clearer by rephrasing it as "P controls s means that if P asks for s to be executed then s will indeed be executed by the agent in charge of the access control decision".

In future extensions, it may be desirable to replace material implication with an intuitionistic or a relevant implication. At present, a principal may pretend to control whatever she does not explicitly request i.e. $\neg(P\,\mathtt{says}\,s) \supset P\,\mathtt{controls}\,s$. Such property may be a feature if we want to model 'bluffing' principals. Anyhow, this is sufficient for modelling a wide range of situations.

To show how the logic can be used, we consider the case of the access control structure of Fig. 8.1.

We use an assumption $P \Rightarrow Q$ for every direct subsumption relation between roles. So we have $\mathit{SysAdm} \Rightarrow \mathit{SwAdm}$ and $\mathit{SwAdm} \Rightarrow \mathit{Usr}$. The role membership of the various agents is also described with the $\Rightarrow$ operator, so we have $maj \Rightarrow \mathit{SysAdm}$ and $fm205 \Rightarrow \mathit{RemUsr}$. We do not need to specify in details all relations between agents and roles: the logic does the transitive closure of the speaks-for relation for us. For instance $pb \Rightarrow \mathit{Usr}$ is a logical consequence of our assumptions. We can verify it with the tableau procedure in §9.1.

We also introduce some privileges: every remote user can log in with ftp but only authorized users can use telnet. The privilege of reading one's mail is a possible example of the use of conjunction: the mail belongs to $fm205$ so he (or a delegate of him) is the only one who can read it, but he must log in as a legitimate user. Notice that the system administrator can change the

[5]Since we are interested in access control and not in authentication or confidentiality we assume that the problems of associating keys with principals or finding whether keys are good etc. have been tackled already (see [20, 101, 169]).

$$\text{agents} = \begin{cases} maj \Rightarrow SysAdm \\ gt \Rightarrow (SwAdm \,\&\, DepPostM) \\ pb \Rightarrow PostMaster \\ fm205 \Rightarrow RemUsr \end{cases} \qquad \text{priv.} = \begin{cases} SysAdm \;\texttt{controls}\; (fm205 \Rightarrow Usr) \\ Usr \;\texttt{controls}\; login(telnet) \\ RemUsr \;\texttt{controls}\; login(ftp) \\ (fm205 \,\&\, Usr) \;\texttt{controls}\; read(mail) \end{cases}$$

$$\text{groups} = \begin{cases} SysAdm \Rightarrow SwAdm \\ SysAdm \Rightarrow PostMaster \\ SwAdm \Rightarrow Usr \\ PostMaster \Rightarrow DepPostM \\ DepPostM \Rightarrow Usr \\ Usr \Rightarrow RemUsr \end{cases} \qquad \text{req.} = \begin{cases} maj \;\texttt{says}\; (fm205 \Rightarrow Usr) \\ fm205 \;\texttt{says}\; login(telnet) \\ fm205 \;\texttt{says}\; read(mail) \end{cases}$$

Figure 8.2: A Logical Formalization of Fig. 8.1

group membership of $fm205$. So, the possibility of logging from remote sites depends on maj's statements.

Some possible statements are shown with the formalization in Fig. 8.2.

8.3.2 Formal Syntax

The language is the following:

$$\begin{array}{rcl} P, Q & ::= & a \mid P \,\&\, Q \mid (P \mid Q) \\ s, s' & ::= & r \mid \neg s \mid s \wedge s' \mid P \;\texttt{says}\; s \mid (P \Rightarrow Q) \end{array}$$

where r is an atomic request (a propositional letter) and a is an atomic principal (and agent or a key). Other connectives are abbreviations, e.g. $s \supset s' \equiv \neg(s \wedge \neg s')$. We assume that in $P \Rightarrow Q$ either P or Q is an atomic principal, w.l.o.g. since $P \Rightarrow Q$ is equivalent to $P \Rightarrow a \wedge a \Rightarrow Q$ for a new atomic a.

For sake of readibility we do not introduce signed formulae but they can be easily introduced along the same lines of dynamic logic (§5.3). We also try to omit parentheses whenever possible. So, we make the following assumptions: $\mid$ and $\&$ bind more strongly than $\Rightarrow$; operator over principals have higher precedence than modal and propositional connectives; modal connectives such as `says` and `controls` have a higher precedence than propositional ones. We assume the usual precedence among propositional connectives.

So $\neg(A \mid B \;\texttt{says}\; s) \wedge A \mid B \Rightarrow B \supset B \mid A \;\texttt{says}\; s$ should be read as follows:

$$(\neg\, ((A \mid B) \;\texttt{says}\; s) \wedge ((A \mid B) \Rightarrow B)) \supset ((B \mid A) \;\texttt{says}\; s))$$

A statement is *left (right) restricted* when speaks-for subformulae have the form $a \Rightarrow Q$ (respectively $P \Rightarrow a$) i.e. the left (right) principal is atomic. It is *weakly left (right) restricted* if all speaks-for statements under the scope of an even number of negation are left (right) restricted. This means that arbitrary statements $P{\Rightarrow}Q$ are admitted only under the scope of an odd number of negations[6]. It is *request restricted* when in each statement of the form $P\,\texttt{says}\,s$, the statement s is either an atomic request or a group membership (both possibly negated). For instance the logical formalisation of the access control structure in Fig. 8.2 is left, right and request restricted.

In practice statements are right and request restricted. If $\Rightarrow$ is used for hierarchies and group and role membership, as in Fig. 8.1, the rightmost principal is atomic. Moreover, in almost all systems [153], privileges attributions are represented by ACL (Access Control Lists). In the DEC-SRC language, an ACL for a request r is simply the conjunction of statements $\bigwedge_i P_i\,\texttt{controls}\,r$, where r is uninterpreted [101, 154]. If we add, among the possible privileges, the possibility to hand over delegation to other principals such as $P_i\,\texttt{controls}\,(Q_j{\Rightarrow}a_k)$, then we still have a right restricted language. Some examples can be found in section 8.4.

8.3.3 Kripke Semantics

The semantics is based on Kripke models [62, 89]: a relation models the compatibility of a state with the requests made by a principal in another state. A model is a pair $\langle W, \mathcal{I}\rangle$, where W is a non empty set of states and $()^{\mathcal{I}}$ an interpretation such that for every atomic principal a one has $(a)^{\mathcal{I}} \subseteq W \times W$ and for every propositional letter r one has $(r)^{\mathcal{I}} \subseteq W$. The interpretation $()^{\mathcal{I}}$ is extended as follows:

$$\begin{array}{lcl}
(\neg s)^{\mathcal{I}} & = & W - (s)^{\mathcal{I}} \\
(s \wedge s')^{\mathcal{I}} & = & (s)^{\mathcal{I}} \cap (s')^{\mathcal{I}} \\
(P{\Rightarrow}Q)^{\mathcal{I}} & = & \text{if } (Q)^{\mathcal{I}} \subseteq (P)^{\mathcal{I}} \text{ then } W \text{ else } \emptyset \\
(P\,\texttt{says}\,s)^{\mathcal{I}} & = & \left\{w \mid \forall v \in W \text{ if } \langle w,v\rangle \in (P)^{\mathcal{I}} \text{ then } v \in (s)^{\mathcal{I}}\right\} \\
(P \,\&\, Q)^{\mathcal{I}} & = & (P)^{\mathcal{I}} \cup (Q)^{\mathcal{I}} \\
(P \mid Q)^{\mathcal{I}} & = & \left\{\langle w,v\rangle \mid \exists u\, \langle w,u\rangle \in (P)^{\mathcal{I}} \text{ and } \langle u,v\rangle \in (Q)^{\mathcal{I}}\right\}
\end{array}$$

As we have done for dynamic logic (§5.3) we write $w \Vdash s$ for $w \in (s)^{\mathcal{I}}$ and say that w satisfies s. We interchange a set of statements with their conjunction. A model $\langle W, \mathcal{I}\rangle$ satisfies a set of statements S iff every world in W satisfies S.

Definition 8.1 *A statement s is* valid *if every model $\langle W, \mathcal{I}\rangle$ satisfies s. It is* a logical consequence *of global assumptions G and local assumptions U iff in every model $\langle W, \mathcal{I}\rangle$ which satisfies G if a world w is such that $w \Vdash U$ then $w \Vdash s$.*

[6]For instance the formula $\neg(a\,\texttt{says}\,((b \,\&\, c){\Rightarrow}d))$ is weakly left restricted since the group membership $(b \,\&\, c){\Rightarrow}d$ is under the scope of one negation.

In this framework the global assumptions are non-logical axioms describing the access control system: groups membership, privileges attributions etc.

Global axioms can be added in the axiomatization of [1] with the modal deduction theorem [89, 62], but their explicit representation is more effective because the modal deduction theorem leads to an exponential blow up (§4.5).

8.3.4 Semantical Properties

The semantics of $P \Rightarrow Q$ reflects global properties of the model, is close to the universal modality [76, 77], and can introduce axiom schemas on the fly.

We can represent explicitly the relation with the universal modality *univ*:

$$P \Rightarrow Q \equiv univ \, \texttt{says} \, (P \Rightarrow_{loc} Q)$$

where we have the following two conditions:

$$\begin{array}{rcl} (P \Rightarrow_{loc} Q)^{\mathcal{I}} & = & \left\{ w \mid \forall v \in W \text{ if } \langle w, v \rangle \in (Q)^{\mathcal{I}} \text{ then } \langle w, v \rangle \in (P)^{\mathcal{I}} \right\} \\ (univ)^{\mathcal{I}} & = & W \times W \end{array}$$

A key property is the possibility of *introducing axiom schemas "on the fly"*. For instance $P \Rightarrow P \mid P$ forces the transitivity of relation $(P)^{\mathcal{I}}$, where P may be a complex principal. Yet, these global properties may or may not be present. As an example, suppose we have $A \, \texttt{says} \, (B \Rightarrow B \mid B)$. Transitivity of B will follow only if $\neg A \, \texttt{says} \, \bot$ is the case. So B's properties depend on the particular global axioms and theorems we are trying to prove.

Another feature is the *absence of the tree-model property* characterizing standard modal logics [172, 89]. For instance the following formula has no tree model at all:

$$\begin{array}{ll} a \Rightarrow a \mid a \, \wedge & \% \;\; a \text{ is transitive} \\ \neg(a \Rightarrow b) \, \wedge & \% \;\; b \text{ is not included in } a \\ b \, \texttt{says} \, \bot \wedge \neg(a \, \texttt{says} \, \bot) \, \wedge & \% \;\; b \text{ is inconsistent and } a \text{ is not} \\ a \, \texttt{says} \, (b \, \texttt{says} \, \bot \wedge \neg a \, \texttt{says} \, \bot) & \% \;\; \text{and so on (at least for } a) \end{array}$$

Yet, it is satisfiable in the world 1 of the model with $W = \{1, 2, 1.a.3\}$ and $(a)^{\mathcal{I}} = \{\langle 1, 1.a.3 \rangle, \langle 1.a.3, 1.a.3 \rangle\}$ and $(b)^{\mathcal{I}} = \{\langle 2, 2 \rangle\}$. The key point is that this model has *two* clusters (connected components) so that 1 satisfies the (local) `says` statements and 2 satisfies the global $\neg(a \Rightarrow b)$.

In model theoretic terms, (un)satisfiability is not preserved under disjoint union as in traditional modal logics (see van Benthem [172]). This is due to the hidden presence of the universal modality which makes impossible the complete characterization of the logic with an Hilbert system [76, 77, 172].

8.3.5 Undecidability

Validity is *undecidable* in general and a reduction to Thue systems (without details) is pointed out by Abadi et al. [1]. A simpler proof uses the techniques of Schild [158] and reduces validity to the world problem of (semi)groups.

We map each element of the group a_g to an atomic principal a and composition "$\circ$" to quoting "$\,|\,$". Equations between words $p \cong q$ becomes conjunctions of speaks-for statements $P \Rightarrow Q \wedge Q \Rightarrow P$ (for short $P = Q$) for the corresponding P and Q.

To define groups we must add, for each atomic principal a, a new principal a^c corresponding to its converse. Then we add the equation $a \,|\, a^c = e$ where e is the atomic principal corresponding to the identity element e_g.

Finally we use a new principal g and the global axioms $g \,|\, a = g$ for every atomic a and the statement $g\,\texttt{says}\,(P = Q)$ for every $p \cong q$ characterizing the group. Then one can prove that $g\,\texttt{says}\,(P' = Q')$ is valid with those assumptions iff equation $p' \cong q'$ holds for the group [158].

8.4 Some Examples

In this section we use the logic to formalize the information of agents who have to make the access control decision for a number of problems.

We have already shown the formalization of the access control structure of Fig. 8.1 in Fig. 8.2. Then we can consider the following problem:

Example 1 A software agent works as a firewall in a Computer Laboratory whose policy is formalized in Fig. 8.2. What should the agent do when maj grants to $fm205$ the membership in the role of $User$ and $fm205$ asks to login with telnet and tries to read his mail, should the agent grant $login(telnet) \wedge read(mail)$?

If we abbreviate $req \doteq login(telnet) \wedge read(mail)$, then the statement we would like to check is the following:

$$maj\,\texttt{says}\,(fm205 \Rightarrow User) \wedge fm205\,\texttt{says}\,req \supset req$$

The global assumptions in use are denoted by *agents*, *groups* and *priv.* in Fig. 8.2. If that statement is a logical consequence of our assumptions then the agent in charge of the firewall could "safely" grant the request of $fm205$.

The next example is the following:

Example 2 The agent *Paymaster* is responsible for the automatic handling of invoices and payments in the department. *Alice* is responsible for the

$$\text{agents} = \begin{cases} Alice \Rightarrow Account \\ Bob \Rightarrow Marketing \\ Charlie \Rightarrow Store \\ Sam \Rightarrow Server \end{cases} \qquad \text{priv.} = \begin{cases} Marketing\ \texttt{controls}\ order \\ Account\ \texttt{controls}\ pay \\ Store\ \texttt{controls}\ recvd \\ Alice\ \texttt{controls}\ (Bob\ \texttt{says}\ order) \\ Server\ \texttt{controls}\ (K_A \Rightarrow Alice) \\ Server\ \texttt{controls}\ (K_B \Rightarrow Bob) \end{cases}$$

$$\text{msg} = \begin{cases} K_A\ \texttt{says}\ Bob\ \texttt{says}\ order \\ K_C\ \texttt{says}\ recvd \\ K_S\ \texttt{says}\ K_A \Rightarrow Alice \\ K_S\ \texttt{says}\ K_C \Rightarrow Charlie \end{cases} \qquad \begin{array}{l} \text{cert.} = \{\ K_S \Rightarrow Sam \\ \\ \text{constr.} = \{\ Account\ \texttt{says}\ (order \wedge recvd \supset pay) \end{array}$$

Figure 8.3: A Model with Keys for Example 2

accounts, *Bob* for marketing and ordering of goods, and *Charlie* is the store-keeper in charge of receiving and shipping of goods. *Alice* is trusted in forwarding *Bob* orders. *Sam* is the security agent responsible for the authentication server. The policy of the department is to pay after the store has certified the arrival of goods. Messages and certificates are shown in Fig. 8.3. Should the agent *Paymaster* actually *pay*?

The corresponding formalization is shown in Fig. 8.3. We used the same technique used for Fig. 8.2 in §8.3.1: group memberships are represented by statements of the form $agent \Rightarrow department$ for every agent and every department; key certificates have the form $key \Rightarrow agent$; signed messages are denoted by $key\ \texttt{says}\ message$. The set of privileges is immediate.

Another example concerns the access control decision in presence of the standard hierarchy of "clearance" with read-down and write-up policies (see Sandhu & Samarati [153] for a discussion). For instance, with a read-down security policy an agent can only read documents with a clearance lower than his own. Although it seems a problem restricted to the military, its relevance spans far beyond this field. Video on demand is a good example.

Example 3 Consider the hierarchy established by a corporation for video on demand: F for films suitable for the whole family, PG for those requiring parental guidance, 15 for those limited to teenager at least 15 years old, and A for adults only. The users are *Sam* with A privileges, *Bob* with 15 and *Charlie* who has none. The agent *Alice*, called "Cyber-sitter", supervises the requests from the users which are entitled to different movies and tries to avoid that minors have access to unsuitable movies. She received a message and some delegation certificates from the *Decoder* (Fig. 8.4). Should $load(m_1) \wedge view(m_1)$ be granted with the equivalent of a read-down policy for viewing?

In other terms is $load(m_1) \wedge view(m_1)$ a logical consequence of the access control model described in Fig. 8.4? Again, recall that we represent the infor-

$$\text{groups} = \begin{cases} A \Rightarrow 15 \\ 15 \Rightarrow PG \\ PG \Rightarrow F \end{cases} \qquad \text{priv.} = \begin{cases} A\ \texttt{controls}\ view(m_1) \\ A\ \texttt{controls}\ load(m_1) \\ 15\ \texttt{controls}\ load(m_1) \\ 15\ \texttt{controls}\ view(m_2) \\ F\ \texttt{controls}\ stop(m_1) \\ Decoder\ \texttt{controls}\ (K_S \Rightarrow Sam) \\ Decoder\ \texttt{controls}\ (K_B \Rightarrow Bob) \\ Decoder\ \texttt{controls}\ (K_C \Rightarrow Charlie) \end{cases}$$

$$\text{agents} = \begin{cases} Sam \Rightarrow A \\ Bob \Rightarrow 15 \\ Charlie \Rightarrow F \end{cases}$$

$$\text{cert.} = \begin{cases} K_D \Rightarrow Decoder \\ K_S \Rightarrow Sam \end{cases} \qquad \text{msg} = \begin{cases} K_D\ \texttt{says}\ K_B \Rightarrow Bob \\ K_B\ \texttt{says}\ Sam\ \texttt{says}\ (load(m_1) \wedge view(m_1)) \end{cases}$$

Figure 8.4: A Read-down Policy for "Cyber-sitter"

mation from the viewpoint of *Alice*, the agent in charge of the access control decision. Even in this simple case is not easy to see that it is *not*. Hence the need of an automated reasoning method.

For a "real-life" example we take delegation without certificates from Abadi et al. [1, page 719] where a careful (and non trivial) Hilbert proof is given. The problem that an agent C has to face is the following:

Example 4 "A delegates to B who makes requests to C. For instance A may be a user with a sufficiently powerful smart-card, B a workstation and C a file server. [...] When B wishes to make a request r on A's behalf, B sends the signed requests along with A's name... in the format $K_B\ \texttt{says}\ (A\ \texttt{says}\ r)$...
When C receives the request r he has evidence that B has said A has requested r but not that A has delegated to B; then C consults the ACL for request r and determines whether the request should be granted. [...] A certification authority provides the certificates for the principals' public keys as needed. The necessary certificates are $K_S\ \texttt{says}\ (K_A \Rightarrow A)$ and $K_S\ \texttt{says}\ (K_B \Rightarrow B)$, where K_S is S's public key."

We are asking whether $G \models s \supset r$ where G and s are defined as follows:

$$\begin{aligned} G &\doteq \{ K_S \Rightarrow S, S\ \texttt{controls}\ (K_B \Rightarrow B), Sc \Rightarrow A \} \cup ACL \\ s &\doteq K_B\ \texttt{says}\ (Sc\ \texttt{says}\ r) \wedge K_S\ \texttt{says}\ (K_B \Rightarrow B) \end{aligned}$$

Where ACL is a set of statements of the form $P\ \texttt{controls}\ s$. For instance we may assume that $(B \mid A)\ \texttt{controls}\ r \in ACL$.

8.5 Incorporating Roles

New security paradigms have stressed the importance of an explicit treatment of roles in the design and verification of access control systems and policies.

Roles simplify the design of the system and make possible to use engineering principles such as least privilege, separation of duties etc. Comprehensive surveys can be found in the works of Ferraiolo et al. [58] at the US Department of Standards, or Sandhu et al. [154].

Roles are typically assumed by principals to accomplish some particular duties. A simple example is the command `su` (super-user) in UNIX systems. By executing `pb@ely% su webmaster` the user *pb* assume the role of *webmaster* to accomplish some tasks (such as changing the department home page) which require a particular set of privileges. As mentioned already, the distinction between roles and groups can be tricky and we refer to the literature for further discussion [58, 154]. An intuitive understanding of the concept of role and of the difference between the principals P and $P \,\mathtt{as}\, R$ is sufficient here.

A role is typically used by a principal in the format $P \,\mathtt{as}\, R$, where principal P claims to be using role R, e.g. $pb \,\mathtt{as}\, webmaster$. This can be encoded by modelling role assumption " $\mathtt{as}$ " with quoting " $|$ " as suggested in section 8.3.1.

Yet, in the formalism and deduction method presented so far, roles are just "principals like any other". After the encoding, the design of properties and global assumptions characterizing them is up to the user of our method.

It is possible to deal with them explicitly, along the lines proposed by Abadi et al. [1] for an axiomatic approach or by Massacci [118] for a tableau-based deduction. To this extent we must introduce distinguished symbols for atomic roles $R_1 \dots R_n$ and define their properties by adding suitable axiom schemata or tableau rules.

The first step is establishing the relationships between normal agents and roles. A possible consideration is the following: if *Bob* does not restrict his privileges when making a request to *Alice*, then *Alice* should conclude that *Bob* wants to use all possible roles entrusted to him. This avoids a denial of service from *Alice* and is characterized by the axiom schema below:

$$P \Rightarrow P \,\mathtt{as}\, R$$

We can also require *idempotency* and enforce the following axioms:

$$R \,\mathtt{as}\, R \Rightarrow R \qquad\qquad R \Rightarrow R \,\mathtt{as}\, R$$

There seems to be no special reason why $fm205 \,\mathtt{as}\, user$ should be different from $fm205 \,\mathtt{as}\, user \,\mathtt{as}\, user$.

Other properties, adopted in [1], may be less intuitive and appealing. For instance consider commutativity of roles, represented by the axioms below:

$$R_1 \,\mathtt{as}\, R_2 \Rightarrow R_2 \,\mathtt{as}\, R_1 \qquad\qquad R_2 \,\mathtt{as}\, R_1 \Rightarrow R_1 \,\mathtt{as}\, R_2$$

Commutativity may lead to undesirable results when combined with the need of avoiding denial of service. Indeed we can expect to have $P \,\mathtt{as}\, R_1 \,\mathtt{as}\, R_2$ to have the same privileges of $P \,\mathtt{as}\, R_2$ and hence by commutativity of $P \,\mathtt{as}\, R_1$. So we may lose the possibility of "downgrading" our privileges which may be advisable when performing sensitive operation. For example, suppose that $fm205$ assumed the role $root$ in the course of some operations. Later on he may wish to pass to $fm205 \,\mathtt{as}\, root \,\mathtt{as}\, user$, somehow downgrading his privileges for some particular task. With the above mentioned axioms he will find itself with the full privileges of $root$ rather than $user$ as he expected. Moreover, if R_1 and R_2 are incompatible roles we may also derive an inconsistency.

The direct embedding preserves the flexibility of the calculus but may generate many irrelevant properties and therefore may hinder the effectiveness of the deduction method.

Another possibility is to design deduction rules for roles, which take into account their specific properties. The trade off is that we may hardwire desirable and less desirable security properties into the deduction methods.

A combination of both methods may be worth investigating.

Chapter 9

Tableaux for the DEC-SRC Calculus

Finally we apply our general methodology for the design of efficient and approximate reasoning in propositional, modal and dynamic logic to the DEC-SRC logic.

We describe the basic tableau calculus and give some examples of deduction and prove its soundness and completeness. The general deductive machinery from the previous chapters can now be easily extended and we can concentrate on the peculiar (and difficult) characteristics of the logic.

9.1 The Tableau Calculus

We capitalize on the deductive machinery already defined for modal logics (§.3) and dynamic logic (§5). This shows the flexibility of the approach via prefixed tableaux.

Prefixed tableaux for the DEC-calculus use *prefixed statements*, i.e.pairs $\sigma : s$ where s is a statement and σ is an alternating sequence of integers n and atomic principals a called prefix and defined as $\sigma ::= n \mid \sigma.a.n$.

Remark. A key difference from standard prefixed tableaux is that a prefix does not always start with 1. So a set of prefixes is a *forest of trees*, where arcs are labelled with atomic principals and nodes with integers.

With k different initial prefixes we have, in graph-theoretic terminology, k connected components or clusters [89]. With global axioms and the operator $\Rightarrow$ we can impose an euclidean or transitive closure on a cluster but we cannot collapse two clusters.

The definition of tableau is identical to dynamic logic: a *tableau* $\mathcal{T}$ is a rooted (binary) tree where nodes are labelled with prefixed statements in the usual fashion. What changes is the notion of branch:

$$\langle\,|\,\rangle\ \frac{\sigma : \neg(P\,|\,Q\ \mathtt{says}\ s)}{\sigma : \neg P\ \mathtt{says}\ (Q\ \mathtt{says}\ s)} \qquad [\,|\,]\ \frac{\sigma : P\,|\,Q\ \mathtt{says}\ s}{\sigma : P\ \mathtt{says}\ (Q\ \mathtt{says}\ s)}$$

$$\langle\,\&\,\rangle\ \frac{\sigma : \neg(P\,\&\,Q\ \mathtt{says}\ s)}{\sigma : \neg P\ \mathtt{says}\ s \qquad \sigma : \neg Q\ \mathtt{says}\ s} \qquad [\,\&\,]\ \frac{\sigma : P\,\&\,Q\ \mathtt{says}\ s}{\begin{array}{c}\sigma : P\ \mathtt{says}\ s\\ \sigma : Q\ \mathtt{says}\ s\end{array}}$$

$$\mathrm{Glob}\ \frac{\vdots}{\sigma : s}\quad \text{if } \sigma \text{ is present in } \mathcal{B} \text{ and } s \in G_{\mathcal{B}}$$

$$\langle a\rangle \frac{\sigma : \neg(a\ \mathtt{says}\ s)}{\sigma.a.m : \neg s}\ \sigma.a.m \text{ new} \qquad [a]\ \frac{\sigma : a\ \mathtt{says}\ s}{\sigma.a.n : s}\ \sigma.a.n \text{ present}$$

$$D(a)\ \frac{\sigma : a\ \mathtt{says}\ s}{\sigma : \neg a\ \mathtt{says}\ \neg s}\ \text{ with some } \sigma.a.n \text{ present in the branch}$$

Figure 9.1: PDL-like Rules for Modal Connectives

Definition 9.1 *A* branch *of a tableau $\mathcal{T}$ is a pair $\langle \mathsf{B}, G_{\mathcal{B}}\rangle$ where $\mathcal{B}$ is a path from the root to a leaf of $\mathcal{T}$ and $G_{\mathcal{B}}$ is a set of global axioms.*

Each time we split a branch of the tree we should duplicate (in theory) the set of global axioms. This definition is essential because we need to modify the set of global axioms during the deduction process and therefore different branches may end up with different global axioms.

The notion of prefixes present or new in a branch is standard (§3, §5).

9.1.1 Tableau rules

The rules for propositional connectives are identical to those for modal and dynamic logics (§3.3 and §5.4) and are omitted. The rules for conjunction, quoting, the use of global axioms and the transitional rules for atomic principals are in Fig. 9.1.

It is immediate to notice the similarity of the $|$ and $\&$ operators with sequential and nondeterministic composition in PDL (see again §5.4).

To cope with $\Rightarrow$ we introduce a new set of propositional atoms x_i (distinct from r) to mark unsaid statements as in Fig. 9.2. The meaning of $P{\Rightarrow}Q$ is that for all s, if $P\ \mathtt{says}\ s$ is true then $Q\ \mathtt{says}\ s$ is also true. its negation means that there is "something" (an unknown x_i) which P said but Q didn't. The first two rules correspond to the local features of the $\Rightarrow$ operator, whereas the last is due to its "universal" flavor. The $\langle U_{gr}\rangle$-rule combines both aspects.

$$\langle R_{gr}\rangle : \frac{\sigma : P \Rightarrow A \quad \sigma : \neg(A \,\mathtt{says}\, s)}{\sigma : \neg(P \,\mathtt{says}\, s)} \qquad [L_{gr}] : \frac{\sigma : A \,\mathtt{says}\, s \quad \sigma : A \Rightarrow Q}{\sigma : Q \,\mathtt{says}\, s}$$

$$\langle U_{gr}\rangle : \frac{\sigma : \neg(P \Rightarrow Q) \quad x_i \text{ and } n \text{ new}}{\begin{array}{c} n : P \,\mathtt{says}\, x_i \\ n : \neg(Q \,\mathtt{says}\, x_i) \end{array}} \qquad [U_{gr}] : \frac{\sigma : P \Rightarrow Q}{G_{\mathcal{B}} := G_{\mathcal{B}} \cup \{P \Rightarrow Q\}}$$

Figure 9.2: Rules for the speaks-for operator

Remark. Weakly left restricted statements do *not* need rule $\langle R_{gr}\rangle$ and $D(a)$ while right restricted languages do not need rule $[L_{gr}]$.

Indeed the completeness proof (Thm. 9.9) for weakly left restricted fragments does not make use of the rules $\langle R_{gr}\rangle$ and $D(a)$.

Further simplifications are possible: $P \Rightarrow (A \,\&\, B)$ is logically equivalent to $P \Rightarrow A \wedge P \Rightarrow B$. This rule can be added since it may lead to right-restricted formulae. Moreover we can apply the following technique:

Technique 9.1 *Rule $\langle U_{gr}\rangle$ must be applied only once for each subformula $\neg(P \Rightarrow Q)$, no matter its prefix.*

We can introduce the concept of reduced formula and contradictory branch as in modal and dynamic logic and apply all corresponding techniques. For instance rule $\langle a\rangle$ can be skipped if a prefixed formula $\sigma.A.n : \neg s$ is already present for some n etc.

Definition 9.2 *A branch $\langle \mathcal{B}, G_{\mathcal{B}}\rangle$ is* contradictory *if $\mathcal{B}$ contains both $\sigma : s$ and $\sigma : \neg s$, for some s and σ. It is* open *if all possible rules have been applied and it is not contradictory. A tableau is* closed *if all branches are contradictory; it is* open *if at least one branch is open.*

Definition 9.3 *A* tableau proof *for statement s with global axioms G is a closed tableau starting with the branch $\langle \{ 1 : \neg s \}, G\rangle$.*

In a dual way a satisfiability witness is an open branch of the tableau starting with $\langle \{ 1 : s \}, G\rangle$, when the calculus is complete for the fragment at hand.

9.1.2 Decidability in practice

A decision method (rather than a semi-decidable procedure which could be based on first order translations) is important for security analysis because satisfiability gives information on security weaknesses.

For a *decision method* a simple condition, checkable in polynomial time, can be imposed on the global axioms and the consequence s. Associate a graph to the global axioms and the negation of the formula to be proved:

- each atomic principal is represented by a node,
- for every $P \Rightarrow Q$ (under the scope of an even number of negation) draw an arc from the atomic principals in P to those in Q

If the resulting graph is *acyclic* then the tableau construction terminates by using loop checking with an extended notion of the Fischer-Ladner closure. In the embedding of the word problem (§8.3), the principal g creates cycles.

In access control, acyclicity is not a restriction but a requirement [58, 154]: if $\Rightarrow$ is used for hierarchies of groups/roles then cycles are not allowed.

9.2 Some Examples of Deduction

For sake of simplicity, we assume that we can directly reduce $\supset$ and `controls` with the obvious α and β rules rather than translating them back to $\wedge$, $\neg$ and `says` and then apply the corresponding rules.

The first example is the derivation of *hand-off axioms* [1, 101]:

$$\neg(a\, \mathtt{says}\, \bot) \ \supset \ (a\, \mathtt{controls}\, (P \Rightarrow a)).$$

We need a statement of this form whenever principal a must be able to hand over her privileges to P. It cannot be proved within the Hilbert system developed by [1] and must be added to the assumptions. The tableau derivation is shown in Fig. 9.3. Once again, the key step is the application of rule $[U_{gr}]$ which cannot be axiomatized.

To check that global assumptions must be linked to a branch, try the statement below using one set of global assumptions for the whole tableau. This statement is not valid, yet the tableau constructed by using only one set of global assumptions can be closed.

$$(a\, \mathtt{says}\, (a \Rightarrow b) \wedge a\, \mathtt{says}\, r \wedge a\, \mathtt{says}\, \neg r) \supset a \,\&\, b\, \mathtt{says}\, \bot$$

Intuitively this formula is not valid. It states that there is a principal a claiming to speak on behalf of another principal b. Unfortunately, it also states that a is inconsistent. Out of this information there is no reason to conclude that *both* a and b are inconsistent. The agent a may well claim to speak for b (and hence make b also inconsistent) but, after all, a is inconsistent.

As a practical example, we show the deduction necessary for Example 4 (pag.169). We use the logic for the reasoning of the agent C and add a level

$^{(a)}$ $1 : \neg(\neg(a\ \mathtt{says}\ \bot) \supset (a\ \mathtt{controls}\ P{\Rightarrow}a))$	
$G := \{\}$	
$^{(b)}$ $1 : \neg(a\ \mathtt{says}\ \bot)$	by α to (a)
$^{(c)}$ $1 : \neg a\ \mathtt{controls}\ (P{\Rightarrow}a)$	
$^{(d)}$ $1 : a\ \mathtt{says}\ (P{\Rightarrow}a)$	by reduce `controls` to (c)
$^{(e)}$ $1 : \neg(P{\Rightarrow}a)$	
$^{(f)}$ $2 : P\ \mathtt{says}\ x_1$	by $\langle U_{gr}\rangle$ to (e)
$^{(g)}$ $2 : \neg(a\ \mathtt{says}\ x_1)$	
$^{(h)}$ $1.a.3 : \neg\bot$	by $\langle a\rangle$ to (b)
$^{(i)}$ $1.a.3 : P{\Rightarrow}a$	by $[a]$ to (d)
$^{(l)}$ $G_1 := \{\, P{\Rightarrow}a \,\}$	by $[U_{gr}]$ to (i)
$^{(m)}$ $2 : P{\Rightarrow}a$	by *Glob* to (G_1)
$^{(n)}$ $2 : \neg(P\ \mathtt{says}\ x_1)$	by $\langle R_{gr}\rangle$ to (g, m)
contradiction between (g, n)	

Figure 9.3: Tableau Proof of an Hand-off Axiom

of indirection to the original problem by modeling explicitly the smart-card Sc. The statement we want to prove is the following:

$$(K_B\ \mathtt{says}\ (Sc\ \mathtt{says}\ r) \wedge K_S\ \mathtt{says}\ (K_B{\Rightarrow}B) \wedge (B \mid A)\ \mathtt{controls}\ r) \supset r$$

Therefore the tableau starts with

$$1 : \neg\,((K_B\ \mathtt{says}\ (Sc\ \mathtt{says}\ r) \wedge K_S\ \mathtt{says}\ (K_B{\Rightarrow}B) \wedge (B \mid A)\ \mathtt{controls}\ r) \supset r)$$

The tableau proof is shown in Fig. 9.4. In Fig. 9.4 only $[L_{gr}]$-rule is used. A derivation with only $\langle R_{gr}\rangle$-rule is possible.

This is an example of the "not compilability" of $P{\Rightarrow}Q$ into axiom schemata (or rules). Indeed the statement $K_B{\Rightarrow}B$ corresponds to the semantical property $B^{\mathcal{I}} \subseteq K_B^{\mathcal{I}}$, i.e. to the axiom schema $K_B\ \mathtt{says}\ s \supset B\ \mathtt{says}\ s$. The problem is that it is not always valid, since it depends on the server's statement $S\ \mathtt{says}\ (K_B{\Rightarrow}B)$. Without the server's certificate, it doesn't hold. So, if we try to prove a different theorem without this certificate this axiom schema turns out to be invalid. The possibility of adding "on-line" properties is critical here, because delegations and groups membership depend on security policies and current certificates and therefore cannot be fixed a priori in the calculus.

Last, we show the complex deduction which is necessary for Example 1 (page 167) in Fig. 9.5 on page 183. We shorten $SysAdm$ in Sys, $login(telnet)$ in $telnet$, and $read(mail)$ in $mail$. For sake of readability we omit the root of the tableau which is labelled by the negated goal formula:

$^{(a)}$ $1 : \neg((fm\ \mathtt{says}\ (telnet \wedge mail) \wedge maj\ \mathtt{says}\ (fm{\Rightarrow}usr)) \supset (telnet \wedge mail))$

$^{(a)}\ 1 : K_B\ \mathsf{says}\ (Sc\ \mathsf{says}\ r)$
$^{(b)}\ 1 : K_S\ \mathsf{says}\ (K_B \Rightarrow B)$
$^{(c)}\ 1 : B \mid A\ \mathtt{controls}\ r$
$^{(d)}\ 1 : \neg r$
$^{(e)}\ 1 : K_S \Rightarrow S$
$^{(f)}\ 1 : S\ \mathsf{says}\ (K_B \Rightarrow B)$
$^{(g)}\ 1 : S\ \mathtt{controls}\ (K_B \Rightarrow B)$

$\swarrow \searrow$

$^{(h_1)} 1 : \neg(S\ \mathsf{says}\ (K_B \Rightarrow B))$
$\times$

$^{(h_2)} 1 : K_B \Rightarrow B$
$^{(i)}\ 1 : B\ \mathsf{says}\ (Sc\ \mathsf{says}\ r)$

$\swarrow \searrow$

$^{(l_1)}\ 1 : \neg(B \mid A\ \mathsf{says}\ r)$
$^{(m)}\ 1 : \neg(B\ \mathsf{says}\ (A\ \mathsf{says}\ r))$
$^{(n)}\ 1.B.2 : \neg(A\ \mathsf{says}\ r)$
$^{(o)}\ 1.B.2 : Sc\ \mathsf{says}\ r$
$^{(p)}\ 1.B.2 : Sc \Rightarrow A$
$^{(q)}\ 1.B.2 : A\ \mathsf{says}\ r$
$\times$

$^{(l_2)}\ 1 : r$
$\times$

We omit again the negated theorem in the root. The prefixed statements (a), (b), (c), and (d) are obtained by α-rules from the negated theorem. (e) is obtained by rule *Glob* and (f) by rule $[L_{gr}]$ from (b, e). The prefixed statement (g) is added by rule *Glob* too. Then we reduce `controls` in (g) yielding two branches.

One branch closes immediately. In the other branch, we derive (i) by rule $[L_{gr}]$ from (a, h_2) and then split again the branch by reducing (c). Next we apply rule $\langle \mid \rangle$ to (l_1) yielding (m). The prefixed statement (n) is introduced by rule $\langle B \rangle$ from (m). Since $1.B.2$ is now present we can apply rule $[B]$ to (i) and add (o). Then we introduce (p) by rule *Glob* and apply $[L_{gr}]$ to (o, p).

Figure 9.4: Tableau Proof of Delegation without Certificates

9.3 Soundness and (Partial) Completeness

The flexibility and modularity of prefixed tableaux greatly simplifies our job. The overall structure of the proofs of soundness and completeness is similar to the previous ones for modal (§3.6) and dynamic logic (§5.6). We must only adapt the proofs to the new connectives.

9.3.1 Soundness

To prove soundness we map prefixes to states and show that satisfiability is preserved by tableaux rules.

Definition 9.4 *Let $\mathcal{B}$ be a branch and $\langle W, \mathcal{I} \rangle$ a model, a* mapping *is a function $\imath()$ from prefixes to states such that for all σ and $\sigma.\mathbf{a}.n$ present in $\mathcal{B}$ it is $\langle \imath(\sigma), \imath(\sigma.A.n) \rangle \in (\mathbf{a})^{\mathcal{I}}$.*

We need to redefine the concept of SAT branches.

Definition 9.5 *A tableau branch $\langle \mathcal{B}, G_{\mathcal{B}} \rangle$ is SAT in the model $\langle W, \mathcal{I} \rangle$ iff*

1. *$\langle W, \mathcal{I} \rangle$ satisfies $G_{\mathcal{B}}$;*
2. *there is a mapping $\imath()$ such that for every $\sigma : s$ present in $\mathcal{B}$ one has $\imath(\sigma) \Vdash s$.*

A tableau is SAT if one branch is such.

We can now give a proof of the safe extension lemma (a detailed proofs is in appendix C.1).

Theorem 9.6 (Safe Extension) *If T is a SAT tableau for the model $\langle W, \mathcal{I} \rangle$, then the tableau T' obtained by an application of a tableau rule is also SAT for the same model.*

The only new case from §3.6 and §5.6 is the reduction of the operator $\Rightarrow$. The argument requires to apply both the local property and the universal properties of $\Rightarrow$.

To prove soundness of rules $\langle R_{gr} \rangle$ and $[L_{gr}]$ we use the local properties: by hypothesis $P \Rightarrow Q$ holds and then we use the induced properties on the relations of $(P)^{\mathcal{I}}$ and $(Q)^{\mathcal{I}}$.

For rule $[U_{gr}]$ we note that either $(P \Rightarrow Q)^{\mathcal{I}} = W$ or $(P \Rightarrow Q)^{\mathcal{I}} = \emptyset$. Since it is not empty as it contains $\imath(\sigma)$ we conclude it holds everywhere in the model.

For rule $\langle U_{gr} \rangle$ we must update *both* the mapping $\imath()$ *and* the interpretation $()^{\mathcal{I}}$ since we introduce a new x_i and a new n. The extension of the mapping follows the proofs of the corresponding safe extension lemma for modal and dynamic logics (Thm. 3.16 or Thm. 5.10). For the valuation we exploit the fact that x_i is new and therefore we set it to true only to the new world which is introduced by change of mapping.

The soundness theorem follows with a standard argument.

Theorem 9.7 (Strong Soundness) *If s has a tableau proof with global assumptions G then s is a logical consequence of G.*

9.3.2 Completeness

The completeness proof is given for weakly left restricted statements and its extension to right restricted statements is sketched.

At first we we apply a systematic and fair procedure (see [62, 79] for a definition) to our initial tableau. At the end of this process (possibly ad infinitum) we either close the tableau or get an open branch. Then we can show how an open branch can be used to build a model for the initial formula.

Theorem 9.8 (Model Existence) *If $\langle \mathcal{B}, G_{\mathcal{B}} \rangle$ is an open branch with weakly left restricted statements only, then it is SAT in a model $\langle W, \mathcal{I} \rangle$.*

The detailed proof can be found in §C.1, we only sketch the main ideas. The mayor differences with the proof for modal and dynamic logics is that the presence of the $\Rightarrow$ relation imposes constraints on the accessibility relation which are not known a priori.

So we must work in two stages.

1. At first we construct a pre-model as it was a standard modal logic, by using prefixes σ as worlds, assigning the valuation to agents such that $(a)^{\mathcal{J}} = \{ \sigma, \sigma.a.n \}$ and to proposition according the prefixed statements presents in the branch $(r)^{\mathcal{J}} = \{ \sigma \mid \sigma : r \in \mathcal{B} \}$.

2. Then we start an iterative *closure phase* where we construct the final valuation $()^{\mathcal{I}}$ from the prevalaution. For each constraint $A{\Rightarrow}P$ we add all prefixes from $(P)^{\mathcal{I}}$ to $(A)^{\mathcal{I}}$. We must repeat this process until we arrive at a fixed point.

Then we start the hard part of the proof which requires to show that is a prefixed statement occurs on the branch the statement is satisfied in the corresponding world.

Here we need to perform an induction on the number of iterations in the closure phase. The base case (which is also repeated at ecah step of the induction) is the usual argument for modal and dynamic logic: if $\sigma : Q \,\texttt{says}\, s$ and σ^* is Q-connected to σ then we can prove that one also has $\sigma^* : s$. In this part we use the standard modal rules.

From one iteration to the other we add new prefixes to $(a)^{\mathcal{I}}$ so the modal rules are not anymore sufficient to prove the above argument for a. However, we can use the rule $[L_{gr}]$ to generate the $P \,\texttt{says}\, s$ from $a \,\texttt{says}\, s$. Then we repeat the modal argument using $P \,\texttt{says}\, s$.

The completeness theorem follows with a standard argument.

Theorem 9.9 (Strong WL-Completeness) *If s is a logical consequence of G and $G \cup \{\neg s\}$ is weakly left restricted then s has a tableau proof.*

For the right and request restricted fragment of the language the key point is that we only have literals or statements of the form $P{\Rightarrow}A$ under the scope of `says`. The operator $\Rightarrow$ does not create problems given its global nature and the only difficult part is due to literals l (r or $\neg r$).

The previous proof does not work since $\sigma : P \,\texttt{says}\, s$ for non atomic P cannot propagate over $\sigma : P{\Rightarrow}a$. However we can prove the dual of the induction step: if $\sigma : \neg(a \,\texttt{says}\, l)$ and $P{\Rightarrow}A$ is also present then there is is a $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{I}}$ such that $\sigma^* : \neg l$. This is sufficient for the proof of the following theorem:

Theorem 9.10 (Strong WRR-Completeness) *If s is a logical consequence of G and $G \cup \{\neg s\}$ is weakly right and request restricted then s has a proof.*

9.4 Using Efficient and Approximate Techniques

We do not need to devise ad hoc mechanisms to make these tableaux efficient; we simply apply the techniques we have developed for propositional logic (§2) and modal logics (§3).

The application of the propositional and modal constraint propagation techniques can be immediately done without difficulties.

The *confluence of the calculus* follows from the proof of confluence for modal logic K(§4.3). The *decidability techniques* using loop-checking can be applied in the same fashion (§4.4).

With the DEC-SRC calculus, the set of global assumptions will seldom be empty, so we cannot use the local check on the length of prefixes for termination. Still, if all speaksfor operators $P \Rightarrow Q$ are under the scope of an odd number of negations, then we can apply the techniques of §4.5 to obtain a *polynomial space algorithm* for validity checking. With $P \Rightarrow Q$ operators, we can apply the techniques devised for PDL (§5.7) to transform the NEXPTIME naive application of tableau search strategies into a EXPTIME algorithm.

The second step is the application of the approximation techniques for propositional (§6) and modal logics (§7). The framework is unchanged. The approximate closure and the approximated reduction of a branch can be performed under the same conditions. We only clarify the *semantics of approximation* for this particular application.

The first operation we have in mind is checking the *approximate consistency of an access control policy.* Complete approximations tend to be more conservative. They would like to have a model where only high level properties are sufficient to grant the consistency of the policy, without getting bogged down in details. So, if a model requires to set down what happens in detail they say that the access control policy is inconsistent. Sound approximations, assume that once high level properties are consistent we can always fix the low level details to get consistency.

The dual situation is presented by *approximate decisions on granting a request*, if these are based on the idea that a request should be an approximate logical consequence of our access control policy and the statements of the various principals. With sound approximations we take a conservative approach: when granting a request by approximate reasoning we know that the request would have been granted also by the full fledged analysis of the system. We just neglected some minor access control privileges. The trade-off is that we reject (at a first approximation) legitimate access control requests which require careful analysis of low level privileges. With complete approximations we take a more liberal approach: we give a quick permission rather than getting bogged down in low level details.

With the propositional approximation (§6.2) we need to give the semantic intuition behind the partition into three sets, for *interesting*, *unknown*, and *incoherent* propositions. For the DEC-SRC calculus, propositions correspond to atomic requests and their interpretation is that they may be granted (true) or not (false). Therefore interesting requests are the only requests which we want to consider at this stage of approximation. The rest are low level details we would like to neglect.

We take a don't care approach towards incoherent requests: they may also be inconsistent with each other. Recall that incoherent properties cannot be used to derive a proof. So we use imprecise requests for sound approximations: never grant an access control request if it requires to go down into details.

We take a dual approach on unknown request: we reject all models where the truth value of unknown requests must be taken into account. We can use this technique for complete approximations: never accepts the consistency of an access control policy if it requires to go down into details.

Modal approximations (§7.2) offer again three possible approaches: *normal* agents, *k-skeptical* agents and *k-credulous* agents. In a similar way of the propositional calculus we can use these techniques for approximating the answer of the system for both granting access and checking consistency.

$$
\begin{array}{l}
{}^{(a)}\ 1 : \ldots \\
{}^{(b)}\ 1 : fm\ \mathtt{says}\ (telnet \wedge mail) \\
{}^{(c)}\ 1 : maj\ \mathtt{says}\ (fm \Rightarrow Usr) \\
{}^{(d)}\ 1 : \neg(telnet \wedge mail) \\
{}^{(e)}\ 1 : maj \Rightarrow Sys \\
{}^{(f)}\ 1 : Sys\ \mathtt{says}\ (fm \Rightarrow Usr) \\
{}^{(g)}\ 1 : Sys\ \mathtt{controls}\ (fm \Rightarrow Usr)
\end{array}
$$

$$
\swarrow \quad \searrow
$$

$$
\begin{array}{ll}
{}^{(h_1)} 1 : \neg(Sys\ \mathtt{says}\ (fm \Rightarrow Usr)) & {}^{(h_2)} 1 : fm \Rightarrow Usr \\
\quad \times & {}^{(i)}\ 1 : Usr\ \mathtt{says}\ (telnet \wedge mail) \\
& \qquad \swarrow \quad \searrow \\
& {}^{(l_1)} 1 : \neg telnet \qquad {}^{(l_2)} 1 : \neg mail \\
& \quad \mathcal{T}_1 \qquad\qquad\qquad \mathcal{T}_2
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{c}{\mathcal{T}_1} \\
\multicolumn{2}{c}{{}^{(m)} 1 : Usr\ \mathtt{controls}\ telnet} \\
\multicolumn{2}{c}{\swarrow \quad \searrow} \\
{}^{(n_1)} 1 : \neg(Usr\ \mathtt{says}\ telnet) & {}^{(n_2)} 1 : telnet \\
{}^{(o)}\ 1.Usr.2 : \neg telnet & \quad \times \\
{}^{(p)}\ 1.Usr.2 : telnet \wedge mail & \\
{}^{(q)}\ 1.Usr.2 : telnet & \\
{}^{(s)}\ 1.Usr.2 : mail & \\
\quad \times &
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{c}{\mathcal{T}_2} \\
\multicolumn{2}{c}{{}^{(t)}\ 1 : (fm\ \&\ Usr)\ \mathtt{controls}\ mail} \\
\multicolumn{2}{c}{\swarrow \quad \searrow} \\
{}^{(u_1)} 1 : \neg((fm\ \&\ Usr)\ \mathtt{says}\ mail) & {}^{(u_2)} 1 : mail \\
\qquad \swarrow \quad \searrow & \quad \times \\
{}^{(v_1)} 1 : \neg(fm\ \mathtt{says}\ mail) \qquad {}^{(v_2)} 1 : \neg(Usr\ \mathtt{says}\ mail) & \\
\quad \mathcal{T}_{21} \qquad\qquad\qquad\qquad \mathcal{T}_{22} &
\end{array}
$$

We recall that the prefixed statement (a) (omitted here and shown in the text) is the negation of the goal formula. We obtain (b), (c), and (d) from (a) by a number of α-rules. The prefixed statement (e) is introduced from the assumptions by rule *Glob*. From (b, e) we derive (f) by the $[L_{gr}]$-rule. (g) is also introduced by rule *Glob* and the direct reduction of `controls` in (g) yields (h_1) and (h_2).

One branch closes immediately. In the other branch we apply rule $[L_{gr}]$ to (h_2, b) and obtain (i). Next we apply rule β to (d) and split the tableau with (l_1) and (l_2).

As for $\mathcal{T}_1$ we first apply rule *Glob* to introduce (m) and then reduce `controls` with a β-rule. Again, one branch closes immediately and, in the other branch, we apply rule $\langle Usr \rangle$ to (n_1) yielding (o). Since the prefix $1.Usr.2$ is present we can apply rule $[Usr]$ to (i) and add (p). Then we apply rule α to (p) and we are done.

To close $\mathcal{T}_2$ we apply rule *Glob* to obtain (t) and then reduce `controls`. We apply $\langle\,\&\,\rangle$ to (u_1) splitting further the tableau. The resulting tableau $\mathcal{T}_{21}$ and $\mathcal{T}_{22}$ are substantially identical to $\mathcal{T}_1$, replacing *telnet* with *mail*.

Figure 9.5: Deduction for the request of Example 1

Part IV

Conclusion and Bibliography

Chapter 10

Conclusion

In this dissertation we proposed a number of innovations for tableau calculi for a wide range of logics. We can now look back to our initial aims, of designing tableau methods which are

- effective,
- efficient,
- approximable,
- widely applicable

and see what has been achieved.

As for *effectiveness* we have designed novel decision methods for multimodal logics and dynamic logic with converse, in the latter case closing a problem left open by Pratt [144]. For modal logics we have been able to eliminate the usage of analytic cut and to design a simple (polynomial time) technique for setting decision procedures for transitive modal logics (modeling agents with positive introspection) which can be directly applied to the translation methods of Ohlbach [135, 136] and Nonnegart [133], for which only exponential bounds were proved by Schmidt [157] with more complex techniques. For dynamic logic, we have been able to transform the traditional "generate-and-test" construction of automata theoretic and tableau techniques —see the work of Vardi & Wolper [174], Pratt [144] or the survey edited by Moller & Birtwistle [130] — into a single phase. Merging the generation of the pseudo-model, which satisfies formulae like "next A", with the testing phase of eventualities, which check formulae like "eventually A", simplifies notably the readability of the proof search and its efficiency.

As for *efficiency*, we have shown how the incorporation of constraint propagation techniques subsumes a number of other proposals for propositional deduction, such as the Davis Putnam Longeman Loveland procedure [42], and leads to a notable speed-up on hard benchmarks of industrial relevance. On modal logic we have been able to prove the confluence of our calculus, an important property in the quest for efficient proof search without backtracking. We have also identified the proof techniques needed for a provably PSPACE proof search algorithm, for validity checking in modal logic. This has particular relevance since we have also shown that translation methods á la Ohlbach [135, 136] requires exponential space. We have also shown a number of negative results on the maximum efficiency which is possible to achieve for certain deduction problems, no matter the deductive technique at hand. Also for dynamic logic we have designed a particular technique to transform a naive NEXPTIME proof search algorithm into an EXPTIME algorithm.

As for *approximation* techniques we have introduced the key notion of approximate proof, matching the original intuition of Wang [179] and a number of recent requirements on anytime reasoning systems proposed by Russel & Zilberstein [150]. The intuition of splitting the atomic constituents of the language into interesting, incoherent and unknown parts makes it possible to devise in an uniform way sound, complete and multi-direction approximations to classical reasoning. The generality of our approach has made it possible to subsume a number of approaches previously developed by Cadoli & Schaerf [22, 155], Dalal [39, 38, 37], Levesque [104] and is very close to a number of other proposals, such as one by Fagin, Halpern & Vardi [55, 56].

As for *wide applicability*, we have chosen to apply our methodology and proof techniques developed beyond the realms of the abstract syntax of propositional, modal and dynamic logics. We have first tried our techniques on the propositional IFIP benchmark on hardware verification and then singled out a comprehensive case study. Hence, the choice of the calculus for access control developed at the Digital System Research Center by Abadi, Lampson *et al.* [1, 101, 180] as the formal backbone behind a real implemented systems. Access control is an important part of computer security and the successful application of our methodology to this problem is an important step in the direction of formal verification with tableau methods.

10.1 Methodological Contributions

These technical contributions can be further distilled into few methodological innovations. Three main threads run throughout this dissertation:

- the proof theoretic use of labels as a form of *naming*;
- the use of equational reasoning for *constraint propagation*;
- the notion of *incremental proof construction* and hence of approximation.

The use of *naming*, of proof-theoretic labels for semantical "things" has been pervasive throughout this dissertation. Names for possible worlds in a

Kripke model, for eventualities in dynamic logic, for unsubstantiated delegation certificates in the DEC-SRC calculus have been the proof theoretical tools with which we have build most results.

The use of *constraint propagation* to eliminate redundancies in the proof search by equality reasoning is the second important contribution. Constraint propagation provides a general technique which subsumes many other, seemingly different methods such as `DPLL` [41, 42], `KE` [35], `KSAT` [72], and gives a huge speed-up on hard benchmarks for propositional reasoning. The experimental analysis we have carried shows that constraint propagation is a necessary tool for fast tableau theorem proving. It is also essential for the development of computational guarantees on anytime proof procedures.

This line of research, embedding a form of equational simplification into deductive methods, is likely to be one of the key issues of automated deduction in the near future (for instance see the work of Beckert on equality [9], Uribe & Stickel on unit propagation [170], and Voronkov on subsumption [176]).

The last thread is *incremental proof construction*: proofs are large objects, exponentially or even super exponentially large; to dominate them we must be able to visit them locally and construct them incrementally. The design of our deductive methods is geared towards proof rules which makes this incremental construction possible, from propositional logic (with local rather than global simplification rules) to dynamic logic (with the use of naming for eventualities). The quest for incremental construction is then naturally casted in the notion of *approximate proof*. Approximate proofs converge towards exact proofs as soon as our resources (time, motivations etc.) make that possible. The construction of an approximate proof theory has provided an unifying view of many different proposals for tractable reasoning developed in AI such as those of Dalal [39, 38, 37], Fagin, Halpern & Vardi [55, 56], Cadoli & Schaerf [22, 155].

10.2 Open Problems and Future Works

The closing of old open problems that we have seen throughout this dissertation is matched by the opening of new ones. For some of them we can already identify the direction in which a potentially successful research may be carried, while for others we must explore an unknown territory.

An open problem is the *experimental validation of modal and dynamic logics theorem proving*. Hard benchmarks which are thoroughly assessed and are of industrial relevance (e.g. the IFIP benchmarks [27]) have been developed only for propositional logics. As a consequence, claims of presumed (comparative) efficiency are extremely weak and can be easily reversed. For instance, the contradictory results and rebuttals between Giunchiglia & Sebastiani [72] and Hustadt & Schmidt [90] are a consequence of this lack of assessed benchmarks. We have tried to clear the air a bit with complexity theoretical analysis, but the development of industrial strength benchmark for modal logic is an absolute need for the theorem proving community. This is witnessed by the organization at the TABLEAUX conference of a comparison for modal provers.

The next issue is the *extension to first and higher order logics.* The expressiveness of these logic and, in particular, the use of variables and quantifiers, is necessary to represent many important problems in formal verification (see for instance Rushby [149] or Paulson [142]).

Constraint propagation techniques cannot be simply plugged over, without risking major inefficiencies: variables cannot be dealt with simple syntactic checking and rewriting. For instance the formula $p(x)$ does not represent the same property in $\forall x.p(x)$ and $\exists x.p(x)$. On the other side $\forall x.p(x)$ and $\forall y.p(y)$ may well be considered identical.

A possible approach is to combine naming with constraint propagation. In a nutshell, to explicitly represent the free variables which are universally quantified in a formula and the variables that are not. This idea, following the proposals of Beckert [11, 9] and Voronkov [45] for handling variable unification, has been pursued already by Massacci [117] and looks promising.

The extension of approximation methods looks more problematic: we can partition objects (first order terms) and predicates in interesting and not interesting classes but then, how do we evaluate an interesting predicate on a meaningless object? The difficulty is the definition of a clear semantics for sound and complete approximations. A promising approach has been proposed by Chaudhuri & Kolaitis [26] for the approximation of Datalog programs.

The introduction and extension of approximate theorem proving techniques can well be applied beyond our idea of access control where approximate proofs guaranteed only a partial form of security, trading off guarantees for speed. A possible field of application is interactive theorem proving for formal verification. Timely response is a key requisite for interactive proving: if a technique does not respond within few minutes, it is likely to be timed-out by the user. In this setting, approximate proofs, combined with anytime proof search procedures, could be a definite asset: the user will be left with an approximate proof (rather than just "no reply") from which she can start for further improvements. Deductive planning and scheduling in a reactive environment is also another possible application, in this case fulfilling the original requirement of Dean & Boddy for anytime deductive planners [44].

The major problem opened by the introduction of approximation techniques is the definition of a *measure of approximation.* This has been pretty well defined for the approximation of satisfiability problems (for instance see Ausiello *et al.* [6]): we try to maximize the number of formulae satisfied by a truth assignment, and the ratio between the maximum number of satisfiable formulae and our approximate result is the measure we are looking for.

This measure could be adapted for sound approximation of propositional and modal logics but it does not fit with complete (and unsound) approximations. Entirely new techniques must be investigated in this direction.

Bibliography

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[2] M. Abadi and Z. Manna. Modal theorem proving. In *Proc. of the 8th International Conf. on Automated Deduction (CADE-86)*, 1986.

[3] A. Anderson and N. Belnap. *Entailment: the Logic of Relevance and Necessity.* Princeton University Press, Princeton, NJ, 1975.

[4] R. Anderson. A security policy model for clinical information systems. In *Proc. of the 15th IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, 1996.

[5] C. Angelo, D. Verkest, L. Claesen, and H. De Man. On the comparison of HOL and Boyer-Moore for formal hardware verification. *Formal Methods in System Design*, 2(1):45–72, 1993.

[6] G. Ausiello, P. Crescenzi, and M. Protasi. Approximate solution of NP optimization problems. *Theoretical Computer Science*, 150(1):1–55, 1995.

[7] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In H. Boley and M. Richter, editors, *Proc. of the International Workshop on Processing Declarative Knowldege (PDK-91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.

[8] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper tableaux. In J.J. Alferes, L.M. Pereira, and E. Orlowska, editors, *Proc. of the 6th European Workshop on Logics in Artificial Intelligence (JELIA-96)*, volume 1126 of *Lecture Notes in Artificial Intelligence*, pages 1–17. Springer-Verlag, 1996.

[9] B. Beckert. Semantic tableaux with equality. *J. of Logic and Computation*, 7(1):38–58, 1997.

[10] B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. In Didier Galmiche, editor, *Proc. of the International Conf. on Analytic Tableaux and Related Methods (TABLEAUX-97)*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 91–108. Springer-Verlag, 1997.

[11] B. Beckert and J. Possega. `leanTAP`: Lean tableau-based deduction. *J. of Automated Reasoning*, 15(3):339–358, 1995.

[12] D. Bell and L. La Padula. Secure computer systems: unified exposition and MULTICS. Report ESD-TR-75-306, The MITRE Corporation, March 1976.

[13] S. Bellantoni, T. Pitassi, and A. Urquhart. Approximation and small-depth frege proofs. *SIAM J. of Computing*, 21(6):1161–1179, 1992.

[14] N. Belnap. A useful four-valued logic. In G. Epstein and J. Dunn, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. Reidel, 1977.

[15] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In T. Margaria and B. Steffen, editors, *Proc. of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 107–126. Springer-Verlag, 1996.

[16] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for CTL^*. In *Proc. of the 10th International Conf. of Logic in Computer Science (LICS-95)*, pages 388–397. IEEE Computer Society Press, 1995.

[17] W. Bibel. *DEDUCTION, Automated Logic.* Academic Press, 1993.

[18] P. Blackburn and E. Spaan. A modal perspective on computational complexity of attribute value grammar. *J. of Logic, Language and Information*, 2:129–169, 1993.

[19] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[20] M. Burrows, M. Abadi, and R.. Needham. A logic for authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[21] S. Buvač, V. Buvač, and I. Mason. Metamathematics of contexts. *Fundamenta Informaticae*, 23(3):263–301, 1995.

[22] M. Cadoli. *Tractable Reasoning in Artificial Intelligence*, volume 941 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 1995.

[23] L. Catach. TABLEAUX, a general theorem prover for modal logics. *J. of Automated Reasoning*, 7:489–510, 1991.

[24] S. Cerrito and M. Cialdea Myer. Hintikka multiplicities in matrix decision methods for some propositional modal logics. In Didier Galmiche, editor, *Proc. of the International Conf. on Analytic Tableaux and Related Methods (TABLEAUX-97)*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 138–152. Springer-Verlag, 1997.

[25] S. Cerrito and M. Cialdea Myer. A polynomial translation of S4 into T and contraction-free tableaux for S4. *Bulletin of the Interest Group in Pure and Applied Logic*, 5(2):287–300, 1997.

[26] S. Chaudhuri and P. Kolaitis. Can datalog be approximated? *J. of Computer and System Sciences*, 55(2):355–369, 1997.

[27] L. Claesen, editor. *Formal VLSI Correctness Verification: VLSI Design Methods*, volume II. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[28] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *Proc. of the 6th IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, 1987.

[29] A. Condon. Approximate solutions to problems in PSPACE. *SIGACT News*, (9):4–13, 1996.

[30] S. Cook. The complexity of theorem proving procedures. In *Proc. of the 3rd ACM Symposium on Theory of Computing (STOC-71)*, pages 151–158, 1971.

[31] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. In *Proc. of the 11th National Conf. on Artificial Intelligence (AAAI-93)*, pages 21–27. AAAI Press/The MIT Press, 1993.

[32] J. Crawford and B. Kuipers. Towards a theory of access-limited logic for knowledge representation. In *Proc. of the 1st International Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)*, pages 67–78. Morgan Kaufmann, Los Altos, 1989.

[33] F. Cuppens and R. Demolombe. A deontic logic for reasoning about confidentiality. In *3rd International Workshop on Deontic Logic in Computer Science*, Sesimbra, Portugal, 1996.

[34] G. D'Agostino, A. Montanari, and A. Policriti. A set-theoretic translation method for polymodal logics. *J. of Automated Reasoning*, 15:317–337, 1995.

[35] M. D'Agostino and M. Mondadori. The taming of the cut. *J. of Logic and Computation*, 4(3):285–319, 1994.

[36] M. Dalal. Efficient propositional constraint propagation. In W. Swartout, editor, *Proc. of the 10th National Conf. on Artificial Intelligence (AAAI-92)*, pages 409–414. AAAI Press/The MIT Press, 1992.

[37] M. Dalal. Anytime families of tractable propositional reasoners. In *International Symposium on Artificial Intelligence and Mathematics AI/MATH-96*, pages 42–45, 1996. Extended version submitted to Annals of Mathematics and Artificial Intelligence.

[38] M. Dalal. Semantics of an anytime family of reasoners. In *Proc. of the 12th European Conf. on Artificial Intelligence (ECAI-96)*, pages 360–364. John Wiley & Sons, 1996.

[39] M. Dalal and D. Etherington. Tractable approximate deduction using limited vocabularies. In *Proceeding of the 9th Canadian Conf. on Artificial Intelligence (AI-92)*, pages 206–221, 1992.

[40] M. Davis. Eliminating the irrelevant from mechanical proofs. In N.C. Metropolis, A. H. Taub, J. Todd, and C.B. Tompkins, editors, *Proc. of the 15th Symposium in Applied Mathematics*, pages 15–30. American Mathematical Society, 1963.

[41] M. Davis and H. Putnam. A computing procedure for quantificational theory. *J. of the ACM*, 7(3):201–215, 1960.

[42] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[43] G. De Giacomo and F. Massacci. Tableaux and algorithms for propositional dynamic logic with converse. In M. McRobbie and J. Slaney, editors, *Proc. of the 13th International Conf. on Automated Deduction (CADE-96)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 613–628. Springer-Verlag, 1996. Extended version in accepted for publication in *Information and Computation*, special issue on the"Federated Logic Conferences".

[44] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. of the 7th National Conf. on Artificial Intelligence (AAAI-88)*, pages 49–54. AAAI Press/The MIT Press, 1988.

[45] A. Degtyarev and A. Voronkov. Equality elimination for the tableau method. In J. Calmet and C. Limongelli, editors, *Proc. of the International Symposium on the Design and Implementation of Symbolic Computation Systems (DISCO-96)*, volume 1128 of *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag, 1996.

[46] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. of the 4th International Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 551–561. Morgan Kaufmann, Los Altos, 1994.

[47] A. del Val. An analysis of approximate knowledge compilation. In C. Mellish, editor, *Proc. of the 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 830–836. Morgan Kaufmann, Los Altos, 1995.

[48] J. Delgrande. Accessibility in logics of explicit belief. In *Proc. of the 3rd International Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, pages 450–461. Morgan Kaufmann, Los Altos, 1992.

[49] S. Demri. Uniform and non uniform strategies for tableaux calculi for modal logics. *J. of Applied Non-Classical Logics*, 5(1):77–96, 1995.

[50] F. Donini, F. Massacci, D. Nardi, and R. Rosati. A uniform tableaux method for nonmonotonic modal logics. In J. Alferes, L. Pereira, and E. Orlowska, editors, *Proc. of the 6th European Workshop on Logics in Artificial Intelligence (JELIA-96)*, volume 1126 of *Lecture Notes in Artificial Intelligence*, pages 87–103. Springer-Verlag, 1996.

[51] J. Doyle and R. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297, 1991.

[52] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment: crash failure. *Information and Computation*, 88(2), 1990.

[53] A. Emerson. Automated temporal reasoning about reactive systems. In Moller and Birtwistle [130], pages 41–101.

[54] P. Enjalbert and L. Fariñas del Cerro. Modal resolution in clausal form. *Theoretical Computer Science*, 65:1–33, 1989.

[55] R. Fagin and J. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[56] R. Fagin, J. Halpern, and M. Vardi. A nonstandard approach to the logical omniscience problem. *Artificial Intelligence*, 79(2):203–240, 1995.

[57] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.

[58] D. Ferraiolo, J. Cugini, and R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proc. of the Annual Computer Security Applications Conf.*. IEEE Computer Society Press, 1995.

[59] D. Ferraiolo, D. Gilbert, and N. Lynch. An examination of federal and commercial access control policy needs. In *Proc. of the 16th NIST-NCSC National (U.S.) Computer Security Conf.*, pages 107–116, 1993.

[60] N. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[61] F. Fitch. Tree proofs in modal logic. *J. of Symbolic Logic*, 31, 1966.

[62] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Reidel, Dordrecht, 1983.

[63] M. Fitting. *First Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.

[64] M. Fitting. Basic modal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, pages 365–448. Oxford University Press, 1993.

[65] N. Friedman and J. Halper. On the complexity of conditional logics. In *Proc. of the 4th International Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 202–213, 1994.

[66] D. Gabbay. Labelled deductive systems. Technical Report MPI-I-94-223, Max Plank Institute für Informatik (MPII), Saarbrüken, Germany, 1994. To appear as a book by Oxford University Press.

[67] G. Gallo and G. Pretolani. A new algorithm for the propositional satisfiability problem. *Discrete Applied Mathematics*, 60:159–179, 1995.

[68] G. Gallo and M. Scutellà. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–227, 1988.

[69] I. Gent. Theory matrices (for modal logics) using alphabetical monotonicity. *Studia Logica*, 52:233–257, 1993.

[70] G. Gentzen. In M. Szabo, editor *The collected papers of G. Gentzen*. North-Holland Publ. Co., Amsterdam, Amsterdam, 1969.

[71] M. Georgeff and A. Rao. The semantics of intention maintenance for rational agents. In Chris Mellish, editor, *Proc. of the 14th International Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 704–710. Morgan Kaufmann, Los Altos, 1995.

[72] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In Michael A. McRobbie and John K. Slaney, editors, *Proc. of the 13th International Conf. on Automated Deduction (CADE-96)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 583–597. Springer-Verlag, 1996.

[73] F. Giunchiglia and T. Walsh. The inevitability of inconsistent abstract spaces. *J. of Automated Reasoning*, 11:23–41, 1993.

[74] L. Giuri and P. Iglio. A formal model for role based access control with constraints. In *Proc. of the 10th IEEE Computer Security Foundations Workshop*, pages 136–145. IEEE Computer Society Press, 1996.

[75] J. Glasgow, G. MacEwen, and P. Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, 1992.

[76] V. Goranko. Modal definability in enriched languages. *Notre Dame J. of Formal Logic*, 31(1), 1990.

[77] V. Goranko and S. Passy. Using the universal modality: Gains and questions. *J. of Logic and Computation*, 2(1):5–30, 1992.

[78] M. Gordon and T. Melham. *Introduction to HOL*. Cambridge University Press, 1993.

[79] R. Goré. Tableaux method for modal and temporal logics. Technical Report TR-ARP-15-5, Australian National University, 1995. To appear as chapter on the "Handbook of Tableau Methods" by Kluwer.

[80] J. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed computing*, 3(4):159–177, 1989.

[81] J. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(2):361–372, 1995.

[82] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[83] J. Harrison. Binary decision diagrams as a HOL derived rule. *The Computer Journal*, 38:162–170, 1995.

[84] J. Harrison. Stalmarck's algorithm as a HOL derived rule. In *Proc. of the 9th International Conf. on Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, 1996.

[85] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[86] A. Heuerding, M. Seyfried, and H. Zimmermann. Efficient loop-check for backward proof search in some non-classical logics. In *Proc. of the 5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX-96)*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 210–225. Springer-Verlag, 1996.

[87] J. Hintikka. *Knowledge and Belief.* Cornell University Press, Ithaca, New York, 1962.

[88] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. of the ACM*, 27(4):797–821, 1980.

[89] G. Hughes and M. Cresswell. *An Introduction to Modal Logic.* Methuen, London, 1968.

[90] U. Hustadt and R. Schmidt. On evaluating decision procedure for modal logic. In M. Pollack, editor, *Proc. of the 15th International Joint Conf. on Artificial Intelligence (IJCAI-97)*, pages 202–207, 1997.

[91] D. Johnson and M. Garey. *Computers and Inctractability: A Guide to the Theory of NP-Completeness.* Freeman, S. Francisco, 1979.

[92] D. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 67–162. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[93] S. Kanger. Law and logic. *Theoria*, 38(3):105–132, 1972.

[94] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of the 10th European Conf. on Artificial Intelligence (ECAI-92)*, pages 359–363. John Wiley & Sons, 1992.

[95] D. Kozen and J. Tiuryn. Logic of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume II, chapter 14, pages 789–840. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[96] S. Kripke. Semantical analysis of modal logic I: normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[97] S. Kripke. Semantical analysis of modal logic II: non-normal propositional calculi. In J.W. Addison, L. Henkin, and A. Tarski, editors, *Symposium on the Theory of Models*, volume 31 of *SLFM*, pages 206–220. North-Holland Publ. Co., Amsterdam, 1965.

[98] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. of Computing*, 6(3):467–480, September 1977.

[99] G. Lakemeyer. Limited reasoning in first-order knowledge bases. *Artificial Intelligence*, 71:1994, 1994.

[100] G. Lakemeyer. Limited reasoning in first-order knowledge bases. *Artificial Intelligence*, 84:209–255, 1996.

[101] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[102] B. Lampson. Protection. *ACM Opearting Systems Reviews*, 8(1):18–24, 1974.

[103] R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableau calculi. *J. of Automated Reasoning*, 13(3):297–337, 1994.

[104] H. Levesque. A logic of implicit and explicit belief. In *Proc. of the 4th National Conf. on Artificial Intelligence (AAAI-84)*, pages 198–202, 1984.

[105] H. Levesque. Logic and the complexity of reasoning. *J. of Philosophical Logic*, 17:355–389, 1988.

[106] H. Levesque. All I know, a study in autoepistemic logic. *Artificial Intelligence*, 42, 1990.

[107] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems (Safety)*, volume 2. Springer-Verlag, 1995.

[108] M. Marathe, H. Hunt III, and S. Ravi. The complexity of approximating PSPACE-complete problems for hierarchical specifications. In *Proc. of the International Colloqium on Algebra and Programming (ICALP-93)*, pages 77–88, 1993.

[109] M. Marathe, H. Hunt III, R. Stearn, and V. Radhakrishnan. Approximation schemes for PSPACE-complete problems for succinct specifications. In *Proc. of the 26th ACM SIGACT Symposium on Theory of Computing (STOC-94)*, pages 468–477. ACM Press and Addison Wesley, 1994.

[110] W. Marek, G. F. Schwarz, and M. Truszczynski. Modal nonmonotonic logics: ranges, characterization, computation. *J. of the ACM*, 40(4):963–990, 1993.

[111] B. Marick. The `verus` design verification system. In *Proc. of the 2nd IEEE Symposium on Security and Privacy*, pages 150–157. IEEE Computer Society Press, 1983.

[112] J. Maslov. The inverse method for establishing deducibility for logical calculi. In V. Orevkov, editor, *The Calculi of Symbolic Logic I*, number 98 in Proc. of the Steklov Institue of Mathematics, pages 25–98. American Mathematical Society, 1968.

[113] F. Massacci. Strongly analytic tableaux for normal modal logics. In A. Bundy, editor, *Proc. of the Twelfth International Conf. on Automated Deduction (CADE-94)*, volume 814 of *Lecture Notes In Artificial Intelligence*, pages 723–737. Springer-Verlag, 1994.

[114] F. Massacci. Superficial tableaux for contextual reasoning. In S. Buvač, editor, *Proc. of the AAAI-95 Fall Symposium on "Formalizing Context"*, number FS-95-02 in AAAI Technical Reports Series, pages 60–66. AAAI Press/The MIT Press, 1995.

[115] F. Massacci. Approximate reasoning for contextual databases. In M. G. Radle, editor, *Proc. of the Eighth IEEE International Conf. on Tools with Artificial Intelligence (ICTAI-96)*, pages 308–315. IEEE Computer Society Press, 1996.

[116] F. Massacci. Contextual reasoning is NP-complete. In W. J. Clancey and D. Weld, editors, *Proc. of the Thirteenth National Conf. on Artificial Intelligence (AAAI-96)*, pages 621–626. AAAI Press/The MIT Press, 1996.

[117] F. Massacci. A proof theory for tractable approximations of propositional reasoning. In M. Lenzerini, editor, *Proc. of the Fifth Conf. of the Italian Association for Artificial Intelligence (AI*IA-97)*, volume 1321 of *Lecture Notes In Artificial Intelligence*, pages 219–230. Springer-Verlag, 1997.

[118] F. Massacci. Reasoning about security: a logic and a decision methods for role-based access control. In D. Gabbay, R. Kruse, A. Nonnegart, and H. J. Ohlbach, editors, *Proceeding of the International Joint Conf. on Qualitative and Quantitative Practical Reasoning (ECSQARU/FAPR-97)*, volume 1244 of *Lecture Notes In Artificial Intelligence*, pages 421–435. Springer-Verlag, 1997.

[119] F. Massacci. Simplification with renaming: A general proof technique for tableau and sequent-based provers. Technical Report 424, Computer Laboratory, University of Cambridge (UK), 1997.

[120] F. Massacci. Tableaux methods for access control in distributed systems. In D. Galmiche, editor, *Proc. of the International Conf. on Analytic Tableaux and Related Methods (TABLEAUX-97)*, volume 1227 of *Lecture Notes In Artificial Intelligence*, pages 246–260. Springer-Verlag, 1997.

[121] F. Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In H. de Swart, editor, *Proc. of the Second International Conf. on Analytic Tableaux and Related Methods (TABLEAUX-98)*, Lecture Notes In Artificial Intelligence. Springer-Verlag, 1998.

[122] F. Massacci. Tableaux methods for formal verification in multi-agent distributed systems. *J. of Logic and Computation*, 8(3):1–28, 1998. To appear in the special issue on "Multi-agent Systems".

[123] D. McAllester. Truth maintenance. In *Proc. of the 8th National Conf. on Artificial Intelligence (AAAI-90)*, pages 1109–1116. AAAI Press/The MIT Press, 1990.

[124] J. McCarthy. notes on formalizing context. In *Proc. of the 13th International Joint Conf. on Artificial Intelligence (IJCAI-93)*, pages 555–560. Morgan Kaufmann, Los Altos, 1993.

[125] C. McCollum, J. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC - defining new forms of access control. In *Proc. of the 9th IEEE Symposium on Security and Privacy*, pages 190–200, 1990.

[126] M. McFarland. Formal verification of sequential hardware: a tutorial. *IEEE Trabsactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(5):633–654, 1993.

[127] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - Asiacrypt 94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150. Springer-Verlag, 1995.

[128] G. Mints. Gentzen-type systems and resolution rules. In *International Conf. on Computer Logic (COLOG)*, volume 417 of *Lecture Notes in Computer Science*, pages 198–231. Springer-Verlag, 1988.

[129] J. Moffet and M Sloman. Policy hierarchies for distributed systems management. *IEEE J. on Selected Areas in Communications*, 11(9), 1993.

[130] F. Moller and G. Birtwistle, editors. *Logics for Concurrency (Structure versus Automata)*, volume 1043 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[131] D. Mundici. Satisfiability in many-valued sentential logic is NP-complete. *Theoretical Computer Science*, 52(1-2):145–153, 1987.

[132] N. Murray and E. Rosenthal. On the computational intractability of analytic tableau methods. *Bulletin of the Interest Group in Pure and Applied Logic*, 2(2):205–228, 1994.

[133] A. Nonnengart. First-order modal logic theorem proving and functional simulation. In *Proc. of the 13th International Joint Conf. on Artificial Intelligence (IJCAI-93)*, pages 80–85. Morgan Kaufmann, Los Altos, 1993.

[134] H. Ohlbach. A resolution calculus for modal logic. In *Proc. of the 9th International Conf. on Automated Deduction (CADE-88)*, volume 310 of *Lecture Notes in Computer Science*, pages 500–516. Springer-Verlag, 1988.

[135] H. Ohlbach. Semantic-based translation methods for modal logics. *J. of Logic and Computation*, 1(5):691–746, 1991.

[136] H. Ohlbach. Translation methods for non-classical logics - an overview. *Bulletin of the Interest Group in Pure and Applied Logic*, 1(1):69–89, 1993.

[137] M. Ohnishi and K. Matsumoto. Gentzen method in modal calculi. *Osaka Mathematical Journal*, 9:113–130, 1957.

[138] M. Ohnishi and K. Matsumoto. Gentzen method in modal calculi, II. *Osaka Mathematical Journal*, 11:115–120, 1959.

[139] F. Oppacher and E. Suen. HARP: a tableau-based theorem prover. *J. of Automated Reasoning*, 4:69–100, 1988.

[140] P. Patel-Schneider. A four-valued semantics for terminological logic. *Artificial Intelligence*, 38:319–351, 1989.

[141] L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[142] L. Paulson. Proving properties of security protocols by induction. In *10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.

[143] J. Pitt and J. Cunningham. Theorem proving and model building with the calculus KE. *Bulletin of the Interest Group in Pure and Applied Logic*, 4(3):129–150, 1996.

[144] V. Pratt. A near-optimal method for reasoning about action. *J. of Computer and System Sciences*, 20:231–255, 1980.

[145] V. Pratt. A practical decision method for propositional dynamic logic. In *Proc. of the 10th ACM Symposium on Theory of Computing (STOC-78)*, pages 326–337, 1978.

[146] V. Pratt. Models of program logics. In *Proc. of the 20th Annual Symposium on the Foundations of Computer Science (FOCS-79)*, pages 115–122. IEEE Computer Society Press, 1979.

[147] Veikko Rantala. Impossible worlds semantics and logical omniscience. *Acta Philosophica Fennica*, 35:106–115, 1982.

[148] A. Robinson. A machine oriented logic based on the resolution principle. *J. of the ACM*, 12(1):23–41, 1965.

[149] J. Rushby. Automated deduction and formal methods. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of the International Conf. on Computer-Aided Verification (CAV-96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 1996.

[150] S. Russel and S. Zilberstein. Composing real-time systems. In *Proc. of the 12th International Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 212–217, 1991.

[151] A. Russo. Generalising propositional modal logic using labelled deductive systems. In *Proceeding of the International Workshop on Frontiers of Combining Systems (FroCoS-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.

[152] R. Sandhu. The typed access matrix model. In *Proc. of the 11th IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Computer Society Press, 1992.

[153] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, pages 40–48, September 1994.

[154] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access controls models. *IEEE Computer*, 29(2), February 1996.

[155] M. Schaerf and M. Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:1–62, 1995.

[156] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th International Joint Conf. on Artificial Intelligence (IJCAI-91)*, pages 466–471. Morgan Kaufmann, Los Altos, 1991.

[157] R. Schmidt. Resolution is a decision procedure for many propositional modal logics. In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic*. CSLI Publications, 1997. Extended version as Technical Report MPI-I-97-2-002, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1997.

[158] M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proc. of the 1st International Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)*, pages 421–431, 1989.

[159] S. Schmitt and C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. In Michael A. McRobbie and John K. Slaney, editors, *Proc. of the 13th International Conf. on Automated Deduction (CADE-96)*, volume 1104, pages 418–432. Springer-Verlag, 1996.

[160] G. Schwarz. Gentzen style systems for K45 and KD45. In A. Meyer and M. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1989.

[161] B. Selman and H. Kautz. Knowlege compilation and theory approximation. *J. of the ACM*, 43(2):193–224, 1996.

[162] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the 10th National Conf. on Artificial Intelligence (AAAI-92)*, 1992.

[163] R. Smullyan. *First Order Logic*. Springer-Verlag, 1968. Republished by Dover, New York, in 1995.

[164] R. Smullyan. Uniform gentzen systems. *J. of Symbolic Logic*, 33(4):549–559, 1968.

[165] C. Stirling. Modal and temporal logics for processes. In Moller and Birtwistle [130], pages 149–237.

[166] C. Stirling and D. Walker. Local model checking in modal mu-calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[167] R. Streett and E. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Control*, 81:249–264, 1989.

[168] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proc. of the 10th IEEE Symposium on Security and Privacy*, pages 156–170. IEEE Computer Society Press, 1991.

[169] P. Syverson and P. van Oorschot. On unifying some cryptographic protocols logics. In *Proc. of the 13th IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1994.

[170] T. Uribe and M. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In J. P. Jouannaud, editor, *Proc. of the 1st International Conf. on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[171] A. Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, Dec 1995.

[172] J. van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II. Reidel, 1986.

[173] R. van der Meyden. The dynamic logic of permission. *J. of Logic and Computation*, 6(3):465–479, 1996.

[174] M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986.

[175] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[176] A. Voronkov. The anatomy of vampire (implementing bottom-up procedures with code trees). *J. of Automated Reasoning*, 15(2):237–265, 1995.

[177] K. Wallace and G. Wrightson. Regressive merging in model elimination tableau-based theorem provers. *Bulletin of the Interest Group in Pure and Applied Logic*, 3(6):921–937, 1995.

[178] L. Wallen. *Automated Deduction in Nonclassical Logics*. The MIT Press, 1990.

[179] H. Wang. Towards mechanical mathematics. *IBM J. for Research and Development*, 4:2–22, 1960.

[180] E. Wobber, M. Abadi, and M. Burrows. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, 1994.

[181] P. Wolper. The tableaux method for temporal logics. *Logique et Analyse*, 110-111:119–136, 1985.

[182] W. Woods and J. Schmolze. The KL-ONE family. In *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992.

[183] M. Ying. A logic for approximate reasoning. *J. of Symbolic Logic*, 59(3):830–837, 1994.

[184] H. Zhang and M. Stickel. An efficient algorithm for unit-propagation. In *Proc. of the 4th International Symposium on Artificial Intelligence and Mathematics*, 1996.

Appendix A

Proofs of Part I

A.1 Proofs of Chapter 3

Proof of THEOREM 3.9 (page 54).

Proof. The key point is that if n_ψ is the number of ν subformulae and p_ψ the number of π subformulae in Subfml(ψ) of the (un)simplified formula and $\texttt{poly}(\psi, \varphi)$ the (polynomial) cost of propositional simplification (ignoring modal operators), then the worst case of the whole operation of K-simplification is polynomial in $O((p_\psi + n_\psi) \times \texttt{poly}(\psi, \varphi))$. $\square$

Proof of LEMMA 3.13 (page 58).

Proof. The case for (K), (D) and (T) is immediate.

Rule (4) is used for transitive models. Assume that $w \Vdash \nu(a)$ and $w\mathcal{R}_a v$ then, for any $u \in W$ such that $v\mathcal{R}_a u$ it is $w\mathcal{R}_a u$, by transitivity. Hence $u \Vdash \nu_0$. Since u is arbitrary, also $v \Vdash \nu(a)A$. For rule (4^π) suppose that $w\mathcal{R}_a wv$ and $v \Vdash \pi(a)$. The semantics forces the existence of a world u such that $v\mathcal{R}_a u$ and $u \Vdash \pi_0$. By transitivity $w\mathcal{R}_a u$ and therefore $w \Vdash \pi(a)$.

Rule (4^R) is used for euclidean relations. Suppose that $w \Vdash \nu(a)$ and $w\mathcal{R}_a v$. For any u such that $w\mathcal{R}_a u$, it is $v\mathcal{R}_a u$, since $\mathcal{R}_a$ is euclidean. Thus if $v \Vdash \nu(a)$ it is $u \Vdash \nu_0$ and then $w \Vdash \nu(a)$.

For rule (B), assume that $w\mathcal{R}_a v$ and $v \Vdash \nu(a)$. By symmetry it is $v\mathcal{R}_a w$ and therefore $w \Vdash \nu_0$.

Rule (Cxt)can be used for euclidean and contextual models. Suppose the model is euclidean and that $w\mathcal{R}_a v$ with $v \Vdash \nu(a)$. Then let $w\mathcal{R}_a u$ and $u\mathcal{R}_a t$. The properties of $\mathcal{R}_a$ forces $u\mathcal{R}_a v$ and then $v\mathcal{R}_a t$. Thus $t \Vdash A$. Since t is arbitrary, it is $u \Vdash \nu(a)$ and $w \Vdash [a]\, \nu(a)$. If the L-model is a contextual model (see [21] or Table 3.2), then we have directly $v\mathcal{R}_a t$ and the same reasoning applies.

Delayed rules require the same arguments with the condition $w_0\mathcal{R}_a w$. $\square$

Proof of THEOREM 3.16 (page 59).

Proof. If $\mathcal{T}$ is L-SAT, there is a branch $\mathcal{B}$ which is L-SAT with an SST-interpretation $\imath()$. The cases where $\mathcal{T}'$ has been obtained by applying a rule to a branch different from $\mathcal{B}$, or the rule applied is an α, β, *Loc* or *Glob* rule follow exactly [62].

The different cases are ν-rules and the π-rule. So let $\langle W, \{ \mathcal{R}_a \}_{a \in \mathcal{A}}, \mathcal{I} \rangle$ be the model for which $\mathcal{B}$ is L-SAT.

For SST-rules characterizing L, consider whether they are static, backward or forward. If a *static rule* has been applied then the prefix σ is present in $\mathcal{B}$ and $\imath()$ is defined on it. By inductive hypothesis $\imath(\sigma) \Vdash \varphi$ and by Lemma 3.13 we are done. *Forward and backward rules* involve prefixes of the form σ and $\sigma.a.n$ both already present on the branch. By Def. 3.14 it is $\imath(\sigma)\mathcal{R}_a\imath(\sigma n)$ and the claim follows by Lemma 3.13.

Delayed rules require an "ancestor" world w_0: by Theorem 3.12, if $\sigma.a.n$ and $\sigma.a.n.a.m$ are in the branch then σ is there too. Hence $\imath()$ is defined on all three prefixes and by inductive hypothesis $\imath(\sigma)\mathcal{R}_a\imath(\sigma.a.n)$ and $\imath(\sigma.a.n)\mathcal{R}_a\imath(\sigma.a.n.a.m)$. Apply Lemma 3.13.

The π-rule introduces a new prefix, and $\imath()$ must be extended. By hypothesis the original branch $\mathcal{B}$ is L-SAT and thus $\imath(\sigma)\pi(a)$. The semantics forces the existence of a world $w \in W$ such that $\imath(\sigma)\mathcal{R}_a w$ and $w \Vdash \pi_0$. Then extend $\imath()$ as follow:

$$\jmath(s) \doteq \textbf{if } s = \sigma.a.n \textbf{ then } w \textbf{ else } \imath(s)$$

We must prove that $\jmath()$ is an SST-interpretation. On every prefix s different from $\sigma.a.n$ the mapping $\jmath(s)$ coincides with $\imath(s)$ which is an SST-interpretation. Moreover $\jmath(\sigma)\mathcal{R}_a\jmath(\sigma.a.n)$ by construction. Since $\sigma.a.n$ is new, there is no prefix $\sigma.a.n.a.m$ in the branch which requires us to prove $\jmath(\sigma.a.n)\mathcal{R}_a\jmath(\sigma.a.n.a.m)$. The branch is L-SAT on the same L-model with SST-interpretation $\jmath()$. □

The following proofs for Theorem 3.18 are given for the single agent case to simplify the notation. Since we have no interconnection axiom between modalities, the proofs lift immediately to the multi agent case.

Proof of THEOREM 3.18 (page 60).

[Cxt] *Proof.* For the logic of contextual reasoning we must prove the following property [21]: "if $\sigma \rhd \sigma_a$ and $\sigma \rhd \sigma_b$ and $\sigma_a \rhd \sigma_c$, then $\sigma_b \rhd \sigma_c$". The conditions on $\rhd$ for Cxt are the following:

$$\begin{array}{lll} \sigma \rhd \sigma_a & \text{iff} & |\sigma_a| = |\sigma| + 1 \\ \sigma \rhd \sigma_b & \text{iff} & |\sigma_b| = |\sigma| + 1 \\ \sigma_a \rhd \sigma_c & \text{iff} & |\sigma_c| = |\sigma_a| + 1 \end{array}$$

by simple substitutions it is $|\sigma_c| = |\sigma_a| + 1 = |\sigma| + 2 = |\sigma_b| + 1$. □

Proof of THEOREM 3.18 (page 60).

[K4] *Proof.* Transitivity equals "$\sigma_a \rhd \sigma_b$ and $\sigma_b \rhd \sigma_c$ implies $\sigma_a \rhd \sigma_c$". The two initial conditions impose the properties below on $\rhd$ (Table 3.5):

$$\begin{array}{llll} \sigma_a \rhd \sigma_b & \text{iff} & \sigma_b = \sigma_a.\sigma' & \text{with } |\sigma'| \geq 1 \\ \sigma_b \rhd \sigma_c & \text{iff} & \sigma_c = \sigma_b.\sigma'' & \text{with } |\sigma''| \geq 1 \end{array}$$

By a simple concatenation we have that $\sigma_c = \sigma_a.\sigma'.\sigma''$ and $|\sigma'.\sigma''| \geq 1$. □

Proof of THEOREM 3.18 (page 60).

[K5] *Proof.* An euclidean accessibility relation is characterized by the property "$\sigma_a \rhd \sigma_b$ and $\sigma_a \rhd \sigma_c$ implies $\sigma_b \rhd \sigma_c$". We have the following cases to combine according the condition on prefixes for K5:

1. $\sigma_a \rhd \sigma_b$ if $\sigma_a = \sigma$ and $\sigma_b = \sigma.n$,
2. $\sigma_a \rhd \sigma_c$ if $\sigma_a = \sigma'$ and $\sigma_c = \sigma'.n'$
3. $\sigma_a \rhd \sigma_b$ if $\sigma_a = \sigma.\sigma'$ and $\sigma_b = \sigma.\sigma''$ with $|\sigma'| \geq 1$ and $|\sigma''| \geq 1$,
4. $\sigma_a \rhd \sigma_c$ if $\sigma_a = \sigma_1.\sigma_1'$ and $\sigma_c = \sigma_1.\sigma_1''$ with $|\sigma_1'| \geq 1$ and $|\sigma_1''| \geq 1$.

Combining the first and the second cas we have that $\sigma_b = \sigma.n$ and $\sigma_c = \sigma'.n'$. Then we know that both σ and σ' must be longer than 1. So, both have a a common initial prefix σ_0 which is then extended by at least n (for σ_b) and n' (for σ_c). Then $\sigma_b \rhd \sigma_c$ according Table 3.5.

Combining the third and the fourth case, the reasoning is identical. By definition both σ_1 and σ must be longer than 1. Since both σ and σ_1 are initial parts of the same prefix σ_a then we can only have three cases: $\sigma = \sigma_1$ or $\sigma_1 = \sigma.\tau$ or $\sigma = \sigma_1.\tau$ for some prefix τ such that $|\tau| \geq 1$.

$\sigma = \sigma_1$: this condition implies $\sigma_b = \sigma.\sigma''$ and $\sigma_c = \sigma.\sigma_1$. Since $|\sigma''| \geq 1$ and $|\sigma_1''| \geq 1$ we have directly $\sigma_b \rhd \sigma_c$;

$\sigma_1 = \sigma.\tau$: we substitute σ_1 in the equation of σ_c and obtain $\sigma_b = \sigma.\sigma''$ together with $\sigma_c = \sigma.\tau.\sigma_1$. Since $|\sigma''| \geq 1$ and $|\tau.\sigma_1| \geq 1$, we obtain $\sigma_b \rhd \sigma_c$;

The remaining case is similar. □

Proof of LEMMA 3.19 (page 62).

Proof. If the syntactic accessibility relation $\rhd$ accesses only *close neighbors* the proof is trivial: complete reduction coincides with SST reduction. For instance, for the logic B there are three cases: $\sigma \rhd_{\mathsf{B}} \sigma$, $\sigma \rhd_{\mathsf{B}} \sigma.a.n$ and $\sigma.a.n \rhd_{\mathsf{B}} \sigma$. Apply respectively SST reduction for rule (T), (K), and (B).

If $\rhd$ connects *forward prefixes*, then $\sigma \rhd \sigma^*$ implies that σ^* has the form $\sigma.(a.n_i)^k$ for $k \geq 1$ (or $k \geq 0$ if the logic is reflexive). By Theorem 3.12, for every $j = 1 \ldots k$ also $\sigma.(a.n_i)^j$ is present. Apply reduction w.r.t.(4) or (4^D) to construct the chain $\sigma : \nu(a) \in \mathcal{B}$ implies $\sigma.a.n_1 : \nu(a) \in \mathcal{B}$, implies $\sigma.a.n_1.a.n_2 : \nu(a) \in \mathcal{B}$ etc. Once arrived at $\sigma.(a.n_i)^{k-1} : \nu(a)A \in \mathcal{B}$ apply SST reduction w.r.t.(K) to obtain $\sigma^* : \nu_0 \in \mathcal{B}$.

When *backward prefixes* are connected by $\rhd$, then $\sigma \rhd \sigma^*$ implies that for some prefix σ_0 we have $\sigma = \sigma_0.(a.n_i)^k$ and $\sigma^* = \sigma_0.a.m$ (or $\sigma^* = \sigma_0$ if the logic is reflexive). By Theorem 3.12, for every $j = 1 \ldots k$ also $\sigma_0.(a.n_i)^j$ is in $\mathcal{B}$. For every j, if $\sigma_0.(a.n_i)^{j+1} : \nu(a) \in \mathcal{B}$ then $\sigma_0.(a.n_i)^j : \nu(a) \in \mathcal{B}$, because $\sigma_0.(a.n_i)^{j+1} : \nu(a) \in \mathcal{B}$ is reduced for rule (4^R) by hypothesis. Therefore, $\sigma_0 : \nu(a)$ is present in $\mathcal{B}$. Then we apply the reduction for rule (K) and obtain $\sigma_0.a.m : \nu_0 \in \mathcal{B}$.

For K5 we combine these technique: we apply reduction for rule (4^R) until we can conclude that $1.a.n : \nu(a)$ is present. If $\sigma^* = 1.a.m$ then we use reduction for rule (4^R) followed by (K). Otherwise, first (Cxt) obtaining $1 : \nu^2(a)$, and then (K). From $1.a.m : \nu(a) \in \mathcal{B}$, the reasoning is identical to that for syntactical relations $\rhd$ connecting forward prefixes: first we use reduction for rule (4^D) and last reduction for rule (K).

For the logic Cxt apply reduction for rule (Cxt) until we have shown that $1 : \mathtt{t}.[a]^n A \in \mathcal{B}$, then apply reduction for (K) up to σ^*.

The branches of the calculi $\mathsf{K45}$, $\mathsf{KD45}$, $\mathsf{S5}$ with rule (4^π) have prefixes of the form 1 and $1.n$, since we restricted the application of the π-rule. For moving $\nu(a)$ we apply (4^R) once and then either (K) or (T). □

Proof of THEOREM 3.20 (page 62).

Proof. Construct the model as follows:

$$\begin{array}{rcl} W & \doteq & \{\langle\sigma\rangle \ : \ \sigma \text{ is present in } \mathcal{B}\} \\ \langle\sigma\rangle\, \mathcal{R}\, \langle\sigma^*\rangle & \text{iff} & \sigma \rhd_a \sigma^* \\ (p)^{\mathcal{I}} & \doteq & \{\sigma \mid \sigma : \mathtt{t}.p \in \mathcal{B}\} \end{array}$$

If L is serial, extend $\mathcal{R}_a$ by setting $\langle\sigma\rangle\, \mathcal{R}_a\, \langle\sigma\rangle$ if there are no formula of the form $\sigma : \pi(a)$ in $\mathcal{B}$. This guarantees that $\sigma : \nu(a) \notin \mathcal{B}$. Otherwise, since σ is reduced for (D) we would have had $\sigma : \mathtt{f}.[a]\perp \in \mathcal{B}$, contradiction.

By Theorem 3.18, the relation $\mathcal{R}_a$ satisfies the properties of L. We must show that $\mathcal{B}$ is L-SAT on model $\langle W, \{\mathcal{R}_a\}_{a\in\mathcal{A}}, \mathcal{I}\rangle$ with an SST-interpretation $\imath()$. In this case the interpretation is the identity function.

For every φ we can now prove that if $\sigma : \varphi \in \mathcal{B}$ then $\langle\sigma\rangle \Vdash \varphi$ by induction on the construction of φ. We focus on modal connectives.

If $\sigma : \pi(a) \in \mathcal{B}$ we have $\sigma.a.n : \pi_0 \in \mathcal{B}$ for some $\sigma.a.n$, since $\sigma : \pi(a)$ must be reduced for rule (π). By inductive hypothesis $\sigma.a.n \Vdash \pi_0$ and $\langle\sigma\rangle\, \mathcal{R}_a\, \langle\sigma.a.n\rangle$ by construction. Therefore $\langle\sigma\rangle \Vdash \pi(a)$. If $\sigma : \nu(a) \in \mathcal{B}$ then, by Lemma 3.19, for every σ^* present in $\mathcal{B}$ such that $\sigma \rhd_a \sigma^*$, it is $\sigma^* : A \in \mathcal{B}$. By inductive hypothesis $\langle\sigma^*\rangle \Vdash A$ and $\langle\sigma\rangle\, \mathcal{R}_a\, \langle\sigma^*\rangle$ by construction. Therefore $\langle\sigma\rangle \Vdash \nu(a)$.

For every $A \in G$ and every σ in $\mathcal{B}$, the prefixed formula $\sigma : \mathtt{t}.A$ is present in the branch $\mathcal{B}$ because σ is reduced for the $(Glob)$-rule in $\mathcal{B}$. Then $\langle\sigma\rangle \Vdash \mathtt{t}.A$. With the same argument, if $A \in U$ then $\langle 1\rangle \Vdash \mathtt{t}.A$. □

A.2 Proofs of Chapter 4

Proof of THEOREM 4.4 (page 66).

Proof. Query combination is not trivial only in the following cases:

1. the formula A is valid for L_{fix} and $\mathsf{L}_{var} \subset \mathsf{L}_{fix}$;
2. the formula A is not valid for L_{fix} and $\mathsf{L}_{fix} \supset \mathsf{L}_{var}$.

In the first case use the formula $A[B] = A_{fix} \vee \neg B$ where A_{fix} is an instance of the axiom schema which distinguishes L_{fix} from L_{var} (see Def. 4.2). The variables of B must be distinct from the variables of A_{fix}. The answer of the PSPACE-box is always "valid" and the PTIME box is "not valid" iff B is satisfiable for PC.

In the second case use $A[B] = A_{var} \wedge \neg B$ which yields the constant answer "not valid" from the PSPACE box.

If the output of the PSPACE-box is constant , whatever its input, then we can drop the box altogether and feed directly the output into the PTIME-box. In this way we have a polynomial time machine for PC-satisfiability. □

Reduction Precondition	Reduction Relation
$\sigma : \alpha \in \mathcal{B}$	$\mathcal{B} \xrightarrow{\alpha} \mathcal{B} \cup \{\sigma : \alpha_1, \sigma : \alpha_2\}$
$\sigma : \beta \in \mathcal{B}$	$\mathcal{B} \xrightarrow{\beta} \mathcal{B} \cup \{\sigma : \beta_1\} \vert \mathcal{B} \cup \{\sigma : \beta_2\}$
$\sigma : \nu(a) \in \mathcal{B}$ and $\exists\varphi,\ \sigma.a.n : \varphi \in \mathcal{B}$	$\mathcal{B} \xrightarrow{K} \mathcal{B} \cup \{\sigma.a.n : \nu_0\}$
$\sigma : \pi(a) \in \mathcal{B}$ and $\forall\psi,\ \sigma.a.n : \psi \notin \mathcal{B}$	$\mathcal{B} \xrightarrow{\pi} \mathcal{B} \cup \{\sigma.a.n : \pi_0\}$

Table A.1: Reduction Rules for SST

Proof of THEOREM 4.9 (page 69).

Proof. . We show a stronger claim, namely

$$\forall x, x'yz \text{ if } x \sim x' \ \&\ x \to y \ \&\ x' \to z \text{ then } \exists u, u_e\ y \to u \ \&\ z \to u_e \ \&\ u \sim u_e$$

In words, "if we apply a rule to the tableau x yielding y or another rule yielding z, then we can apply another rule to y to obtain u and similarly to z obtaining u_e". The two possible results u and u_e must be identical up to renaming of prefixes.

The standard Knuth-Bendix method for proving confluence [88, page 797] is to prove that each critical pair satisfies the above mentioned claim. For SST critical pair can only be formed when we reduce two formulae in the same branch, since reductions in different branches do not interact[1].

For clarity of exposition, we consider first the basic logic K and the case where $x = x'$. This is also the most difficult case.

The cases of superpositions can be seen more easily if we reformulate the rules as in Table. A.1, where $\mathcal{B}$ is a branch (set of prefixed formulae).

A first observation is that formulae with different prefixes do not interact.

Proposition A.1 *Let $\sigma_1 : \varphi_1$ and $\sigma_2 : \varphi_2$ be prefixed formulae with $\sigma_1 \neq \sigma_2$ and let $\mathcal{B}'$ be the reduction of $\mathcal{B}$ using rule (r_1) on $\sigma_1 : \varphi_1$. If rule (r_2) can be applied to $\sigma_2 : \varphi_2$ in $\mathcal{B}$, then it can also be applied in $\mathcal{B}'$.*

This reduces the cases of superposition to the following four:

1. the prefixed formulae $\sigma : \nu(a)$, $\sigma.a.n : \varphi$ and $\sigma.a.m : \psi$ are in $\mathcal{B}$;
2. the prefixed formulae $\sigma : \nu(a)$, $\sigma : \nu(a)'$ and $\sigma.n : \varphi$ are in $\mathcal{B}$;
3. the prefixed formulae $\sigma : \nu(a)$, $\sigma : \pi(a)$ and $\sigma.a.n : \varphi$ are in $\mathcal{B}$ but no prefixed formula $\sigma.a.m : \psi$ is present in $\mathcal{B}$ ($\sigma.a.m$ is new);
4. the prefixed formula $\sigma : \pi(a)$ and $\sigma : \pi(a)'$ are in $\mathcal{B}$ and the prefixes $\sigma.n$ and $\sigma.m$ are new.

[1]This is quite different from free variables tableaux where variables span over branches and therefore the reduction (unification) in a branch affects other branches.

In the first case we apply rule (K) either to $\sigma.a.n$ or for $\sigma.a.m$. Suppose we apply first (K) to $\sigma.a.n$ yielding $\mathcal{B} \rightarrow \mathcal{B} \cup \{\sigma.a.n : \nu_0\}$. Since $\sigma.a.m : \varphi$ is still present in the new branch and ν formulae can be reused, we apply again (K) obtaining $\mathcal{B} \cup \{\sigma.a.n : \nu_0, \sigma.a.m : \nu_0\}$. If we apply (K) first to $\sigma.a.m$ and second to $\sigma.a.n$ we obtain $\mathcal{B} \cup \{\sigma.a.m : \nu_0, \sigma.a.n : \nu_0\}$. The final outcome is the same.

For case (2), use the same argument. The final outcome of both reduction paths is equal to $\mathcal{B} \cup \{\sigma.a.n : \nu_0, \sigma.a.n : \nu_0'\}$.

For (3) we can use rule (K) on $\sigma.a.n$ or introduce a new prefix with (π). We can change the set of new prefixes. If we reduce the branch using rule (K), we do not introduce new prefixes and $\sigma.a.m$ would still be new in $\mathcal{B} \cup \{\sigma.a.n : \nu_0\}$. In other words, if for no φ we have $\sigma.a.m : \varphi \in \mathcal{B}$, then clearly $\sigma.a.m : \varphi \notin \mathcal{B} \cup \{\sigma.a.n : \nu_0\}$. Therefore we can use rule (π) and obtain $\mathcal{B} \cup \{\sigma.a.n : \nu_0, \sigma.a.m : \pi_0\}$.

If we apply (π) first, we obtain $\mathcal{B} \cup \{\sigma.a.m : \pi_0\}$. By hypothesis, for all ψ we have $\sigma.a.m : \psi \notin \mathcal{B}$. Since $\sigma.a.n : \varphi \in \mathcal{B}$, it is $\sigma.a.m \neq \sigma.a.n$. We can now apply rule (K) obtaining the branch $\mathcal{B} \cup \{\sigma.a.n : \nu_0, \sigma.a.m : \pi_0\}$. Again the two final outcomes are identical and we are done.

In the last case we must use a renaming to prove confluence. Suppose we reduce first $\sigma : \pi(a)$ and obtain $\mathcal{B} \cup \{\sigma.a.n_1 : \pi_0\}$. In the new branch, the prefix $\sigma.a.n_1$ is not anymore new. So the next reduction forces the use of $\sigma.a.m_1$. The final result is the branch $\mathcal{B}_1 = \mathcal{B} \cup \{\sigma.n_1 : \pi_0\ \sigma.m_1 : \pi_0'\}$.

If we reduce $\sigma : \pi(a)'$ with $\sigma.a.m_2$ we obtain the branch $\mathcal{B} \cup \{\sigma.a.m_2 : \pi_0'\}$. A further (π)-reduction yields the branch $\mathcal{B}_2 = \mathcal{B} \cup \{\sigma.a.m_2 : \pi_0',\ \sigma.n_2 : \pi_0\}$. At this stage we only know that there are two new prefixes $\sigma.a.n$ and $\sigma.a.m$. It can well be that $n_1 = n \neq m = n_2$ and $\pi_0 \neq \pi_0'$. Therefore the branches will be different: $\mathcal{B}_1 \neq \mathcal{B}_2$.

Then we define two renamings h_{12} and h_{21}:

$$h_{ij}(s) \quad = \quad \begin{cases} \sigma.a.n_j & \text{if } s = \sigma.a.n_i \\ \sigma.a.m_j & \text{if } s = \sigma.a.m_i \\ s & \text{otherwise} \end{cases}$$

Now we can prove that $\sigma : \varphi \in \mathcal{B}_i$ implies $h_{ij}(\sigma) : \varphi \in \mathcal{B}_j$, and that $h_{ji}(h_{ij}(\sigma)) = \sigma$.

The proof for other logics is a repetition of arguments 1, 2 and 3 for the various ν-rules. We must replace the prefixed formulae $\sigma : \nu(a)$ and $\sigma.a.n : \nu_0$ of rule (K) with the antecedent and the consequent of each ν-rule. We must be able to reduce the same ν-formula more than once, because we have more than one ν-rule and we must be able to apply all rules.

The proof of the general case $x \sim x'$ follows the same pattern and it is simply notationally heavy, since we have two equivalent branches $\mathcal{B}_1$ and $\mathcal{B}_2$ and two renamings h_{12} (from $\mathcal{B}_1$ to $\mathcal{B}_2$) and h_{21} (in the other direction).

The first step is the following:

Proposition A.2 *A rule (r) can be applied to $\sigma : \varphi$ in $\mathcal{B}_i$ if and only if it can be applied on $h_{ij}(\sigma) : \varphi$ in $\mathcal{B}_j$.*

Second, we reformulate proposition A.1 as follows:

Proposition A.3 *Let $\sigma_1 : \varphi \in \mathcal{B}_1$ and $\sigma_2 : \psi \in \mathcal{B}_2$ be prefixed formulae such that $h_{12}(\sigma_1) \neq \sigma_2$. Let $\mathcal{B}_1^r$ be the reduction of $\mathcal{B}_1$ using rule (r) on $\sigma_1 : \varphi$ and $\mathcal{B}_2^r$ be the* *reduction of $\mathcal{B}_2$ using (r) on $h_{12}(\sigma_1) : \varphi$. If rule (r_2) can be applied to $\sigma_2 : \psi$ in $\mathcal{B}_2$, then it can also be applied in $\mathcal{B}_2^r$.*

The cases of superposition can be reformulated along the same lines. The proof is substantially unchanged. Case (4) is the only difficult one.

In $\mathcal{B}_1$ we have the prefixed formulae $\sigma_1 : \pi(a)$, $\sigma_1 : \pi(a)'$. The prefixes $\sigma_1.a.n_1$ and $\sigma_1.a.m_1$ are new. The same conditions (changing subscript) hold for $\mathcal{B}_2$. We have $h_{ij}(\sigma_i) = \sigma_j$. We have no constraints on $\sigma_i.a.n_i$ and $\sigma_i.a.m_i$ because they are new.

As in the proof for $x = x'$, renamings must be updated when the branch $\mathcal{B}_1$ reduces to $\mathcal{B}_1 \cup \{ \sigma_1.a.n_1 : \pi_0, \sigma_1.a.m_1 : \pi_0' \}$ and the branch $\mathcal{B}_2$ reduces to the branch $\mathcal{B}_2 \cup \{ \sigma_2.a.n_2 : \pi_0, \sigma_2.a.m_2 : \pi_0' \}$.

$$h'_{ij}(s) = \begin{cases} h_{ij}(\sigma_i).a.n_j & \text{if } s = \sigma_i.a.n_i \\ h_{ij}(\sigma_i).a.m_j & \text{if } s = \sigma_i.a.m_i \\ h_{ij}(s) & \text{otherwise} \end{cases}$$

By hypothesis, it is $h_{ij}(\sigma_i) = \sigma_j$ and the mappings h'_{ij} give the desired renamings. □

Proof of THEOREM 4.15 (page 75).

Proof. If a tableau proof is found, the correctness theorem 3.17 does the job.

If some rules are still applicable, then the strategy itself was not systematic or possibly incomplete. In this case, continue the reduction with any systematic strategy. By proof confluence (Thm. 4.9) we must end up in one of the remaining cases.

Finally, if the strategy terminates and no rule is applicable in at least a branch $\mathcal{B}$, then we must prove that $\mathcal{B}$ can be pruned into a π/completed branch.

At first, a π-formula $\sigma : \pi$ which may not be reduced according Def. 4.11 only if $|\sigma| = hb_{\mathsf{L}}$. Indeed suppose that $\sigma : \pi$ is not reduced but there is no shorter copy σ_0 where it is reduced. If $|\sigma| < hb_{\mathsf{L}}$ then rule π would be still applicable according technique T.4.4 and this contradicts the assumption that no rule is applicable in $\mathcal{B}$.

The case for logics K, D, T, KB, KDB, and B is simple: every rule that increases the length of the prefix strictly reduces the number of modal connectives. So any prefix with length equal to $hb_{\mathsf{L}} = 1 + d$ has no modal connectives. We obtain directly a π/completed branch. □

Proof of THEOREM 4.16 (page 75).

Proof. We repeat the argument used for Thm. 4.15 until we have proved that the only unreduced prefixed formulae $\sigma : \pi(a)$ are those with $|\sigma| = hb_{\mathsf{L}}$.

The problem with $\mathsf{K4}$ and its extensions is that rule (4) increases the length of the prefix without decreasing the number of modal connectives. However, we can observe that the number of ν-formulae increases monotonically with the length of the prefix:

Proposition A.4 *If the prefix σ_0 is an initial subsequence of σ in the branch $\mathcal{B}$ then $\sigma_0 : \nu \in \mathcal{B}$ implies $\sigma : \nu \in \mathcal{B}$.*

If $\sigma : \nu$ is not present, then we could apply rule (4) until $\sigma : \nu$ is introduced. This is against the hypothesis that no rule is applicable to branch $\mathcal{B}$.

Now, we start pruning the tree: delete from the branch all possible subtrees Ftree($\sigma.n$) such that for every $\sigma : \pi$ which is fulfilled by $\sigma.n$ there is a shorter modal copy σ_0 where $\sigma_0 : \pi$ is reduced for rule (π).

Denote by $\mathcal{B}'$ the final set of prefixed formulae which cannot be pruned anymore. It is easy to prove that proposition 4.17 is still verified in $\mathcal{B}'$, and that the pruning lemma 4.13 guarantees the following property:

Proposition A.5 *Let $\sigma : \pi$ be a prefixed formula in $\mathcal{B}'$ then either $|\sigma| = hb_{\mathsf{L}}$ or $\sigma : \pi$ is reduced for rule (π) in $\mathcal{B}'$ or there is a shorter modal copy σ_0 of σ such that $\sigma_0 : \pi$ is reduced for rule (π) in $\mathcal{B}'$.*

The only π-formulae which may violate the definition of π/modal-completeness are those with length hb_{L}. Since $\mathcal{B}'$ cannot be pruned anymore, we have

Proposition A.6 *Let $\sigma : \pi$ be a prefixed formula in $\mathcal{B}'$ which is not reduced for (π). For every σ_0 shorter than σ either σ_0 is not a modal copy of σ or $\sigma_0 : \pi$ is not present in $\mathcal{B}'$.*

Then we prove that the longest prefix σ on the $\mathcal{B}'$ which can satisfy proposition A.6 is long $hb_{\mathsf{L}} - 1 = 1 + d_p + p \times n$. We need the following result:

Proposition A.7 *If every prefix from σ_0, $\sigma_0.n_1$ up to $\sigma_0.n_1 \ldots n_k$ is a modal copy of σ_0 then each $\sigma_0 \ldots n_i$ fulfills a different π-formula $\pi(a)$ in $\mathcal{B}'$.*

Otherwise we apply the pruning lemma to $\sigma.n_1 \ldots n_i$ and delete the whole subtree generated by it, against the hypothesis that $\mathcal{B}'$ is the final result of the pruning.

Given propositions 4.17 and 4.18 the worst case longest sequence of prefixes we can build without violating proposition A.6 is shown below:

$$\underbrace{1 \rhd 1.n_1 \rhd \rhd 1.n_1 \ldots n_{p-1}}_{0\ \nu-\text{formulae}} \rhd 1 \ldots n_{p-1} n_p \rhd \underbrace{1 \ldots n_p.n_{p+1} \rhd \rhd 1 \ldots n_{2p-1}}_{1\ \nu-\text{formula}} \rhd$$

$$\rhd \underbrace{1 \ldots n_{p+p} \rhd \rhd 1 \ldots n_{p+2p}}_{2\ \nu-\text{formulae}} \rhd \quad \rhd \quad \rhd \underbrace{1 \ldots . n_{p+(n-1)p+1} \rhd \rhd 1 \ldots n_{p+np}}_{n\ \nu-\text{formulae}}$$

Consider the first sequence with zero ν formulae: they are clearly modal copies of each other. Each time we pass from $1 \ldots n_i$ to $1 \ldots n_i.n_{i+1}$ there must be a different π-formula which is fulfilled (Prop. 4.18). We have at most p different π-subformulae and we can arrive only till $1 \ldots n_p$ before triggering the pruning lemma. Suppose that no ν-formula is present in $1 \ldots n_p$. Then, for every $1 \ldots n_p : \pi$ there is a shorter modal copy $1 \ldots n_i$ where $1 \ldots n_i : \pi$ is reduced. By proposition A.6, we cannot have a longer prefix $1 \ldots n_p.n_{p+1}$ in $\mathcal{B}'$.

In the worst case, only one ν formula will be introduced ex novo in $1 \ldots n_p$. This formula will continue to be present from now on. Again, before finding another ν-formula we can arrive at most to $1 \ldots . n_{p+p}$.

We continue this reasoning until we arrive at $1 \ldots n_{p+np}$. This prefix is the longest prefix that respects proposition A.6 (the length is $1+p+n\times p$ because of the initial 1). If we add a new prefix it will have the same modal formulae of $1 \ldots n_{p+(n-1)p+1}$ and every π-formula will be already reduced in one of the prefixes between $1 \ldots n_{p+(n-1)p+1}$ and $\sigma_{1+(n+1)p}$, thus triggering the pruning lemma.

Therefore, for every prefix σ_{hb} of length $1 + p + n \times p + 1$, and every unreduced π-formula $\sigma_{hb} : \pi$, there must be a shorter modal copy σ_0 such that $\sigma_0 : \pi$ is reduced. Hence the pruned branch $\mathcal{B}'$ is π modal completed.

We can improve the bound further by noting that the first part of the sequence, with zero ν-formulae, can only be generated by π formulae not under the scope of a necessity operator. In this case the number of connectives strictly decreases after each rule application. Therefore the first part can be bounded by d_p.

This yields the final upper bound of $hb_{\mathsf{L}} = 2 + d_p + n \times p$. □

Proof of COROLLARY 4.26 (page 79).

Proof. Suppose that n is bounded by a polynomial in the size of G, U and A, i.e. $n \leq poly(|G|, |U|, |A|)$.

Then observe that the modal deduction theorem is monotone in n: if (4.1) holds for some n, then it holds for all $m \geq n$. Indeed, $\models_{\mathsf{L}} U \cup \bigcup_{i=0}^{n} \Box^i G \Rightarrow A$ implies $\models_{\mathsf{L}} U \cup \bigcup_{i=0}^{n} \Box^i G \cup \bigcup_{i=n+1}^{m} \Box^i G \Rightarrow A$ by monotonicity of modal logic L.

For each G, U and A we use directly the upper bound $n_{max} = poly(|G|, |U|, |A|)$ and apply the deduction theorem with $n = n_{max}$. The generation of $\bigcup_{i=0}^{n_{max}} \Box^i G$ can be done in polynomial time in the value of n_{max} and the size of G. This would yields a polynomial time transformation of logical consequence into validity. □

Proof of LEMMA 4.29 (page 80).

Proof. Saturation for rules (4) and (4^R) forces all prefixes to have the same ν-formulae. Every prefix is a modal copy of all other prefix.

For the calculus with rule (4^π), simply notice that all prefixes are reduced wrt rule (4^R) and therefore the prefix 1 contains all ν formulae and all π-formulae. Since rule (4) is sound for $\mathsf{K45}$ and the other logics we simply apply rule (4) to all ν formulae in 1 and we are done. □

Proof of THEOREM 4.30 (page 80).

Proof. We can generate at most $p + p_G + 1$ different prefixes, which can be long at most $1 + p + p_G + 1$. Each prefix can have a number of formulae equal at most to the number of subformulae of G, U, and A and this number is linearly bounded by their respective sizes. The theorem follows with a simple multiplication. □

Proof of LEMMA 4.34 (page 84).

Proof. By a double induction on the length of the prefix σ and the length of the sequence. The case for prefix 1 is immediate.

For the first induction, suppose that $\sigma.n$ is confined but there is a sequence of rules that introduces a new prefixed formula $\sigma.n : \varphi$.

The sequence of length 1 can only be composed by a ν-rule. Then, there should be an unreduced $\sigma : \nu$ in $\mathcal{B}$. This formula is also in Btree($\sigma.n$) and therefore $\sigma.n$ could not be confined.

For longer sequences, we apply the induction hypothesis to σ: if σ is confined, no sequence of rules applied to prefixes different from σ can introduce a new prefixed $\sigma : \nu$ in $\mathcal{B}$. Only α and β rules are left and the corresponding formulae are confined by hypothesis. □

A.3 Proofs of Chapter 5

Proof of THEOREM 5.12 (page 103).

Proof. Suppose the contrary: $\mathcal{T}$ is SAT with all SAT branches ignorable (clearly SAT branches cannot be contradictory). It is worth noting that each branch can be ignorable due to a different unfulfilled eventuality $X \doteq \mathtt{t}.\langle P^*\rangle A$ (or even more than one). It is easy to prove the following proposition.

Proposition A.8 *Let $\mathcal{B}$ be an ignorable branch for some $\sigma : X$. For every model $\langle W, \mathcal{I}\rangle$ and for every mapping $\imath()$ such that $\mathcal{B}$ is SAT for them, if $\langle \sigma_i, X_i\rangle$ is in path(σ : X) then $\imath(\sigma_i) \not\Vdash A$.*

This is immediate because of the presence of $\mathtt{f}.A$ on the right-hand branch of the X-rule: it boils down to the definition of SAT branches.

Since $\mathcal{B}$ is SAT, there is a model $\langle W, \mathcal{I}\rangle$ and an mapping $\imath()$ on which $\mathcal{B}$ is SAT with a certain mapping $\imath()$. However, since $\mathcal{B}$ is ignorable, by Lem. 5.11, we can consider the final cluster generated by any prefixed formula $\sigma : \mathtt{t}.X$ for an unfulfilled X.

We can show that there are at least two P-connected prefixes in the final cluster on path($\sigma : X$). Indeed the only way to make a loop, is to have some X_j along the path collapsing into an X_i (maybe itself) and therefore the corresponding prefixes must be different. Let us denote the fully reduced prefix in the loop by $\sigma_0 \equiv \sigma_0'.\mathbf{a}.n_0$ and the copy with $\sigma_\omega \equiv \sigma_\omega'.\mathbf{a}.n_\omega$.

Since $\mathcal{B}$ is SAT, $\imath(\sigma_0) \models \mathtt{t}.\langle P\rangle X$, and therefore an integer N and N+1 states $w_0, \ldots, w_N$ in $\langle W, \mathcal{I}\rangle$ exist such that $\imath(\sigma_0) = w_0$, $\langle w_i, w_{i+1}\rangle \in (P)^{\mathcal{I}}$ and $w_N \Vdash \mathtt{t}.A$.

Since the prefix σ_0 has been fully reduced $\mathcal{B}$ must also contain $\langle \sigma_0 : \mathtt{t}.\langle P\rangle X\rangle$ and this must be reduced. Thus the only way to generate the next $\langle \sigma_1 : \mathtt{t}.X\rangle \succ \langle \sigma_0 : \mathtt{t}.X\rangle$ for going to σ_ω is to perform at least one P-step. Hence, in general, there is a number R of P-step from σ_0 to σ_1 to σ_2 ... to σ_ω for some integer $R \geq 1$

First assume that $N \leq R$. Then we can remap the prefixes $\sigma_0, \ldots, \sigma_N$ so that $\imath(\sigma_i) = w_i$. By Prop. A.8 the formula A cannot be fulfilled by any mapping $\imath()$ of the σ_i in path($\sigma : \mathtt{t}.X$) on the states of $\langle W, \mathcal{I}\rangle$. Hence we get a contradiction.

Then assume that $N > R$. Then there must be $N - R$ P-steps from σ_ω to fulfill A in the model $\langle W, \mathcal{I}\rangle$ under $\imath()$. Now we construct a new model $\langle W, \mathcal{J}\rangle$ as in Fig.5.7, by coping the original model: $W' = \{ w_c \mid w \in W \}$, $(p)^{\mathcal{J}} = \{ w_c \mid w \in (p)^{\mathcal{J}} \}$ and $(a)^{\mathcal{J}} = \{ \langle w_c, v_c\rangle \mid \langle w, v\rangle \in (a)^{\mathcal{J}} \}$. Then we add a new $\mathbf{a}$-arc, where $\mathbf{a}$ is the atomic (direct or converse) program occurring last in σ_0 and σ_ω, i.e.

$$(\mathbf{a})^{\mathcal{J}} \quad = \quad \{ \langle w_c, v_c\rangle \mid \langle w, v\rangle \in (\mathbf{a})^{\mathcal{I}} \} \cup \{ \langle \imath(\sigma_0)_c, \imath(\sigma_\omega)_c\rangle \}$$

The key point is to prove that this new $\mathbf{a}$-arc can be safely added.

Since $\mathcal{B}$ is π-completed, all possible instances of $LB(\mathbf{a})$ have been applied and therefore for every $\mathtt{t}.B \in$ Subfml(ψ) either $\langle \sigma_0 : \mathtt{f}.\langle \mathbf{a}^-\rangle B\rangle$ or $\langle \sigma_0 : \mathtt{t}.\langle \mathbf{a}^-\rangle B\rangle$ is present on the branch, where ψ is the formula we want to prove valid or satisfiable. The prefix σ_ω is a copy of σ_0 by hypothesis, so $\langle \sigma_\omega : \mathtt{f}.\langle \mathbf{a}^-\rangle B\rangle$ is present in the branch iff $\langle \sigma_0 : \mathtt{f}.\langle A^-\rangle B\rangle$ is present. Since the branch is SAT on the original model $\langle W, \mathcal{I}\rangle$, it is $\imath(\sigma_0) \Vdash \langle \mathbf{a}^-\rangle B$ iff $\imath(\sigma_\omega) \Vdash \langle \mathbf{a}^-\rangle B$ for every $\mathtt{t}.B \in$ Subfml(ψ).

Consider now the state $\imath(\sigma_\omega)_c$; the only difference with the original state $\imath(\sigma_\omega)$ is the incoming **a**-arc. But, as we have seen above, the two states see exactly the same formulae of Subfml(ψ) going back through **a**. By the filtration Lemma [60, 95], these are the only formulae necessary for establishing the truth value of ψ. Hence, by induction, we have that $\imath(\sigma_\omega)_c$ satisfies $\langle P^*\rangle A$ in $N-R$ P-steps in the new model (as in the old one).

Then we construct a new mapping $\jmath()$ on the new model as follows: map every prefix shorter or unrelated with σ_0 in the same way as $\imath()$ does, and map $\jmath(\sigma_0)$ on $\imath(\sigma_\omega)_c$. This make the branch still satisfiable: the formulae are the same for both σ_0 and σ_ω and the incoming arc does not affect them. By Thm. 5.10, we can expand the tableau and still preserve SAT.

In the new model, the state $\jmath(\sigma_0)$ fulfills the eventuality $\langle P^*\rangle A$ in $N-R<N$ P-steps. We can repeat the process until we reach an $N' \leq R$, contradiction. □

Proof of THEORE 5.14 (page 104).

Proof. Construct the model as follows:

$$\begin{array}{rcl} W & \doteq & \{\,\sigma \mid \sigma \text{ is present in } \mathcal{B}\,\} \\ (a)^{\mathcal{I}} & \doteq & \{\,\langle\sigma, \sigma.a.n\rangle \mid \sigma \text{ and } \sigma.a.n \text{ present in } \mathcal{B}\,\} \cup \\ & & \{\,\langle\sigma.a^-.n, \sigma\rangle \mid \sigma \text{ and } \sigma.a^-.n \text{ present in } \mathcal{B}\,\} \\ (p)^{\mathcal{I}} & \doteq & \{\,\sigma \mid \sigma : \mathtt{t}.p \in \mathcal{B}\,\} \end{array}$$

To take loops and repetitions into account, we modify slightly the above definition: if σ is a copy of some fully reduced prefix σ_0 then we delete σ from W, replace σ with σ_0 in all transitions $(a)^{\mathcal{I}}$, and construct the mapping $\imath()$.

$$\imath(\sigma) \quad = \quad \begin{cases} \sigma_0 & \text{if } \sigma \text{ is a copy of a fully reduced } \sigma_0 \\ \sigma & \text{otherwise} \end{cases}$$

In a similar way of the modal case (§3.6) we prove that if $\sigma : \varphi \in \mathcal{B}$ then $\imath(\sigma) \Vdash \varphi$ by induction on the construction of φ. We focus on modal and iteration operators.

Suppose that $\sigma : \pi \in \mathcal{B}$. If σ is fully reduced then we apply exactly the argument of Chapter 3. If σ is a copy then the mapping $\imath()$ maps σ on σ_0, which is fully reduced, and the above reasoning applies. Similarly for a^-.

For the necessity operator we show the case for a^-. Suppose that $\langle\sigma : \nu(a^-)\rangle$ is in $\mathcal{B}$ and that σ is fully reduced (the argument for not fully reduced prefixes is analogous to the previous case). By construction the pairs such that $\langle\sigma, \sigma'\rangle \in (a^-)^{\mathcal{I}}$ are:

1. the pairs $\langle\sigma, \sigma.a^-.n\rangle$ for some n;
2. the pair $\langle\sigma_0.a.n, \sigma_0\rangle$ if σ has the form $\sigma_0.a.m$
3. $\langle\sigma, \sigma_0'\rangle$ where σ_0' is fully reduced copy of a prefix σ' which satisfies one the two previous conditions.

For case (1) then for every $\sigma.a^-.n$ present in $\mathcal{B}$ it is $\langle\sigma.a^-.n : \nu_0\rangle \in \mathcal{B}$ by completion w.r.t. $\nu(a^-)^F$. Hence, $\imath(\sigma.a^-.n) \models \nu_0$ by inductive hypothesis. For case (2) consider completion w.r.t. the rule $\nu(a^-)^B$: the prefixed formula $\langle\sigma_0 : \nu_0\rangle$ occurs in $\mathcal{B}$. Case (3) is a repetition of the previous two since σ is fully reduced and the relevant rule to

σ' has been applied already. Since σ'_0 is a copy then we have $\sigma'_0 : \nu_0$ and we are done. Thus, for all $\imath(\sigma')$ it is $\imath(\sigma') \models \nu_0$ by inductive hypothesis and therefore, by definition of $\models$, it is $\imath(\sigma) \Vdash \nu(a^-)$.

For $\langle P^*\rangle A$ we have to prove that whenever $\sigma : \mathtt{t}.X$ appears the corresponding $\mathtt{t}.\langle P^*\rangle A$ is satisfied. The proof is by double induction: on the formula size and on number of $\prec$ steps in path$(\sigma : \mathtt{t}.X)$, using Lem. 5.11. One chooses as a base for the latter induction the top prefix σ_ω such that $\langle \sigma_\omega : \mathtt{t}.A\rangle$ is present. By induction hypothesis it is $\imath(\sigma_\omega) \Vdash \mathtt{t}.A$ and by definition it is $\imath(\sigma_\omega) \Vdash \mathtt{t}.\langle P^*\rangle A$. For the induction step consider a pair $\langle \sigma_j : \mathtt{t}.X_j\rangle \prec \langle \sigma_{j+1} : X_{j+1}\rangle$ in path$(\sigma : \mathtt{t}.X)$ so $\imath(\sigma_{j+1}) \Vdash \mathtt{t}.\langle P^*\rangle A$ by inductive hypothesis. If X_j collapsed into X_{j+i} then σ_j is a copy of σ_{j+1} and the result follows trivially: $\imath()$ maps them on the same world. For the other case this means that $X_{j+i} \equiv X_j \equiv X$. Hence the only way to introduce $\sigma_{j+1} : \mathtt{t}.X$ is to reduce completely $\sigma_j : \mathtt{t}.\langle P\rangle X$. By induction on the construction of P it is possible to verify that $\langle \sigma_j, \sigma_{j+1}\rangle$ is in $(P)^{\mathcal{I}}$ and therefore the claim follows by definition of $\Vdash$. For instance if $P \equiv Q; R$ then by π-completion $\sigma_j : \mathtt{t}.\langle Q\rangle\langle R\rangle X$ is on the branch and therefore there must be σ' such that $\langle \sigma_j, \sigma'\rangle \in (Q)^{\mathcal{I}}$ and $\sigma' : \mathtt{t}.\langle R\rangle X$ is present. This also yields $\langle \sigma', \sigma_{j+i}\rangle \in (R)^{\mathcal{I}}$ and the claim follows by induction. $\square$

Appendix B

Proofs of Part II

B.1 Proofs of Chapter 6

Proof of LEMMA 6.12 (page 122).

Proof. Suppose the contrary and let $\mathcal{B}$ be the SAT branch in the tableau and $\mathcal{I}$ be the corresponding model such that for every $\varphi \in \mathcal{B}$ we have $\mathcal{I} \Vdash \varphi$

Clearly a SAT tableau cannot be approximately contradictory due to the presence of a $\mathtt{f}.\top$ or $\mathtt{t}.\bot$. Moreover we cannot have a signed formula φ such that $\mathtt{meaning}(\varphi) = unk$, because no model can satisfy φ by proposition 6.4. So we are only left with the case where both φ and $\overline{\varphi}$ are present in $\mathcal{B}$ and $\mathtt{meaning}(\varphi) = \mathtt{meaning}(\overline{\varphi}) = int$.

The proof now proceeds by induction on the size of φ. In the base case we have $\varphi = \mathtt{t}.p$ and therefore $\mathcal{I} \Vdash \mathtt{t}.p$ and $\mathcal{I} \Vdash \mathtt{f}.p$ for $p \in int(\mathcal{P}rop)$. This is impossible by definition of forcing.

For the inductive step we have two cases: either φ is of type neg or φ is of type α. Notice that in case $\varphi = \beta$ then by proposition 2.1 we have that $\overline{\varphi} = \alpha$ and we can procede in the proof swapping $\varphi = \overline{\overline{\varphi}}$ and $\overline{\varphi}$. The case for neg is immediate.

If $\varphi = \alpha$ then by hypothesis it is $\mathcal{I} \Vdash \alpha$ and by definition of forcing we have that $\mathcal{I} \Vdash \alpha_1$ and $\mathcal{I} \Vdash \alpha_2$. By proposition 2.1 we have that $\overline{\varphi} = \beta$, $\beta_1 = \overline{\alpha_1}$ and $\beta_2 = \overline{\alpha_2}$. Since $\mathcal{B}$ is SAT we must have that either $\mathcal{I} \Vdash \overline{\alpha_1}$ or $\mathcal{I} \Vdash \overline{\alpha_2}$.

At this stage we cannot yet conclude by inductive hypothesis that this is impossible since one α_i may be incoherent and therefore we may have $\mathcal{I} \Vdash \alpha_i$ and $\mathcal{I} \Vdash \overline{\alpha_i}$.

However, by hypothesis, we have that both φ and $\overline{\varphi}$ are interesting. Since $\overline{\varphi}$ is of type β then, according Table 6.1, it can be interesting only if both β_1 and β_2 are not incoherent. Suppose now that either $\beta_1 = \overline{\alpha_1}$ or $\beta_2 = \overline{\alpha_2}$ are unknown, then according Table 6.1 we have that $\varphi = \alpha$ would be unknown. Since φ is interesting this means that *both* α_1 and α_2 are interesting. Then, we can apply the induction hypothesis and derive a contradiction. $\square$

Proof of THEOREM 6.14 (page 123).

Proof. We construct the model $\mathcal{I}$ in the following way:

$$(p)^{\mathcal{I}} \quad = \top \text{ if } \mathtt{t}.p \in \mathcal{B} \text{ and } p \in int(\mathcal{P}rop)$$

Then we have to prove that if $\varphi \in \mathcal{B}$ then $\mathcal{I} \Vdash \varphi$. by induction on the structure of φ.

If φ is an incoherent formula then by Proposition 6.3 every model satisfies it.

The other cases exploit the fact that φ is reduced by definition of open branch and proceeds by induction. In the base case we have $\varphi = \texttt{t}.p$ or $\texttt{f}.p$ and the proof follows by construction since the branch is open and therefore $\texttt{t}.p \in \mathcal{B}$ iff $\texttt{f}.p \notin \mathcal{B}$ for all interesting p. Notice also that p cannot be unknown otherwise the branch would be closed.

Suppose that φ is of type β and it is not incoherent. According Table 6.1 both β_1 and β_2 must not be incoherent. Since the branch is open then, according Table 6.1, they cannot be both unknown. Moreover, β is reduced and therefore either β_1 or β_2 must be in $\mathcal{B}$. Suppose that β_1 is in $\mathcal{B}$. If it is unknown then the branch would be closed, against the hypothesis that the branch is open. Hence, β_1 is interesting and by induction hypothesis we have $\mathcal{I} \Vdash \beta_1$. By definition of forcing we have $\mathcal{I} \Vdash \beta$. The case for β_2 is identical.

Suppose that φ is of type α and it is not incoherent. According Table 6.1 either α_1 or α_2 is not incoherent. Since the branch is open, none of them is unknown (see Table 6.1). Hence they are both interesting. Moreover, α is reduced and therefore both α_1 or α_2 must be in $\mathcal{B}$. By induction hypothesis we have $\mathcal{I} \Vdash \alpha_1$ and $\mathcal{I} \Vdash \alpha_2$ and the claim follows.

The case for *neg* and rule *Loc* is similar. □

Proof of THEOREM 6.20 (page 124).

Proof. Suppose that there is an open branch in $\mathcal{T}_k$. Clearly this branch cannot contain any formula φ such that $\texttt{meaning}(\varphi) = unk$ otherwise would be closed.

So it must contain only incoherent or interesting formulae. Anyhow, incoherent formulae do not change status from ap_k to ap_{k+1}. By definition of open branch, every formula of type interesting must have been already reduced without yielding a contradiction. So $\mathcal{T}_{k+1}$ has still an open branch. □

Proof of THEOREM 6.21 (page 124).

Proof. Suppose that $\mathcal{T}_k$ is approximately closed. Clearly every branch must be either classically contradictory or contain some formula φ such $\texttt{meaning}(\varphi) = unk$. If this was not the case at least a branch would be open.

However only incoherent formulae do change status form ap_k to ap_{k+1} eventually becoming interesting. Since they do not affect the contradictory status of any branch we still have a closed tableau $\mathcal{T}_{k+1}$. □

Proof of THEOREM 6.25 (page 126).

Proof. We must prove by induction on the number of interesting proposition n that the tableau has exactly 2^n branches.

At first observe that α, *dneg* and *simp* rules do not generate branches. So the only way to generate a branch is either the cut rule or the β rule.

For the base case assume that there is no $p \in int(\mathcal{P}rop)$ in a branch $\mathcal{B}$. Then every φ is either of type incoherent or unknown and the branch is either approximately reduced (and open) or approximately contradictory. In this we do not apply any rule to the branch and we stop with only one branch.

For the inductive case, suppose that there is at least one $p \in int(\mathcal{P}rop)$ in $\mathcal{B}$ after the application of α, *dneg* and *simp* rules. So, we apply the cut rule and generate two branches $\mathcal{B}_1 = \mathcal{B} \cup \{\texttt{f}.p\}$ and $\mathcal{B}_2 \cup \{\texttt{t}.p\}$.

Then we must apply simplification to $\mathcal{B}_1$ and $\mathcal{B}_2$ and in particular we must use $\texttt{f}.p$ and $\texttt{t}.p$ as simplifying formulae. For the properties of simplification (§2.3), the proposition p does not occur anymore in the resulting branches $\mathcal{B}'_1$ and $\mathcal{B}'_2$. Hence each of these branches has at most $n-1$ interesting literals and we are done. □

Proof of LEMMA 6.26 (page 127).

Proof. The proof is by induction on the branching depth of φ. Note that if there is an unknown formula in the branch (with branching depth 0) we are done. Moreover incoherent formulae are not reduced.

If φ is a formula of type α or *neg* then we apply the rules until we arrive to (one or more) β formulae. By construction at least one of these (interesting) β formulae must be of branching depth greater or equal to one. We apply the rule to any of these β rule.

The resulting tableau contain two branches with a formula (the corresponding β_i) with lower branching depth. If it is not zero then we apply the induction hypothesis and we are done.

If the branching depth of one β_i is 0 we must reason by cases. If the formula is unknown then we are done. The same happens if it is a signed atoms. It cannot be an incoherent formula because otherwise β would be incoherent and its branching depth would be zero.

We are only left with an interesting α or *neg* formula. But then this formulae will necessarily introduce an interesting signed literal (this can be easily proved by induction) and the claim follows. □

Proof of THEOREM 6.27 (page 127).

Proof. We must prove that in every branch (a path form the root to the leaves of the tableau) we can have at most $d_k \times \mid int_k(\mathcal{P}rop) \mid$ branching points. The proof is by induction on the number of interesting variables.

Observe that α, *dneg* and *simp* rules do not generate branches, and we apply them before other rules. So the only way to generate a branch is the β rule. Moreover observe that $\texttt{meaning}(\beta) = int$ only if $\text{bdepth}(\beta) \geq 1$.

After the reduction of α, *neg* formulae and the application of simplification we can only have β formulae or signed propositional atoms.

If there is signed atom then, by the property of simplification (§2.3) it cannot occur anymore in the formulae which must be still reduced[1]. So we apply the induction hypothesis and we are done.

If any formula on the branch is unknown the branch is closed and we do not have to expand the tableau further. If all formulae are incoherent then we are also done: the branch is s open and the search is stopped.

So we must have some β formulae with branching depth greater than one. Let β be the formula selected by the algorithm. Since β is a formula with the smallest branching depth, both β_1 and β_2 will be also smaller than any other formula, so they will be selected next. Then we apply Lemma 6.26 and we are guaranteed that in any of the lower branches we always introduce either a new signed propositional atom or

[1]This also holds if U is infinite: after every application of the *Loc* rule the atom is still on the branch, so we again apply simplification and immediately delete it from the new formula.

an unknown formula. We can close the branch with the latter and apply the induction hypothesis to the former. The claim follows. □

Proof of THEOREM 6.37 (page 131).

Proof. To prove this result we transform the k-BCP proof into the (obvious) tableau proof obtained using *cut*, *simp* and *dneg*. We know that this is possible by Thm. 6.36. Then, by the soundness result proved by Dalal [37] we know that A is a logical consequence of A in classical logic.

Then we apply a standard cut elimination procedure following Smullyan [163] or Gentzen [70] to the tabelau proof. The formulae used in this process contains only the variables in I_k (indeed are a subset of I_k). Once that all formulae in a proof are interesting, we known that approximate consequence corresponds to classical consequence and we are done. □

B.2 Proofs of Chapter 7

Proof of LEMMA 7.14 (page 146).

Proof. Suppose the contrary and let $\mathcal{B}$ be the SAT branch in the tableau and $\mathcal{I}$ be the corresponding model such that for every $\varphi \in \mathcal{B}$ we have $\mathcal{I} \Vdash \varphi$

Clearly a SAT tableau cannot be approximately contradictory due to the presence of a $\sigma : \mathtt{f}.\top$ or $\sigma : \mathtt{t}.\bot$.

Suppose that we have a unreduced $\sigma : \varphi$ belonging to a such that the condition $\mathtt{meaning}(a, k_a + 1- \mid \sigma \mid, \varphi) = unk$ holds. We need to apply induction on the construction of φ. For the propositional connectives use the same argument for Prop. 6.4. The difficult case is due to $\pi(a)$ subformulae.

Here we need a nested induction on $k_a + 1- \mid \sigma \mid$. If $k_a + 1 = \mid \sigma \mid$ then we are at the border with an unknown world and therefore, by Prop. 7.2, $\pi(a)$ cannot be satisfiable. For the inductive case $k_a + 1 > \mid \sigma \mid$, according Fig. 7.1 we have that $\mathtt{meaning}(a, k_a + 1- \mid \sigma \mid, \pi(a)) = unk$ iff $\mathtt{meaning}(a, k_a- \mid \sigma \mid, \pi_0) = unk$. Now suppose that $\imath(\sigma) \Vdash \pi(a)$. By definition of forcing there must be a word w such that $w \Vdash \pi_0$. The world w is one step closer to the unknown world and therefore we apply the induction hypothesis and we are done.

Therefore we are only left with the case where $\sigma : \varphi$ and $\sigma : \overline{\varphi}$ in $\mathcal{B}$. We only give the proof for the case where σ belongs to a and $\mathtt{meaning}(a, k_a + 1- \mid \sigma \mid, \varphi) = int$ and $\mathtt{meaning}(a, k_a + 1- \mid \sigma \mid, \overline{\varphi}) = int$. The case for the world 1 is analogous.

The proof is by induction on the size of φ. For propositional atoms and boolean connectives the argument is substantially identical to the proof of Lemma. 6.12.

The only new case is the presence of modal formulae. This means we have a modal formula $\sigma : \varphi$ of type $\pi(b)$ and $\sigma : \overline{\varphi}$ of type $\nu(b)$ which are both interesting. By definition of forcing we have that there is world w such that $\imath(\sigma)\mathcal{R}_b w$ and $w \Vdash \nu_0$ and $w \Vdash \pi_0$ such that $\nu_0 = \overline{\pi_0}$. As in the propositional case, we cannot apply yet the induction hypothesis: here w may well be an incoherent world.

By the definition of $\mathtt{meaning}()$ (see §7.2) we have that $\nu(b)$ and $\pi(b)$ are both interesting only if both π_0 and ν_0 are interesting and we have not reached yet the upper bound k_a. Since the initial world $\imath(1)$ is realistic the world w must be interesting. Now we can apply the induction hypothesis and we are done. □

Proof of LEMMA 7.25 (page 149).

Proof. In the model constructed in the completeness proof we use two extra worlds $\langle int \rangle$ and $\langle unk \rangle$. The other worlds are determined by the generated tree of prefixes rooted in $\langle 1 \rangle$. By construction a prefix can be long at most b_k (otherwise the generating π formula would have been either unknown or incoherent). This means that the overall tree can be long at most $2^{b_k} + 2 + 1$. $\square$

Appendix C

Proofs of Part III

C.1 Proofs of Chapter 9

Proof of THEOREM 9.6 (page 179).

Proof. Let $\langle \mathcal{B}, G_\mathcal{B}\rangle$ be the SAT branch of the tableau and $\imath()$ the corresponding mapping. If the rule in not applied on $\mathcal{B}$ then clearly the new tableau is still SAT. If the rule is applied on $\mathcal{B}$ then we must show that the new branch $\langle \mathcal{B}', G'_\mathcal{B}\rangle$ is still SAT, possibly by changing the mapping. For the propositional connectives and the modal operator $\mathtt{says}$ the proof is identical to that of the logic K (Chapter 3. For quoting and conjunction use the techniques used for PDL (Chapter 5).

The only new cases are for the reduction of the operator $\Rightarrow$.

At first consider rule $\langle R_{gr}\rangle$ and suppose that $\sigma : P{\Rightarrow}a$ and $\sigma : \neg(a\, \mathtt{says}\, s)$ are present in B. Then by hypothesis $\imath(\sigma) \Vdash P{\Rightarrow}a$ and $\imath(\sigma) \not\Vdash a\, \mathtt{says}\, s$. therefore for some w one has $\langle \imath(\sigma), w\rangle \in (a)^\mathcal{I}$ and $w \not\Vdash s$. We also have that $(a)^\mathcal{I} \subseteq (P)^\mathcal{I}$ so that $\langle \imath(\sigma), w\rangle \in (P)^\mathcal{I}$ and thus $\imath(\sigma) \not\Vdash P\, \mathtt{says}\, s$ by definition of semantic forcing. Dually for $[L_{gr}]$ we use the fact that $\imath(\sigma) \Vdash a{\Rightarrow}P$ implies that $(P)^\mathcal{I} \subseteq (a)^\mathcal{I}$. If $\sigma : a\, \mathtt{says}\, s$ is present then for all w such that $\langle \imath(\sigma), w\rangle \in (a)^\mathcal{I}$ one has $w \Vdash s$. Hence we also have $\imath(\sigma) \Vdash P\, \mathtt{says}\, s$.

For rule $[U_{gr}]$ suppose that $\sigma : P{\Rightarrow}Q$ occurs in B. Then $\imath(\sigma) \Vdash P{\Rightarrow}Q$ and therefore $(P{\Rightarrow}Q)^\mathcal{I} = W$ for the global property of $\Rightarrow$(it is not empty as it contains $\imath(\sigma)$). Therefore adding it to $G_\mathcal{B}$ as done by the $[U_{gr}]$ does not change the satisfiability of the branch wrt the first condition on $G_\mathcal{B}$ imposed by Def. 9.5.

If $\sigma : \neg(P{\Rightarrow}Q)$ is present then by hypothesis $\imath(\sigma) \Vdash \neg(P{\Rightarrow}Q)$. Hence there is a $\langle w, w^*\rangle \in (Q)^\mathcal{I}$ with $\langle w, w^*\rangle \notin (P)^\mathcal{I}$. Set $\imath(n) = w$ for the new prefix n and $(x_i)^\mathcal{I} = W - \{\, w^* \,\}$ for the new x_i. Clearly $\imath(n) \Vdash P\, \mathtt{says}\, x_i$ but $\imath(n) \not\Vdash (Q\, \mathtt{says}\, x_i)$.
□

Proof of THEOREM 9.8 (page 180).

Proof. Construct a pre-model $\langle W, \mathcal{J}\rangle$ as follows:

$$\begin{array}{rcl} W & \doteq & \{\, \sigma \mid \sigma \text{ is present in } \mathcal{B} \,\} \\ (a)^\mathcal{J} & \doteq & \{\, \langle \sigma, \sigma.a.n\rangle \mid \sigma \text{ and } \sigma.a.n \text{ are present in } \mathcal{B} \,\} \\ (r)^\mathcal{J} & \doteq & \{\, \sigma \mid \sigma : r \in \mathcal{B} \,\} \end{array}$$

Incorporate the constraints due to $\Rightarrow$ and build $()^{\mathcal{I}}$ from $()^{\mathcal{J}}$ as follows:

- for every formula $\sigma : a{\Rightarrow}P$ occurring in $\mathcal{B}$
  - compute $(P)^{\mathcal{I}}$;
  - if $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{I}}$ then add $\langle \sigma, \sigma^* \rangle$ to $(a)^{\mathcal{I}}$;
- repeat until a fix-point is reached.

After this *closure phase* we must prove that $\mathcal{B}$ is SAT in the model $\langle W, \mathcal{I} \rangle$ (according Def. 9.5) with the mapping $\imath(\sigma) = \sigma$: if $\sigma : s$ is present in B then $\imath(\sigma) \Vdash s$ by induction on the construction of s.

Once we have this result, it is easy to prove that for all $s \in G_{\mathcal{B}}$ we have that for all $\sigma \in W$ it is $\sigma \Vdash s$. By saturation w.r.t. the *Glob* rule, for all σ present in $\mathcal{B}$ and $s \in G_{\mathcal{B}}$, one has $\sigma : s \in \mathcal{B}$ (and hence in W).

The proof is similar to those used for CPDL (§5.6) and modal logics (§3.6) and works by induction on the size of s.

The new connective $\Rightarrow$ can be dealt without much difficulties. By construction we satisfy the local properties of $\Rightarrow$. For the global condition we use mutual saturation between *Glob* and $[U_{gr}]$-rules. If $\sigma : a{\Rightarrow}P$ occurs in $\mathcal{B}$ then $[U_{gr}]$-implies that $a{\Rightarrow}P \in G_{\mathcal{B}}$. By *Glob* we have that for all $s \in G_{\mathcal{B}}$ and all prefixes σ in $\mathcal{B}$ it is $\sigma : s$. Hence every σ satisfies the local condition i.e. W satisfies the global condition and therefore $(P)^{\mathcal{I}} \subseteq (a)^{\mathcal{I}}$.

The difficult case is $\sigma : a \,\mathsf{says}\, s$ since we must prove that for all prefixes $\langle \sigma, \sigma^* \rangle \in (a)^{\mathcal{I}}$ it is $\sigma^* : s$ in $\mathcal{B}$ so that we can apply induction to $\sigma^* : s$, get $\sigma^* \Vdash s$ and then the claim. The difference with traditional modal completeness proofs is that some prefixes σ^* are introduced in $(a)^{\mathcal{I}}$ during the closure phase. Hence we can have

- $\sigma^* = \sigma.a.n$ for some n or
- $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{I}}$ before the closure phase and $(P)^{\mathcal{I}} \subseteq (a)^{\mathcal{I}}$ after a step of the closure phase

Hence the proof goes by induction on the number of closure steps and needs a series of intermediate lemmas and propositions.

The base case requires the following property:

Lemma C.1 *If $\langle W, \mathcal{J} \rangle$ is the pre-model of $\mathcal{B}$ and $\sigma : P \,\mathsf{says}\, s \in \mathsf{B}$ then for all σ^* such that $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{J}}$ then the prefixed formula $\sigma^* : s \in \mathcal{B}$.*

The proof here also require induction on the size of P. Thus is P is an atomic principal A we simply apply saturation w.r.t. rule $[A]$

For complex principals we must first show that if $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{I}}$ then we can "travel" from σ to σ^* using only atomic principals as intermediate steps. To this extent we introduce the definition of the initial segment of P (its head) and the remaining segment of P (its tail) as follows:

$$\begin{array}{rcl} head(a) & = & \{\, a \,\} \\ head(P \,\&\, Q) & = & head(P) \cup head(Q) \\ head(P \,|\, Q) & = & head(P) \end{array}$$

Then we use the following proposition to prove the main lemma:

Proposition C.2 *If $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{J}}$ then σ^* has the form $\sigma.a.n.a_1.n_1 \ldots .a_k.n_k$ and $a \in head(P)$.*

Proposition C.3 *If $\sigma : P\,\mathtt{says}\,s \in \mathcal{B}$, $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{J}}$ and the prefix σ^* has the form $\sigma.a.n.a_1.n_1 \ldots .a_k.n_k$ then there are principal $Q_1, \ldots Q_h$ such that the prefixed formula $\sigma.a.n : Q_1\,\mathtt{says} \ldots Q_h\,\mathtt{says}\,s$ is present in $\mathcal{B}$ and $\langle \sigma.a.n, \sigma^* \rangle \in (Q_1 \mid \cdot \mid Q_h)^{\mathcal{J}}$.*

Notice that the presence of a number of Q_i is simply due to the way in which quoting associates in P. If we had a sort of normal form, for instance always associating quoting to the right, then we would only have one Q.

Once we have proven these results we can prove the main result by induction. For the base case we use Lemma C.1. For the inductive step we must show that these properties are preserved for the partial $()^{\mathcal{I}}$ we construct after each step of the iteration. Thus the inductive hypothesis is simply Lemma C.1 where the partial $()^{\mathcal{I}}$ replace $()^{\mathcal{J}}$.

The key point is to observe that whenever $\sigma : a \Rightarrow P$ is present then by saturation $\sigma : a\,\mathtt{says}\,s$ implies $\sigma : P\,\mathtt{says}\,s$. So we apply the inductive hypothesis to get $\sigma^* : s$ from $\sigma : P\,\mathtt{says}\,s$ and therefore when σ^* was added in $(a)^{\mathcal{I}}$ in the closure phase also $\sigma^* : s$ was present. □

Proof of THEOREM 9.10 (page 181).

Proof. [Sketch] We can prove the dual property of the induction step from the previous proof: if $\sigma : \neg(a\,\mathtt{says}\,l)$ and $P \Rightarrow a$ is also present then there is is a $\langle \sigma, \sigma^* \rangle \in (P)^{\mathcal{I}}$ such that $\sigma^* : \neg l$.

This is sufficient to show that all $\sigma : P\,\mathtt{says}\,l'$ are consistent with each $\sigma : \neg(a\,\mathtt{says}\,l)$. Then we use the $D(a)$-rule, to prove that those P statements are consistent also with each $\sigma : a\,\mathtt{says}\,l''$. By $D(A)$ we obtain $\sigma : \neg(a\,\mathtt{says}\,\neg l'')$ and then apply the dual property.

Since all l, l', l'' are literals this is enough: all $a\,\mathtt{says}\,l$ are consistent among themselves, and each of them with all $P\,\mathtt{says}\,l'$. This means that when we add a $\langle \sigma, \sigma^* \rangle$ from $(a)^{\mathcal{I}}$ to $(P)^{\mathcal{I}}$ in the closure phase we can always extend the valuation of the unspecified l' or l'' in σ^* so that the result is still a model. Again, this only works for *request restricted* statements. □

Dottorato di Ricerca in Ingegneria Informatica
Collana della tesi
Collection of Theses

V-93-1 Marco Cadoli. *Two Methods for Tractable Reasoning in Artificial Intelligence: Language Restriction and Theory Approximation.* June 1993.

V-93-2 Fabrizio d'Amore. *Algorithms and Data Structures for Partitioning and Management of Sets of Hyperrectangles.* June 1993.

V-93-3 Miriam Di Ianni. *On the complexity of flow control problems in Store-and-Forward networks.* June 1993.

V-93-4 Carla Limongelli. *The Integration of Symbolic and Numeric Computation by p-adic Construction Methods.* June 1993.

V-93-5 Annalisa Massini. *High efficiency self-routing interconnection networks.* June 1993.

V-93-6 Paola Vocca. *Space-time trade-offs in directed graphs reachability problem.* June 1993.

VI-94-1 Roberto Baldoni. *Mutual Exclusion in Distributed Systems.* June 1994.

VI-94-2 Andrea Clementi. *On the Complexity of Cellular Automata.* June 1994.

VI-94-3 Paolo Giulio Franciosa. *Adaptive Spatial Data Handling.* June 1994.

VI-94-4 Andrea Schaerf. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* June 1994.

VI-94-5 Andrea Sterbini. *2-Thresholdness and its Implications: from the Synchronization with PVchunk to the Ibaraki-Peled Conjecture.* June 1994.

VII-95-1 Piera Barcaccia. *On the Complexity of Some Time Slot Assignment Problems in Switching Systems.* June 1995.

VII-95-2 Michele Boreale. *Process Algebraic Theories for Mobile Systems.* June 1995.

VII-95-3 Antonella Cresti. *Unconditionally Secure Key Distribution Protocols.* June 1995.

VII-95-4 Vincenzo Ferrucci. *Dimension-Independent Solid Modeling.* June 1995.

VII-95-5 Esteban Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging.* June 1995.

VII-95-6 Michele Flammini. *Compact Routing Models: Some Complexity Results and Extensions.* June 1995.

VII-95-7 Giuseppe Liotta. *Computing Proximity Drawings of Graphs.* June 1995.

VIII-96-1 Luca Cabibbo. *Querying and Updating Complex-Object Databases.* May 1996.

VIII-96-2 Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms.* May 1996.

VIII-96-3 Marco Cesati. *Structural Aspects of Parameterized Complexity.* May 1996.

VIII-96-4 Flavio Corradini. *Space, Time and Nondeterminism in Process Algebras.* May 1996.

VIII-96-5 Stefano Leonardi. *On-line Resource Management with Application to Routing and Scheduling.* May 1996.

VIII-96-6 Rosario Pugliese. *Semantic Theories for Asynchronous Languages.* May 1996.

IX-97-1 Paola Alimonti. *Local search and approximability of MAX SNP problems.* May 1997.

IX-97-2 Tiziana Calamoneri. *Does Cubicity Help to Solve Problems?.* May 1997.

IX-97-3 Paolo Di Blasio. *A Calculus for Concurrent Objects: Design and Control Flow Analysis.* May 1997.

IX-97-4 Bruno Errico. *Intelligent Agents and User Modelling.* May 1997.

IX-97-5 Roberta Mancini. *Modelling Interactive Computing by exploiting the Undo.* May 1997.

IX-97-6 Riccardo Rosati. *Autoepistemic Description Logics.* May 1997.

IX-97-7 Luca Trevisan. *Reductions and (Non-)Approximability.* May 1997.

X-98-1 Gianluca Battaglini. *Analysis of Manufacturing Yield Evaluation of VLSI/WSI Systems: Methods and Methodologies.* April 1998.

X-98-2 Piergiorgio Bertoli. *Using OMRS in practice: a case study with Acl-2.* April 1998.

X-98-3 Chiara Ghidini. *A semantics for contextual reasoning: theory and two relevant applications.* April 1998.

X-98-4 Roberto Giaccio. *Visiting complex structures.* April 1998.

X-98-5 Giampaolo Greco. *Dimension and structure in Combinatorics.* April 1998.

X-98-6 Paolo Liberatore. *Compilation of intractable problems and its application to artificial intelligence.* April 1998.

X-98-7 Fabio Massacci. *Efficient approximate tableaux and an application to computer security.* April 1998.

X-98-8 Chiara Petrioli. *Energy-Conserving Protocols for Wireless Communications.* April 1998.