Two Methods for Checking Formulas of Temporal Logic

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Hugh Wingfield McGuire
June 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

⟨signed⟩

Zohar Manna
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

⟨signed⟩

John C. Mitchell

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

⟨signed⟩

Richard Waldinger

Approved for the University Committee on Graduate Studies:

⟨stamped⟩

Abstract

This dissertation presents two methods for determining satisfiability or validity of formulas of Discrete Metric Annotated Linear Temporal Logic. This logic is convenient for representing and verifying properties of reactive and concurrent systems, including software and electronic circuits.

The first method presented here is an algorithm for automatically deciding whether any given propositional temporal formula is satisfiable and, if so, reporting a model of the formula. The classical algorithm for this task defines possible states as settings of the truth-values of particular formulas which are relevant to the given formula; possible states are constructed and then linked according to their associated formulas' constraints on temporally adjacent states, and then certain fulfillment-conditions are checked. The new algorithm here efficiently extends that treatment to formulas with temporal operators which refer to the past or are metric (i.e. refer to measured amounts of time). Then, whereas classical proofs of correctness for such algorithms are existential, the proof here is constructive; the proof here shows that for any given formula being checked, any model of the formula is embedded in the graph of possible states, which implies that the algorithm here can find the model.

The second method presented in this dissertation is a deduction-calculus for determining the validity of predicate temporal formulas. Previous work on deduction in temporal logic exploits already well-developed techniques of deduction in first-order logic by representing temporal operators via first-order expressions with time reified as quantified expressions of the natural numbers. The new deduction-calculus presented here employs a refined, conservative version of this translation from temporal forms to expressions with time reified. Quantifications are elided, and addition is used instead of classical complicated combinations of comparisons. Ordering conditions on arithmetic expressions can arise, but such are handled automatically via unification and a decision-procedure for Presburger arithmetic. These features make this deduction-calculus very convenient. With deduction-rules such as temporal induction, this deduction-calculus is as powerful as other methods. Further deduction-rules such as rewriting are included for additional convenience.

Acknowledgements

I thank my Advisor, Professor Zohar Manna, for the opportunities that he has provided. I heartily thank Visiting Scholar Amir Pnueli, who initiated and encouraged my work in temporal logic. I also thank Consulting Professor Richard Waldinger, another direct facilitator of this work. I thank my fellow doctoral students Henny Sipma,[1] Nikolaj Bjørner, Tomás Uribe, and Luca de Alfaro for editing my work.[2]

I'm grateful to the many people who facilitated this work. Most recent such facilitators are members of Stanford University's research-community working on formal methods for verification and related purposes; these people include professors, research-associates, and students, both within and outside the Department of Computer Science, who have discussed matters with me and given me pointers. I particularly thank my fellow doctoral students Arjun Kapur and Elizabeth Wolf for such help. More generally, the synergy of the research-community here has nurtured my work.

Previous facilitators of my work include: "W*" Ian Gasarch, a doctoral student of Computer Science at Harvard University when I was there, who led adventurous excursions in effective computability; Gerald Sacks, a Professor of Mathematics at Harvard University when I was there, who introduced me to advanced formal methods and encouraged me in Computer Science; and the Mathematical Association of America, which illuminated how fun mathematical 'work' is.

I am very grateful to Carolyn Tajnai, Assistant Chair of the Department of Computer Science for Graduate Studies and External Relations and also Director of the Computer Forum, for arranging funding for this work. And I thank the external agencies that provided these funds: GTE Foundation and Achievement Rewards for College Scientists Foundation Inc.[3]

In conclusion, I thank those who supported me in other ways, including my family and the Computer Science Department's staff — particularly Secretary Phyllis Winkler.

[1] I take this opportunity to thank Henny Sipma further for vital help at my dissertation-defense.

[2] (Any flaws remaining in this work are mine.)

[3] I also take this opportunity to thank the National Science Foundation for a Minority Graduate Fellowship which funded part of my graduate career before I began this work. Further funding has come from my being a teaching-assistant; I particularly thank Administrative Assistant Claire Stager for facilitating this arrangement.

Contents

List of Tables

Chapter 0

A Temporal Logic

Pregnant in matter, in expression brief,
Let every sentence stand in bold relief!

— Joseph Story *

The language here is the Annotated Temporal Logic of McGuire, Manna, and Waldinger [McgMW94],[1] which is a modal logic derived from the Linear Temporal Logic of Manna and Pnueli [ManP91]. Temporal logic has been found to be well-suited for expressing and then reasoning about properties of reactive and concurrent systems; Pnueli [Pn77] initiated this interest in temporal logic within Computer Science.[2] Re particular features of temporal logic, Pnueli [Pn84] and Lichtenstein, Pnueli, and Zuck [LicPZ85] show how explicitly including past-operators in the language is desirable, and Koymans [Koy90, Koy92] and Henzinger [He91] do the same for metric operators: including these features in the language facilitates expression of some temporal properties such as aspects of a variable's (past) history[3] or occurrence of an event within a few moments from now. A caveat is that whereas one might expect use of the semantics of [Koy90, He91, Koy92] for metric temporal operators, the semantics for such here is slightly different, being derived instead from the work of Wilk and Pnueli [WiP89]; this semantics is simpler, enabling the algorithm of Part I of this dissertation to maintain efficiency. The temporal logic here is linear rather than branching because linear semantics is clearer than branching semantics.[4] (But see also the work of Emerson and Halpern [EH86] re expressibility.) The particular scheme of time-annotation used here was

* "Advice to a Young Lawyer", Stanza I. Written in 1832. Reproduced in Story·W.: *Life and Letters of Joseph Story*, Volume II, Chapter II: "Professorial and Judicial Life", page 88. Charles C. Little and James Brown, Boston, 1851.

[1] The notation here using "@" is new.

[2] For history and perspective, see also the presentation of Glanz [Gl95].

[3] [Pn84] indicates that such historical information facilitates modular verification.

[4] Consider how difficult it is in branching temporal logic to semantically distinguish the formulas $\mathbf{AG}\mathsf{F}p$, $\mathbf{AGA}\mathsf{F}p$, and $\mathbf{AGE}\mathsf{F}p$.

developed (in autumn, 1992) from the work of Shoham [Shoh87] and Abadi and Manna [AbM90]. (Frühwirth [Frü94] has derived a similar but more limited scheme of time-annotation from the work of Kifer and Subrahmanian [KifS92]; see also page 91 below re [Frü94]'s scheme.)

0.1 Syntax

Symbols

Basic symbols of the language here comprise:

- **function-symbol**s, collected in the set F; examples include "f", "g", and zero-arity "a" and "b".

  The meta-symbol "γ" denotes an arbitrary function-symbol.

- **parameter-symbol**s, collected in the set P; examples include "x", "y", and "z".

  The meta-symbol "χ" denotes an arbitrary parameter-symbol; the meta-expression "$\overline{\chi}$" denotes an arbitrary sequence of them, e.g. "$[x_1, x_2, x_3]$".

- **temporal variable**s, collected in the set V; examples include "v".

  The meta-symbol "ν" denotes an arbitrary temporal variable.

- **relation-** or **predicate-symbol**s, collected in the set R; examples include "p", "q", and "r". Relation-symbols used with zero arity, called **proposition**s, are also **temporal**ly **variable**. (The significance of temporal variability is semantic; see below.)

  The meta-symbol "ρ" denotes an arbitrary relation-symbol. The meta-symbol "π" denotes an arbitrary proposition.

- the equality-symbol, "=".

- **boolean operator**s:

symbol	pronunciation	operation	arities
true	true	verity	0
false	false	falsity	0
$\neg$	not	negation	1
$\wedge$	and	conjunction	every integer ≥ 2
$\vee$	or	disjunction	every integer ≥ 2
$\Rightarrow$	implies	implication	2
$\Leftrightarrow$	is equivalent to	equivalence	2
$\otimes$	ex-or	exclusive disjunction	every integer ≥ 2

  The operator $\otimes$ does not accept $\otimes$-formulas as arguments.[5] The meta-symbol "$\star$" denotes an arbitrary boolean operator.

[5]Without this restriction, the formula $(true \otimes true \otimes true)$ would have a value different from that of the formula $((true \otimes true) \otimes true)$; the restriction avoids this situation by simply disallowing the latter formula.

- **quantifier-symbols**:
  - "$\forall$" (pronounced "for all"), which is used for universal quantification.
  - "$\exists$" (pronounced "there exists"), which is used for existential quantification.

  The meta-symbol "$\mathcal{Q}$" denotes an arbitrary quantifier-symbol.
- **temporal operators**:

symbol	pronunciations	arity
first	first	0
$\bigcirc$	next	1
$\Box$	henceforth, box	1
$\Diamond$	eventually, diamond	1
U	until	2
A	awaiting	2
$\ominus$	previously	1
$\boxminus$	hitherto, box-minus	1
$\diamondminus$	once, was, diamond-minus	1
S	since	2
B	barring, back to	2
$\odot$	term-next, next value of	1

  (Instead of "A", [ManP91] uses "W", with the name "unless".)

  The operator "$\odot$" applies to terms; the other operators apply to formulas.

The symbols for time comprise:

- the natural **numerals** "0" and "1", which are zero-arity function-symbols.
- "+", the addition-function-symbol, whose arities comprise every integer ≥ 2 and whose arguments must be time-representing terms (when this symbol is used for time).
- "*pred*", the predecessor-function-symbol, whose arity is 1 and whose argument must be a time-representing term.
- other function-symbols, e.g. "h_1", "h_2", and zero-arity "c_1" and "c_2".

  Each of these function-symbols accepts terms as arguments only if they do not contain any temporal variables or "$\odot$".
- other parameter-symbols, e.g. "t_1", "t_2", "d_1", and "d_2". Such a subscripted or otherwise accented "t" is generally used to represent an absolute time, and such a subscripted or otherwise accented "d" is generally used to represent a duration of time, a.k.a. a delay, a.k.a. a difference between two absolute times.

Symbols for time are not themselves temporally variable; they specify time-frames but are not affected by such.

Additional **numerals** for time are "2", "3", "4", etc., which are abbreviations for "1+1", "1+1+1", "1+1+1+1", etc. (respectively). The meta-symbol "η" denotes an arbitrary numeral. The expression "$\langle\eta + 1\rangle$" denotes "$\eta + 1$" except when η is "0", in which case "$\langle\eta + 1\rangle$" denotes "1".

In addition to the **type** "time" which is specified for the symbols for time, other **types** (a.k.a. "sort"s in a sorted logic) can be specified for other function-symbols, parameter-symbols, temporal variables, and arguments for function- and relation-symbols.

The language here also includes the following binary relation-symbols:

– "$\leq$", "$<$", and "$\cong$"[6], each of whose arity is 2.

For each of these relation-symbols, if one of its arguments is a time-representing term, then its other argument must also be a time-representing term.

The temporal operators $\bigcirc$, $\ominus$, and $\odot$ can be **superscript**ed by numerals; for example: "$\bigcirc^3$". Any other temporal operator can be **metric**ized by **subscript**ing it with one of the comparison-symbols "$\leq$", "$<$", or "$\cong$" and a numeral. For examples, "$\Box_{\leq 4}$" is such a metric temporal operator and "$(p \; \mathsf{U}_{\cong 5} \; q)$" is a complete formula containing such. In such subscripts, the symbols "$\leq$", "$<$", and "$\cong$" are pronounced "through" or "within", "before", or "each", respectively.

Expressions of the Language

A **term** is one of the following:

- a parameter-symbol, e.g. "x".
- a temporal variable, e.g. "v".
- an application of a function-symbol to a number of argument-terms satisfying the function-symbol's requirements of arity and argument-types; e.g. "$f(x)$", "a" (with zero arguments), and "$t + c + 1$" (using multi-ary infix notation for the function-symbol "$+$").

  The **type** of a function-application is the type of its function-symbol.
- an application of the operator "$\odot$" to a term, e.g. "$\odot v$".

  The **type** of a "$\odot$"-term is the type of the sub-term to which "$\odot$" is applied.

The meta-symbols "τ" and "σ" denote arbitrary terms. A **time-representing term** is a term whose type is time. The meta-symbol "θ" denotes an arbitrary time-representing term.

A **quantifier** is the application of a quantifier-symbol to a positive number of parameter-symbols; e.g. "$(\forall x)$", "$(\exists y, z)$".

[6] The pronunciation of "$\cong$" is "congruent". This symbol expresses a variant of modular equivalence: $m \cong n$ means $m = 0 \;\; (MOD \; n)$. (See semantics, below.)

A **formula** is one of the following:

- an application of a relation-symbol (i.e. a predicate-symbol) to a number of argument-terms satisfying the relation-symbol's requirements of arity and argument-types; e.g. "p" (with zero arguments), "$q(x)$", and "$t_1 < t_2$".
- an equation "$\tau = \sigma$", where τ and σ are terms of the same type; e.g. "$a = f(x)$" and "$t = 3$".
- an application of a boolean operator to a number of argument-formulas satisfying the operator's arity and obeying the restriction that the operator $\otimes$ does not accept $\otimes$-formulas as arguments.
- an application of a possibly superscripted or subscripted temporal operator to an appropriate number of argument-formulas.
- a **time-annotated** formula "$[\theta : \psi]$", where θ is a time-representing term and ψ is a formula which is not a time-formula (see below). In a time-annotated formula $[\theta : \psi]$, the term θ is the **time-annotation** and the formula ψ is its argument.[7]
- an application of a quantifier to an argument-formula which does not have any time-annotations which contain any of the quantifier's parameter-symbols.

The meta-symbols "φ", "ψ", and "ξ" denote arbitrary formulas.

A **time-comparison** is an application of either "$=$", "$\leq$", "$<$", or "$\cong$" to two time-representing terms. A **time-formula** is either a time-comparison or an application of a boolean operator or a quantifier to time-formulas.

In a quantification-formula $(\mathcal{Q}\,\overline{\chi})\psi$, i.e. $(\forall\, \chi_1, \chi_2, \ldots, \chi_k)\psi$ or $(\exists\, \chi_1, \chi_2, \ldots, \chi_k)\psi$, each quantified parameter-symbol χ_i is **bound**. In a formula, a parameter-symbol's occurrences that are not bound are **free**. In $(\mathcal{Q}\,\overline{\chi})\psi$, the occurrences of each parameter-symbol χ_i that are bound specifically by the salient quantifier $(\mathcal{Q}\,\overline{\chi})$ are those that are free in ψ when it is considered as an independent formula.

In an application of a boolean operator, a temporal operator, a time-annotation, or a quantifier to argument-formulas (or -terms), the arguments comprise the **scope** of the operator, annotation, or quantifier — except that the scope of a temporal operator or time-annotation does not extend over the scope of any inner time-annotation. For an example of this exception, in the formula $\big[3 : \Box(p \wedge [0 : q])\big]$ the scopes of the time-annotation 3 and the temporal operator $\Box$ do not extend over the scope of the inner time-annotation 0.

[7] See the work of Mints [Mi92] for similar annotation of formulas.

Organization of Operators and Their Arguments

Prefix-notation is used for each operator whose arity is 1; infix-notation is used for each operator whose arity or arities is or are larger. Parentheses ("(" and ")") and brackets ("[" and "]") may be used as delimiters of expressions.

For the binary boolean operator "$\Rightarrow$" and the binary temporal operators, right-associativity is used when delimiters are omitted. For example, the formula $(p \Rightarrow q \Rightarrow r)$ is understood as $\big(p \Rightarrow (q \Rightarrow r)\big)$, and the formula $(p \mathsf{U} q \mathsf{U} r)$ is understood as $\big(p \mathsf{U} (q \mathsf{U} r)\big)$. The binary boolean operator "$\Leftrightarrow$" is not associative; so, for example, the unparenthesized sequence of symbols "$p \Leftrightarrow q \Leftrightarrow r$" is not acceptable (a.k.a. not "well-formed").

Unary operators have the strongest precedence. For example, the formula $(\neg\, p \mathsf{U} q)$ is understood as $\big((\neg p) \mathsf{U} q\big)$ rather than $\big(\neg(p \mathsf{U} q)\big)$. The binary temporal operators all share the next strongest precedence. For examples: the formula $(p \wedge q \mathsf{U} r)$ is understood as $\big(p \wedge (q \mathsf{U} r)\big)$ rather than $\big((p \wedge q) \mathsf{U} r\big)$, and the formula $(p_1 \mathsf{U} p_2 \mathsf{A} p_3 \mathsf{U} p_4 \mathsf{A} p_5)$ is understood as $\big[p_1 \mathsf{U} \big(p_2 \mathsf{A} [p_3 \mathsf{U} (p_4 \mathsf{A} p_5)]\big)\big]$ — applying right-associativity since the operators "U" and "A" each have the same precedence. The precedences of the remaining operators are specified by the following list of them ordered by precedence, from strongest to weakest: "$\wedge$", "$\otimes$", "$\vee$", "$\Rightarrow$", "$\Leftrightarrow$", and time-annotation.

0.2 Semantics

Set-notation used here includes the zero-ary (empty) tuple "()" as an acceptable tuple; this allows S^0, for a set S, to be $\{()\}$. For any set S, the expression "$\wp\big(S\big)$" denotes the power-set of S, i.e. the set of all subsets of S.

The domain for time here is a structure "$\boldsymbol{T}$" which is a copy of the natural numbers $\boldsymbol{N}$.[8] A **time-value** or **time** is an element of $\boldsymbol{T}$. The meta-symbol "t" denotes an arbitrary time-value.

A **model** $\boldsymbol{\mathcal{M}}$ is a structure such that:

- Associated with $\boldsymbol{\mathcal{M}}$ is a nonempty set of **objects**, denoted by "$|\boldsymbol{\mathcal{M}}|$", called the **universe** of $\boldsymbol{\mathcal{M}}$. Some of these objects may have types.
- Model $\boldsymbol{\mathcal{M}}$ assigns type-consistent objects and times to the language's parameter-symbols ("x", "t", etc.), i.e. $\boldsymbol{\mathcal{M}}$ maps P to $|\boldsymbol{\mathcal{M}}| \cup \boldsymbol{T}$. For any parameter-symbol $\chi \in P$, the expression "$\boldsymbol{\mathcal{M}}[\![\chi]\!]$" denotes $\boldsymbol{\mathcal{M}}$'s assignment to χ.
- Model $\boldsymbol{\mathcal{M}}$ assigns type-consistent functions over $|\boldsymbol{\mathcal{M}}| \cup \boldsymbol{T}$ to the language's function-

[8] Thus, time is discrete here.

symbols, obeying arities. For any function-symbol $\gamma \in F$ and arity $k \in \boldsymbol{N}$, the expression "$\mathcal{M}[\![\gamma, k]\!]$" denotes $\mathcal{M}$'s assignment to γ with arity k. Obedience of arity is as follows: For each arity $k \in \boldsymbol{N}$, let the collection $\mathsf{F}_k \subset \wp\big((|\mathcal{M}| \cup \boldsymbol{T})^{k+1}\big)$ denote the functions over $|\mathcal{M}| \cup \boldsymbol{T}$ that have arity k. Then for each function-symbol $\gamma \in F$ and arity $k \in \boldsymbol{N}$, the assignment $\mathcal{M}[\![\gamma, k]\!] \in \mathsf{F}_k$.

– Model $\mathcal{M}$ assigns type-consistent relations over $|\mathcal{M}| \cup \boldsymbol{T}$ to the language's temporally invariable relation-symbols, obeying arities. For any relation-symbol $\rho \in R$ and nonzero arity $k \in \boldsymbol{N} - \{0\}$, the expression "$\mathcal{M}[\![\rho, k]\!]$" denotes $\mathcal{M}$'s assignment to ρ with arity k. Obedience of arity is as follows: For each relation-symbol $\rho \in R$ and arity $k \in \boldsymbol{N} - \{0\}$, the assignment $\mathcal{M}[\![\rho, k]\!] \in \wp\big((|\mathcal{M}| \cup \boldsymbol{T})^k\big)$.

– Associated with model $\mathcal{M}$ is a sequence of **states** indexed by $\boldsymbol{T}$; model $\mathcal{M}$'s state with index $\mathsf{t} \in \boldsymbol{T}$ is denoted by "$\mathcal{M}@\mathsf{t}$".

In addition to inheriting the preceding assignments of the model to temporally invariable symbols, a state of a model assigns appropriate values using the model's universe to temporally variable symbols.[9] Specifically, a state assigns values to the language's temporal variables and propositions (i.e. relation-symbols used with zero arity), mapping V to $|\mathcal{M}|$ and $R \times \{0\}$ to $\wp\big(|\mathcal{M}|^0\big)$. For any temporal variable $\nu \in V$, the expression "$(\mathcal{M}@\mathsf{t})[\![\nu]\!]$" denotes state $(\mathcal{M}@\mathsf{t})$'s assignment to ν. For any proposition $\pi \in R$, the expression "$(\mathcal{M}@\mathsf{t})[\![\pi, 0]\!]$" denotes state $(\mathcal{M}@\mathsf{t})$'s assignment to π. (The "0" indicates the zero arity.)

A model assigns appropriate values to the symbols that represent time — numbers which match to the numerals, addition to the function-symbol "+", and so on. (The predecessor-function assigned to "*pred*" may map the argument-value zero arbitrarily.) A model also assigns appropriate values to the relation-symbols "<", "≤", and "≅" over $\boldsymbol{T}$; in particular, $\mathcal{M}[\![\cong, 2]\!] \cap \boldsymbol{T}^2$ is the set: {all the pairs (m, n) such that m is a multiple of n}.

If a term τ does not contain any temporal variables or "$\odot$", then a model $\mathcal{M}$'s interpretation of τ, denoted by the expression "$\mathcal{M}[\![\tau]\!]$", is an object of $\mathcal{M}$'s universe $|\mathcal{M}|$ determined as follows:

– If term τ is a parameter-symbol χ, then $\mathcal{M}[\![\tau]\!]$ is $\mathcal{M}[\![\chi]\!]$.

– If term τ is $\gamma(\tau_1, \tau_2, \ldots, \tau_k)$, the application of the function-symbol γ with arity $k \in \boldsymbol{N}$ to argument-terms τ_1, τ_2, ..., τ_k, then with $\mathcal{M}[\![\gamma, k]\!]$ being a function, the interpretation $\mathcal{M}[\![\tau]\!]$ is $\mathcal{M}[\![\gamma, k]\!]\big(\mathcal{M}[\![\tau_1]\!], \mathcal{M}[\![\tau_2]\!], \ldots, \mathcal{M}[\![\tau_k]\!]\big)$.

A state $\mathcal{M}@\mathsf{t}$ (for $\mathsf{t} \in \boldsymbol{T}$) interprets those terms the same; additionally:

– If term τ is a temporal variable ν, then $(\mathcal{M}@\mathsf{t})[\![\tau]\!]$ is $(\mathcal{M}@\mathsf{t})[\![\nu]\!]$.

---

[9] Others (e.g. [AbM90]) use the word "rigid" instead of "temporally invariable" and the word "flexible" instead of "temporally variable".

- If term τ is $\odot\sigma$, the application of the temporal operator "$\odot$" to term σ, then $(\mathcal{M}@\mathsf{t})[\![\tau]\!]$ is $\big(\mathcal{M}@(\mathsf{t}+1)\big)[\![\sigma]\!]$.

If a formula φ contains no temporally variable symbols or temporal operators outside the scope of time-annotations in φ, then a model $\mathcal{M}$ either **satisfies** or **falsifies** φ. The expression "$\mathcal{M} \models \varphi$" is an abbreviation for the statement "$\mathcal{M}$ satisfies φ", and the expression "$\mathcal{M} \not\models \varphi$" is an abbreviation for the statement "$\mathcal{M}$ falsifies φ". Satisfaction or falsification of a formula φ by a model $\mathcal{M}$ is determined as follows:

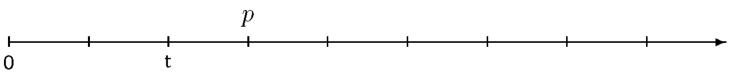
- If formula φ is $\rho(\tau_1, \ldots, \tau_k)$, the application of the relation-symbol ρ with nonzero arity $k \in \boldsymbol{N} - \{0\}$ to argument-terms $\tau_1, \ldots, \tau_k$, then with $\mathcal{M}[\![\rho, k]\!]$ being a relation, $\mathcal{M} \models \varphi$ if and only if (iff) the tuple $\big(\mathcal{M}[\![\tau_1]\!], \mathcal{M}[\![\tau_2]\!], \ldots, \mathcal{M}[\![\tau_k]\!]\big) \in \mathcal{M}[\![\rho, k]\!]$.
- If formula φ is an equation $(\tau = \sigma)$, then $\mathcal{M} \models \varphi$ iff $\mathcal{M}[\![\tau]\!]$ is the same object as $\mathcal{M}[\![\sigma]\!]$.
- $\mathcal{M} \models \mathit{true}$.
- $\mathcal{M} \not\models \mathit{false}$.
- $\mathcal{M} \models \neg\psi$ iff $\mathcal{M} \not\models \psi$.
- $\mathcal{M} \models (\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$ iff $\mathcal{M} \models \psi_1$, $\mathcal{M} \models \psi_2$, $\ldots$, and $\mathcal{M} \models \psi_k$.
- $\mathcal{M} \models (\psi_1 \vee \psi_2 \vee \cdots \vee \psi_k)$ iff $\mathcal{M} \models \psi_1$, $\mathcal{M} \models \psi_2$, $\ldots$, or $\mathcal{M} \models \psi_k$.
- $\mathcal{M} \models (\psi \Rightarrow \xi)$ iff $\mathcal{M} \not\models \psi$ or $\mathcal{M} \models \xi$.
- $\mathcal{M} \models (\psi \Leftrightarrow \xi)$ iff $\mathcal{M}$ satisfies both ψ and ξ or neither of them.
- $\mathcal{M} \models (\psi_1 \otimes \psi_2 \otimes \cdots \otimes \psi_k)$ iff $\mathcal{M}$ satisfies exactly one of ψ_1, ψ_2, $\ldots$, and ψ_k.
- $\mathcal{M} \models (\forall \chi_1, \chi_2, \ldots, \chi_k)\psi$ iff for every k objects o_1, o_2, $\ldots$, and o_k contained in $|\mathcal{M}| \cup \boldsymbol{T}$ (with each o_i's type the same as the corresponding χ_i's type), the model $\widetilde{\mathcal{M}}$ satisfies formula ψ, where model $\widetilde{\mathcal{M}}$'s universe, assignments, and states are the same as $\mathcal{M}$'s except that $\widetilde{\mathcal{M}}[\![\chi_1]\!]$ is o_1, $\widetilde{\mathcal{M}}[\![\chi_2]\!]$ is o_2, $\ldots$, and $\widetilde{\mathcal{M}}[\![\chi_k]\!]$ is o_k.
- $\mathcal{M} \models (\exists \chi_1, \chi_2, \ldots, \chi_k)\psi$ iff $|\mathcal{M}| \cup \boldsymbol{T}$ contains k objects o_1, o_2, $\ldots$, and o_k (with each o_i's type the same as the corresponding χ_i's type) such that the model $\widetilde{\mathcal{M}}$ satisfies formula ψ, where model $\widetilde{\mathcal{M}}$'s universe, assignments, and states are the same as $\mathcal{M}$'s except that $\widetilde{\mathcal{M}}[\![\chi_1]\!]$ is o_1, $\widetilde{\mathcal{M}}[\![\chi_2]\!]$ is o_2, $\ldots$, and $\widetilde{\mathcal{M}}[\![\chi_k]\!]$ is o_k.
- $\mathcal{M} \models [\theta : \psi]$ iff model $\mathcal{M}$'s state $\mathcal{M}@\big(\mathcal{M}[\![\theta]\!]\big)$ satisfies ψ. (Satisfaction/falsification of a formula by a state is defined next.)

A state $\mathcal{M}@\mathsf{t}$, for $\mathsf{t} \in \boldsymbol{T}$, interprets those formulas the same (including time-annotated ones — $\mathcal{M}@\mathsf{t} \models [\theta : \psi]$ iff $\mathcal{M}@\big(\mathcal{M}[\![\theta]\!]\big) \models \psi$); additionally:

- If formula φ is a proposition, i.e. the application of a relation-symbol π to zero arguments, then with $(\mathcal{M}@\mathsf{t})[\![\pi, 0]\!]$ being a zero-arity relation, $\mathcal{M}@\mathsf{t} \models \varphi$ iff the zero-ary (empty) tuple $() \in (\mathcal{M}@\mathsf{t})[\![\pi, 0]\!]$.

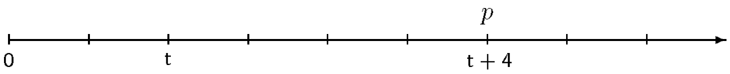
- $\mathcal{M}@\mathsf{t} \models \mathit{first}$ iff t is zero.
- $\mathcal{M}@\mathsf{t} \models \bigcirc\psi$ iff $\mathcal{M}@(\mathsf{t}+1) \models \psi$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models \bigcirc p$:



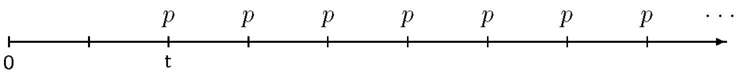
- $\mathcal{M}@\mathsf{t} \models (\bigcirc^{\eta}\psi)$ iff $\mathcal{M}@(\mathsf{t}+\eta) \models \psi$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\bigcirc^{4} p)$:



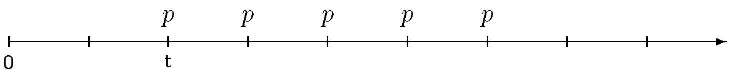
- $\mathcal{M}@\mathsf{t} \models \Box\psi$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}) \models \psi$ for each time-delay $\mathsf{d} \in \boldsymbol{T}$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models \Box p$:



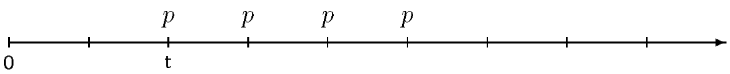
- $\mathcal{M}@\mathsf{t} \models (\Box_{\leq\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}) \models \psi$ for each time-delay $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{d} \leq \eta$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\Box_{\leq 4}\, p)$:



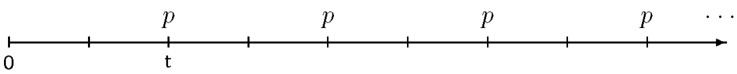
- $\mathcal{M}@\mathsf{t} \models (\Box_{<\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}) \models \psi$ for each time-delay $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{d} < \eta$. (These conditions are vacuously satisfied if η is 0 — regardless of ψ; e.g. the formula $(\Box_{<0}\,\mathit{false})$ is satisfied — always.) An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\Box_{<4}\, p)$:



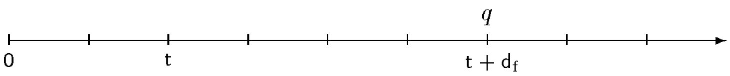
- $\mathcal{M}@\mathsf{t} \models (\Box_{\cong\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}) \models \psi$ for each time-delay $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{d} \cong \eta$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\Box_{\cong 2}\, p)$:



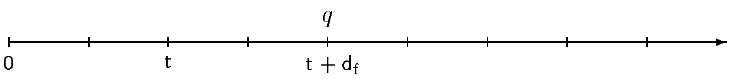
- $\mathcal{M}@\mathsf{t} \models \Diamond\psi$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d_f}) \models \psi$ for some time-delay $\mathsf{d_f} \in \boldsymbol{T}$; then, the formula $\Diamond\psi$ is **fulfilled** at the time $\mathsf{t}+\mathsf{d_f}$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models \Diamond q$:



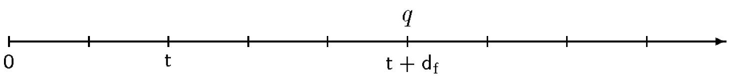
- $\mathcal{M}@\mathsf{t} \models (\Diamond_{\leq\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d_f}) \models \psi$ for some time-delay $\mathsf{d_f} \in \boldsymbol{T}$ such that $\mathsf{d_f} \leq \eta$; then, the formula $\Diamond\psi$ is **fulfilled** at the time $\mathsf{t}+\mathsf{d_f}$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\Diamond_{\leq 4}\, q)$:



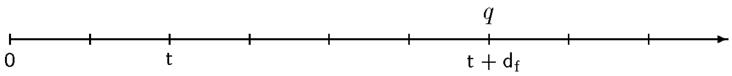
- $\mathcal{M}@\mathsf{t} \models (\Diamond_{<\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d_f}) \models \psi$ for some time-delay $\mathsf{d_f} \in \boldsymbol{T}$ such that $\mathsf{d_f} < \eta$. (Satisfying these conditions is impossible when η is 0; e.g. the formula $(\Diamond_{<0}\,\mathit{true})$ is unsatisfiable.) If satisfied thus, the formula $\Diamond\psi$ is **fulfilled** at the time $\mathsf{t}+\mathsf{d_f}$. An example may be depicted as follows:

  $\mathcal{M}@\mathsf{t} \models (\Diamond_{<4}\, q)$:



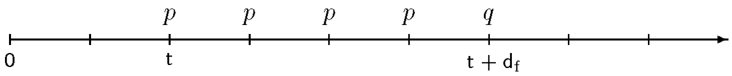
– $\mathcal{M}@\mathsf{t} \models (\Diamond_{\cong\eta}\, \psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}_\mathrm{f}) \models \psi$ for some time-delay $\mathsf{d}_\mathrm{f} \in \boldsymbol{T}$ such that $\mathsf{d}_\mathrm{f} \cong \eta$; then, the formula $\Diamond\psi$ is **fulfilled** at the time $\mathsf{t}+\mathsf{d}_\mathrm{f}$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\Diamond_{\cong 2}\, p)$:



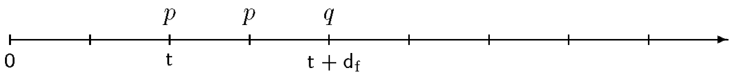
– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}\ \psi)$ iff $\mathcal{M}@(\mathsf{t}+\mathsf{d}_\mathrm{f}) \models \psi$ for some time-delay $\mathsf{d}_\mathrm{f} \in \boldsymbol{T}$ and $\mathcal{M}@(\mathsf{t}+\mathsf{d}_\mathrm{i}) \models \xi$ for each intermediate time-delay $\mathsf{d}_\mathrm{i} \in \boldsymbol{T}$ such that $\mathsf{d}_\mathrm{i} < \mathsf{d}_\mathrm{f}$; then, the formula $(\xi\ \mathsf{U}\ \psi)$ is **fulfilled** at the time $\mathsf{t}+\mathsf{d}_\mathrm{f}$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (p\ \mathsf{U}\ q)$:



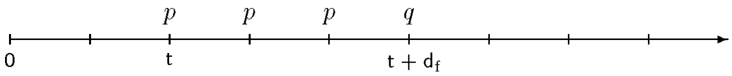
– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{\leq\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\,\mathsf{U}\,\psi)$ with the additional requirement that $\mathsf{d}_\mathrm{f} \leq \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (p\ \mathsf{U}_{\leq 4}\ q)$:



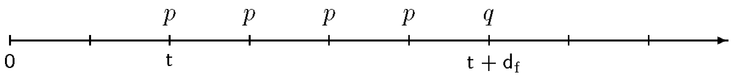
– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{<\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\,\mathsf{U}\,\psi)$ with the additional requirement that $\mathsf{d}_\mathrm{f} < \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (p\ \mathsf{U}_{<4}\ q)$:



– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{\cong\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\,\mathsf{U}\,\psi)$ with the additional requirement that $\mathsf{d}_\mathrm{f} \cong \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (p\ \mathsf{U}_{\cong 2}\ q)$:



– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{A}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models \Box\xi$.

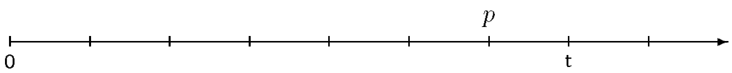
– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{A}_{\leq\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{\leq\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models (\Box_{\leq\eta}\,\xi)$.

– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{A}_{<\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{<\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models (\Box_{<\eta}\,\xi)$.

– $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{A}_{\cong\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{U}_{\cong\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models \Box\xi$ (*sic* — no subscript for the operator "$\Box$" here).

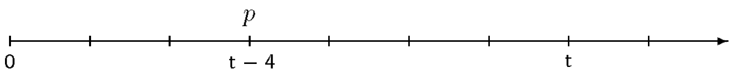
– $\mathcal{M}@\mathsf{t} \models \ominus\psi$ iff $\mathsf{t} \neq 0$ and $\mathcal{M}@(\mathsf{t}-1) \models \psi$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models \ominus p$:



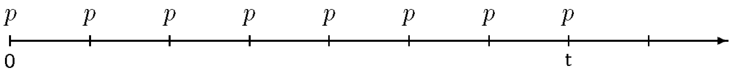
– $\mathcal{M}@\mathsf{t} \models (\ominus^{\eta}\psi)$ iff $\eta \leq \mathsf{t}$ and $\mathcal{M}@(\mathsf{t}-\eta) \models \psi$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\ominus^{4} p)$:



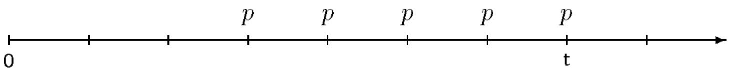
– $\mathcal{M}@\mathsf{t} \models \boxminus\psi$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for each time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$, i.e. iff $\mathcal{M}@\tilde{\mathsf{t}} \models \psi$ for each time-value $\tilde{\mathsf{t}} \in \boldsymbol{T}$ such that $\tilde{\mathsf{t}} \leq \mathsf{t}$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models \boxminus p$:



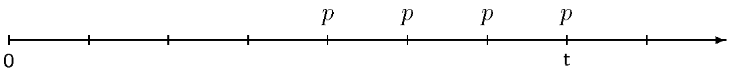
– $\mathcal{M}@\mathsf{t} \models (\boxminus_{\leq\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for each time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} \leq \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\boxminus_{\leq 4}\,p)$:



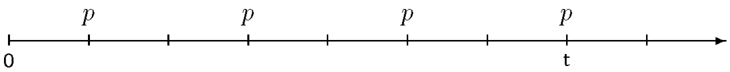
– $\mathcal{M}@\mathsf{t} \models (\boxminus_{<\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for each time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} < \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\boxminus_{<4}\,p)$:



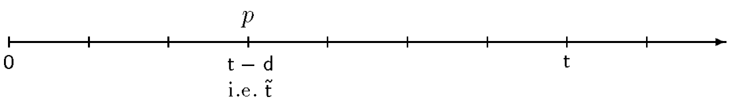
– $\mathcal{M}@\mathsf{t} \models (\boxminus_{\cong\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for each time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} \cong \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\boxminus_{\cong 2}\,p)$:



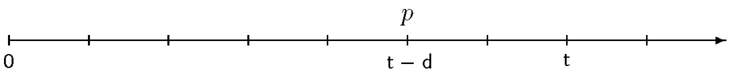
– $\mathcal{M}@\mathsf{t} \models \diamondminus\psi$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for some time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$, i.e. iff $\mathcal{M}@\tilde{\mathsf{t}} \models \psi$ for some time-value $\tilde{\mathsf{t}} \in \boldsymbol{T}$ such that $\tilde{\mathsf{t}} \leq \mathsf{t}$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models \diamondminus p$:



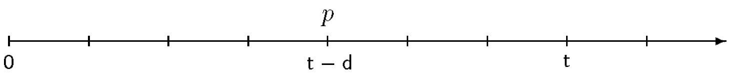
– $\mathcal{M}@\mathsf{t} \models (\diamondminus_{\leq\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for some time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} \leq \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\diamondminus_{\leq 4}\,p)$:



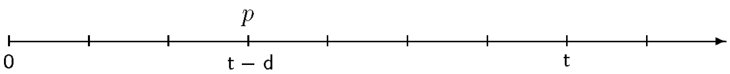
– $\mathcal{M}@\mathsf{t} \models (\diamondminus_{<\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for some time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} < \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\diamondminus_{<4}\,p)$:



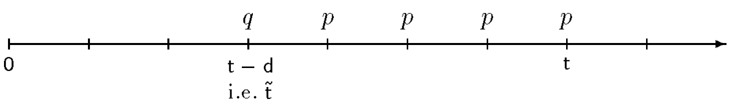
– $\mathcal{M}@\mathsf{t} \models (\diamondminus_{\cong\eta}\,\psi)$ iff $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for some time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and $\mathsf{d} \cong \eta$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (\diamondminus_{\cong 2}\,p)$:



– $\mathcal{M}@\mathsf{t} \models (\xi\,\mathsf{S}\,\psi)$ iff [1] $\mathcal{M}@(\mathsf{t}-\mathsf{d}) \models \psi$ for some time-difference $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{t}-\mathsf{d} \in \boldsymbol{T}$ and [2] $\mathcal{M}@\mathsf{d}_\mathrm{i} \models \xi$ for each intermediate time-difference $\mathsf{d}_\mathrm{i} \in \boldsymbol{T}$ such that $\mathsf{d}_\mathrm{i} < \mathsf{d}$, i.e. iff $[\tilde{1}]$ $\mathcal{M}@\tilde{\mathsf{t}} \models \psi$ for some time-value $\tilde{\mathsf{t}} \in \boldsymbol{T}$ such that $\tilde{\mathsf{t}} \leq \mathsf{t}$ and $[\tilde{2}]$ $\mathcal{M}@\mathsf{t}_\mathrm{i} \models \xi$ for each intermediate time-value $\mathsf{t}_\mathrm{i} \in \boldsymbol{T}$ such that $\tilde{\mathsf{t}} < \mathsf{t}_\mathrm{i}$ and $\mathsf{t}_\mathrm{i} \leq \mathsf{t}$. An example may be depicted as follows:

$\mathcal{M}@\mathsf{t} \models (p\,\mathsf{S}\,q)$:



- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{\leq\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}\ \psi)$ with the additional requirement that $\mathsf{d} \leq \eta$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{<\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}\ \psi)$ with the additional requirement that $\mathsf{d} < \eta$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{\cong\eta}\ \psi)$ is determined like $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}\ \psi)$ with the additional requirement that $\mathsf{d} \cong \eta$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{B}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models \boxminus\xi$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{B}_{\leq\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{\leq\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models (\boxminus_{\leq\eta}\,\xi)$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{B}_{<\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{<\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models (\boxminus_{<\eta}\,\xi)$.
- $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{B}_{\cong\eta}\ \psi)$ iff $\mathcal{M}@\mathsf{t} \models (\xi\ \mathsf{S}_{\cong\eta}\ \psi)$ or $\mathcal{M}@\mathsf{t} \models \boxminus\xi$ (*sic* — no subscript for the operator "$\boxminus$" here).

A situation $\mathcal{M} \models [0 : \varphi]$ is abbreviated as "$\mathcal{M} \models \varphi$". The relationship "$\models$" — i.e. "satisfies" — applies to sets of formulas in a manner naturally derived from its application to individual formulas: if S is a set of formulas, then $\mathcal{M} \models S$ iff $\mathcal{M}$ satisfies each element of S.

A formula is **valid** if it is satisfied by all models.

A **theory of a domain** is specified by a set of formulas A which are called the theory's **domain-axioms**. A formula φ is **valid within the theory of the domain** specified by the set A if for every model $\mathcal{M}$ that satisfies all the domain-axioms A, $\mathcal{M} \models \varphi$.

0.3 Alternative Notations

Alternative notations may be defined using the language above. Some examples are:

- $\widetilde{\ominus}\psi \;\equiv\; \neg\ominus\neg\psi$ ("weakly previously").
- $\widehat{\square}\psi \;\equiv\; \bigcirc\square\psi$ ("hereafter" or "strictly henceforth").
- $(\xi \;\widehat{\mathsf{U}}\; \psi) \;\equiv\; \bigcirc(\xi \;\mathsf{U}\; \psi)$ ("hereafter until").
- $(\psi \Rrightarrow \psi) \;\equiv\; \square(\psi \Rightarrow \psi)$ ("entails").
- $(\square_{=\eta}\, \psi) \;\equiv\; (\bigcirc^{\eta} \psi)$.
- $(\diamond_{=\eta}\, \psi) \;\equiv\; (\bigcirc^{\eta} \psi)$.
- $(\xi \;\mathsf{U}_{=\eta}\; \psi) \;\equiv\; (\square_{<\eta}\, \xi) \wedge (\bigcirc^{\eta} \psi)$.
- $(\square_{\geq\eta}\, \psi) \;\equiv\; (\bigcirc^{\eta} \square\psi)$.
- $(\xi \;\mathsf{U}_{\geq\eta}\; \psi) \;\equiv\; (\square_{<\eta}\, \xi) \wedge \big(\bigcirc^{\eta} (\xi \;\mathsf{U}\; \psi)\big)$.
- $(\square_{[\kappa,\eta]}\, \psi) \;\equiv\; \big(\bigcirc^{\kappa} (\square_{\leq\langle\eta-\kappa\rangle}\, \psi)\big)$, where "$\langle\eta - \kappa\rangle$" is the difference between η and κ.
- $(\xi \;\mathsf{U}_{[\kappa,\eta]}\; \psi) \;\equiv\; (\square_{<\kappa}\, \xi) \wedge \big(\bigcirc^{\kappa} (\xi \;\mathsf{U}_{\leq\langle\eta-\kappa\rangle}\; \psi)\big)$.
- $(\square_{=\kappa\,(MOD\ \eta)}\, \psi)$ or $(\square_{=_{\eta}\kappa}\, \psi) \;\equiv\; \big(\bigcirc^{\kappa} (\square_{\cong\eta}\, \psi)\big)$.
- $(\xi \;\mathsf{U}_{=\kappa\,(MOD\ \eta)}\; \psi)$ or $(\xi \;\mathsf{U}_{=_{\eta}\kappa}\; \psi) \;\equiv\; (\square_{<\kappa}\, \xi) \wedge \big(\bigcirc^{\kappa} (\xi \;\mathsf{U}_{\cong\eta}\; \psi)\big)$.
- $\pi' \;\equiv\; \bigcirc\pi$ for a proposition π; e.g. "p'".
- $\nu' \;\equiv\; \odot\nu$ for a temporal variable ν.

Part I

A Method for Checking Propositional Temporal Formulas

Chapter I.0

Introduction to Part I

"If you can look into the seeds of time,
And say which grain will grow and which will not . . ."
— William Shakespeare *

The propositional fragment of temporal logic is decidable. This part of this dissertation presents one such decision-procedure which is designed for efficiency.[1]

Lichtenstein, Pnueli, and Zuck [LicPZ85] present the fundamental concepts for algorithms such as the one here: Given a formula to check for satisfiability[2], "atoms" (which were originally presented by Fischer and Ladner [FiscL77]) are defined as particular subsets of a certain finite pool of relevant formulas (which are derived from the given formula); atoms are considered like possible states. Atoms are used to construct a "semantic tableau", as discussed by Pratt [Pr78], Wolper [Wo85], and Ben-Ari [Ben93]; a semantic tableau is a directed graph whose vertices are these atoms. In this graph, an edge connects one vertex to another only if such a connection is consistent with the two vertices' $\bigcirc$- and $\ominus$-formulas (e.g. if the first vertex has the formula $\bigcirc p$, then the second vertex should have p.) The originally given formula is satisfiable if — and only if — a path satisfying certain "fulfillment"-conditions can be found in the graph. All of the operations involved in this algorithm are finite, so it is a decision-procedure.[3]

That basic algorithm can be tuned for efficiency via a reachability-restriction (a.k.a. "incrementally" constructing the graph) as in the work of [Pr78], Manna and Wolper [ManWo84, Wo85], and Sherman and Pnueli [SheP89].[4] But those systems do not handle operators that refer to the past,

* *Macbeth*, Act I, Scene III, Lines 58–59. Written in 1605 or 1606.

[1] This part of this dissertation is an extended version of the joint work of Kesten, Manna, McGuire, and Pnueli [KeMMP93].

[2] Checking the validity of a formula can be done by checking the unsatisfiability of the negation of the formula.

[3] This algorithm is ultimately due to Kripke [Kr63].

[4] For a somewhat different algorithm which also employs reachability, see the work of McMillan [Mcm93].

i.e. "⊟", "S", etc. Kesten and Pnueli [KeP91] extended [SheP89] to handle past-operators in some circumstances, but those circumstances were severely restrictive: formulas could not have any future-operator within the scope of any past-operator. (It was not possible to separate this restriction from that algorithm and its proof of correctness.)

The work here extends [SheP89] to handle arbitrary formulas with past-operators while reachability is still employed, so efficiency is maintained. The key idea is that temporal operators determine local constraints (as discussed by Montanari [Mo74], Mackworth [Mac77], Freuder [Fre78], Kumar [Ku92], and Ciapessoni, Corsetti, Crivelli, and Migliorati [CiCCM94]) which can be propagated through a graph — even while the graph is being constructed; indeed, here, it is this propagation which drives the construction. In this scheme, past-operators can be handled exactly like future-operators but with their constraints operating in the reverse direction on edges. There are non-local constraints[5]: ◇- and U-formulas need fulfillment. But as in [SheP89], checking these non-local constraints is easy (i.e. the complexity of this operation is merely linear in the size of the graph).

A further extension to handle metric temporal operators [Koy90, He91, Koy92] is 'free': it is necessary only to determine their constraints as for the basic temporal operators. (Handling "$\cong$" does require adjustment of fulfillment-checking.) Efficiency is maintained if suitable semantics is used.

The classical proof of correctness of such algorithms is existential. For example, the proof in [LicPZ85] shows that if the originally given formula has a model, then the graph's plenitude of atoms includes ones corresponding to the model's states, and the graph's plenitude of edges includes ones corresponding to the model's succession of states, so when the algorithm seeks a path satisfying fulfillment-conditions, it will succeed. But when construction of the graph is restricted via reachability, what guarantees that necessary vertices and edges are present? Previous work does not clearly address this issue.

The proof of correctness here is thorough — and constructive[6]. The key idea of this proof is that given a formula to check for satisfiability, any model of the formula can be embedded in the graph at each stage of the graph's construction. In updating an embedding as the graph is modified, a strategy of being greedy for fulfillment, supplemented by 'patching'[7], succeeds. Thus — given a model — the proof here shows that the algorithm successfully reports satisfiability. Conversely, if

[5] These non-local constraints are related to those discussed in the work of Burch, Clarke, McMillan, and Dill [BuCMD90].

[6] Beeson [Bee85] shows why constructiveness is desirable in Mathematics (see also the work of Martin-Löf [Mar79], Constable et al. [Con86], and Manna and Waldinger [ManWa92]) — perhaps even slightly more desirable in 'applied' Mathematics such as Computer Science.

[7] The idea for this patching is inspired by the work of Soare [So76, So87]

a model is reported by the algorithm to be satisfying, it is indeed so. (Previous work does address this converse somewhat, e.g. [Ben93] (Theorem 5.4.9, on page 226 of that work); a small amount of additional work is necessary to handle the additional past-operators and metric operators.)

Chapter I.1

An Algorithm for Deciding Satisfiability

I.1.1 Notation

The algorithm here checks the satisfiability of any given propositional formula, employing a graph in the manner just described above.

Note that with numerals in metric operators being the only terms of the language used in this part of this dissertation, some symbols for terms are used otherwise here: e.g. "v" and "V" as vertices of the graph, "a" as a set of formulas, and "c" as a constant number. Such reinterpretations are given where they are used.

The relevant graph's vertices are sets of formulas. These sets are called **state-sketch**es (a.k.a. 'atom's) since they can be used to determine states in a model.

In a directed graph, comprising vertices and directed edges, if u and v are vertices connected by an edge from u to v, then vertex v is a **successor** of vertex u, vertex u is a **predecessor** of v, and each of u and v is a **neighbor** of the other. A vertex w is an **eventual successor** of a vertex u if either w is u or w is an eventual successor of an ordinary successor of u.

A **literal** is either a proposition or the negation of a proposition; e.g. "p", "$\neg p$".

Formulas which are input or which otherwise arise during this algorithm are 'simplified' as follows: First, the formula $\neg \mathit{false}$ replaces true.[1] Then, any formula having the form $\neg\neg\varphi$ is simplified to φ. Next, occurrences of the operators "$\bigcirc$" and "$\ominus$" which lack superscripts are automatically given

[1] The formula "true" could be used; but generally, here, such an intrinsically easily satisfiable formula is inconsequential.

the superscript "1"; and occurrences with the superscript "0" are deleted (e.g. "$(\bigcirc^0 p)$" $\longrightarrow$ "p"). Metric temporal operators using the symbol "$<$" are replaced as follows:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\Box_{<0}\,\psi & \longrightarrow & \mathit{true} & \boxminus_{<0}\,\psi & \longrightarrow & \mathit{true} & & & \\
\Diamond_{<0}\,\psi & \longrightarrow & \mathit{false} & \diamondminus_{<0}\,\psi & \longrightarrow & \mathit{false} & \text{“}_{<(\eta+1)}\text{”} & \longrightarrow & \text{“}_{\leq\eta}\text{”} \\
\xi\ \mathsf{U}_{<0}\ \psi & \longrightarrow & \mathit{false} & \xi\ \mathsf{S}_{<0}\ \psi & \longrightarrow & \mathit{false} & & & \\
\xi\ \mathsf{A}_{<0}\ \psi & \longrightarrow & \mathit{true} & \xi\ \mathsf{B}_{<0}\ \psi & \longrightarrow & \mathit{true} & & &
\end{array}$$

Any formula $(\xi\ \mathsf{A}_{\cong 0}\ \psi)$ is changed to $(\psi \vee \Box\xi)$, any formula $(\xi\ \mathsf{B}_{\cong 0}\ \psi)$ is changed to $(\psi \vee \boxminus\xi)$, and any other formula $(\ldots_{\cong 0}\ \psi)$ is reduced to ψ. Lastly, any subscript "$_{\cong 1}$" is erased.

Constraints

As the goal here is to delineate a model along a path in the graph that is constructed, the graph is constrained to conform to characteristics of models, as follows.

The first characteristic is that a model $\mathcal{M}$ has just one initial state $\mathcal{M}@0$. This situation engenders the ***first*-constraint** that when a path in the graph is used to delineate a model, only the path's first state-sketch may contain the formula *first*. The construction of the graph begins with this *first*-constraint satisfied for all paths, actually; indeed, each state-sketch contains either *first* or $\neg$*first*, indicating its (potential) role as either a first state-sketch or a successor-state-sketch in a path used to delineate a model. During subsequent elaboration of the graph, each new state-sketch inherits this feature (either *first* or $\neg$*first*) from an old state-sketch.

A larger class of constraints on the graph is the class of **immediate constraint**s: If a state-sketch s contains a formula φ, then φ must not be *false*, s must not contain $\neg\varphi$,[2] and s must contain certain additional formulas which are specified according to φ's outermost operator(s). For most formulas φ, the options for these additional formulas which s must contain correspond to the various ways of satisfying φ (as specified by semantics); for example, the formula $(p \vee q)$ can be satisfied either by satisfying p or by satisfying q, so if s contains $(p \vee q)$, then s must contain either p or q. For other formulas, satisfaction of them is not reducible like that, so essentially no additional formulas are required to satisfy their immediate constraints; examples of such 'irreducible' formulas are $\neg$*false* and $\bigcirc p$. For every formula φ, the options for additional formulas required to satisfy φ's immediate constraints are specified in Table I.1 (on the following two pages).[3]

---

[2] If φ is a negation $\neg\psi$, remember that $\neg\varphi = \neg\neg\psi$ automatically reduces to ψ, so the condition $\neg\varphi \notin s$ reduces to $\psi \notin s$, as is appropriate.

[3] Compare the reductions here to some of the "rewritings" of Tables II.9, II.10, II.11, and II.12 (on pages 101–103, in Part II).

Table I.1: Options for Additional Formulas Required by Immediate Constraints

Formula	Option #1	Further Options (if any)
false	(no options)	
$\neg\mathit{false}$, any literal, first, $\neg\mathit{first}$	$\{\}$	
$\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k$	$\{\psi_1, \psi_2, \ldots, \psi_k\}$	
$\psi_1 \vee \psi_2 \vee \cdots \vee \psi_k$	$\{\psi_1\}$	$\{\psi_2\}$ $\ldots$ $\{\psi_k\}$
$\psi \Rightarrow \xi$	$\{\neg\psi\}$	$\{\xi\}$
$\psi \Leftrightarrow \xi$	$\{\psi, \xi\}$	$\{\neg\psi, \neg\xi\}$
$\psi_1 \otimes \psi_2 \otimes \cdots \otimes \psi_k$	$\{\psi_1, \neg\psi_2, \neg\psi_3, \ldots, \neg\psi_k\}$	$\{\neg\psi_1, \psi_2, \neg\psi_3, \neg\psi_4, \ldots, \neg\psi_k\}$
		$\{\neg\psi_1, \neg\psi_2, \psi_3, \neg\psi_4, \neg\psi_5, \ldots, \neg\psi_k\}$
		$\vdots$
		$\{\neg\psi_1, \neg\psi_2, \ldots, \neg\psi_{k-2}, \neg\psi_{k-1}, \psi_k\}$
$\neg(\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$	$\{\neg\psi_1\}$	$\{\neg\psi_2\}$ $\ldots$ $\{\neg\psi_k\}$
$\neg(\psi_1 \vee \psi_2 \vee \cdots \vee \psi_k)$	$\{\neg\psi_1, \neg\psi_2, \ldots, \neg\psi_k\}$	
$\neg(\psi \Rightarrow \xi)$	$\{\psi, \neg\xi\}$	
$\neg(\psi \Leftrightarrow \xi)$	$\{\psi, \neg\xi\}$	$\{\neg\psi, \xi\}$
$\neg(\psi_1 \otimes \psi_2 \otimes \cdots \otimes \psi_k)$	$\{\neg\psi_1, \neg\psi_2, \ldots, \neg\psi_k\}$	$\{\psi_1, \psi_2\}$ $\{\psi_1, \psi_3\}$ $\ldots$ $\{\psi_1, \psi_k\}$
		$\{\psi_2, \psi_3\}$ $\ldots$ $\{\psi_2, \psi_k\}$
		$\vdots$ $\iddots$
		$\{\psi_{k-1}, \psi_k\}$
$\bigcirc^{\langle\eta+1\rangle}\psi$	$\{\}$	
$\ominus^{\langle\eta+1\rangle}\psi$	$\{\neg\mathit{first}\}$	
$\neg\left(\bigcirc^{\langle\eta+1\rangle}\psi\right)$	$\{\left(\bigcirc^{\langle\eta+1\rangle}\neg\psi\right)\}$	
$\neg\left(\ominus^{\langle\eta+1\rangle}\psi\right)$	$\{\mathit{first}\}$	$\{\ominus\neg(\ominus^{\eta}\psi)\}$
$\Box\psi$	$\{\psi, \bigcirc\Box\psi\}$	
$\Diamond\psi$	$\{\psi\}$	$\{\bigcirc\Diamond\psi\}$
$\xi\ \mathsf{U}\ \psi$	$\{\psi\}$	$\{\xi, \bigcirc(\xi\ \mathsf{U}\ \psi)\}$
$\xi\ \mathsf{A}\ \psi$	$\{\psi\}$	$\{\xi, \bigcirc(\xi\ \mathsf{A}\ \psi)\}$
$\boxminus\psi$	$\{\psi, \neg\ominus\neg\boxminus\psi\}$	
$\diamondminus\psi$	$\{\psi\}$	$\{\ominus\diamondminus\psi\}$
$\xi\ \mathsf{S}\ \psi$	$\{\psi\}$	$\{\xi, \ominus(\xi\ \mathsf{S}\ \psi)\}$
$\xi\ \mathsf{B}\ \psi$	$\{\psi\}$	$\{\xi, \neg\ominus\neg(\xi\ \mathsf{B}\ \psi)\}$
$\neg\Box\psi$	$\{\Diamond\neg\psi\}$	
$\neg\Diamond\psi$	$\{\Box\neg\psi\}$	
$\neg(\xi\ \mathsf{U}\ \psi)$	$\{\neg\psi\ \mathsf{A}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{A}\ \psi)$	$\{\neg\psi\ \mathsf{U}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg\boxminus\psi$	$\{\diamondminus\neg\psi\}$	
$\neg\diamondminus\psi$	$\{\boxminus\neg\psi\}$	
$\neg(\xi\ \mathsf{S}\ \psi)$	$\{\neg\psi\ \mathsf{B}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{B}\ \psi)$	$\{\neg\psi\ \mathsf{S}\ (\neg\xi \wedge \neg\psi)\}$	

Table I.1 (continued)

$\Box_{\leq 0}\, \psi$	$\{\psi\}$	
$\Box_{\leq \langle \eta+1 \rangle}\, \psi$	$\{\psi,\ \bigcirc(\Box_{\leq \eta}\, \psi)\}$	
$\Diamond_{\leq 0}\, \psi$	$\{\psi\}$	
$\Diamond_{\leq \langle \eta+1 \rangle}\, \psi$	$\{\psi\}$	$\{\bigcirc(\Diamond_{\leq \eta}\, \psi)\}$
$\xi\ \mathsf{U}_{\leq 0}\ \psi$	$\{\psi\}$	
$\xi\ \mathsf{U}_{\leq \langle \eta+1 \rangle}\ \psi$	$\{\psi\}$	$\{\xi,\ \bigcirc(\xi\ \mathsf{U}_{\leq \eta}\ \psi)\}$
$\xi\ \mathsf{A}_{\leq 0}\ \psi$	$\{\psi\}$	$\{\xi\}$
$\xi\ \mathsf{A}_{\leq \langle \eta+1 \rangle}\ \psi$	$\{\psi\}$	$\{\xi,\ \bigcirc(\xi\ \mathsf{A}_{\leq \eta}\ \psi)\}$
$\boxminus_{\leq 0}\, \psi$	$\{\psi\}$	
$\boxminus_{\leq \langle \eta+1 \rangle}\, \psi$	$\{\psi,\ \neg\ominus\neg(\boxminus_{\leq \eta}\, \psi)\}$	
$\diamondminus_{\leq 0}\, \psi$	$\{\psi\}$	
$\diamondminus_{\leq \langle \eta+1 \rangle}\, \psi$	$\{\psi\}$	$\{\ominus(\diamondminus_{\leq \eta}\, \psi)\}$
$\xi\ \mathsf{S}_{\leq 0}\ \psi$	$\{\psi\}$	
$\xi\ \mathsf{S}_{\leq \langle \eta+1 \rangle}\ \psi$	$\{\psi\}$	$\{\xi,\ \ominus(\xi\ \mathsf{S}_{\leq \eta}\ \psi)\}$
$\xi\ \mathsf{B}_{\leq 0}\ \psi$	$\{\psi\}$	$\{\xi\}$
$\xi\ \mathsf{B}_{\leq \langle \eta+1 \rangle}\ \psi$	$\{\psi\}$	$\{\xi,\ \neg\ominus\neg(\xi\ \mathsf{B}_{\leq \eta}\ \psi)\}$
$\Box_{\cong \eta}\, \psi$	$\big\{\psi,\ \big[\bigcirc^{\eta}(\Box_{\cong \eta}\, \psi)\big]\big\}$	
$\Diamond_{\cong \eta}\, \psi$	$\{\psi\}$	$\big\{\bigcirc^{\eta}(\Diamond_{\cong \eta}\, \psi)\big\}$
$\xi\ \mathsf{U}_{\cong \eta}\ \psi$	$\{\psi\}$	$\big\{(\Box_{<\eta}\, \xi),\ \big(\bigcirc^{\eta}(\xi\ \mathsf{U}_{\cong \eta}\ \psi)\big)\big\}$
$\xi\ \mathsf{A}_{\cong \eta}\ \psi$	$\{\psi\}$	$\big\{(\Box_{<\eta}\, \xi),\ \big(\bigcirc^{\eta}(\xi\ \mathsf{A}_{\cong \eta}\ \psi)\big)\big\}$
$\boxminus_{\cong \eta}\, \psi$	$\big\{\psi,\ \neg\big[\ominus^{\eta}\neg(\boxminus_{\cong \eta}\, \psi)\big]\big\}$	
$\diamondminus_{\cong \eta}\, \psi$	$\{\psi\}$	$\big\{\ominus^{\eta}(\diamondminus_{\cong \eta}\, \psi)\big\}$
$\xi\ \mathsf{S}_{\cong \eta}\ \psi$	$\{\psi\}$	$\big\{(\boxminus_{<\eta}\, \xi),\ \big(\ominus^{\eta}(\xi\ \mathsf{S}_{\cong \eta}\ \psi)\big)\big\}$
$\xi\ \mathsf{B}_{\cong \eta}\ \psi$	$\{\psi\}$	$\big\{(\boxminus_{<\eta}\, \xi),\ \neg\big(\ominus^{\eta}\neg(\xi\ \mathsf{B}_{\cong \eta}\ \psi)\big)\big\}$
$\neg(\Box_{\leq \eta}\, \psi)$	$\{\Diamond_{\leq \eta}\, \neg\psi\}$	
$\neg(\Diamond_{\leq \eta}\, \psi)$	$\{\Box_{\leq \eta}\, \neg\psi\}$	
$\neg(\xi\ \mathsf{U}_{\leq \eta}\ \psi)$	$\{\neg\psi\ \mathsf{A}_{\leq \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{A}_{\leq \eta}\ \psi)$	$\{\neg\psi\ \mathsf{U}_{\leq \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\boxminus_{\leq \eta}\, \psi)$	$\{\diamondminus_{\leq \eta}\, \neg\psi\}$	
$\neg(\diamondminus_{\leq \eta}\, \psi)$	$\{\boxminus_{\leq \eta}\, \neg\psi\}$	
$\neg(\xi\ \mathsf{S}_{\leq \eta}\ \psi)$	$\{\neg\psi\ \mathsf{B}_{\leq \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{B}_{\leq \eta}\ \psi)$	$\{\neg\psi\ \mathsf{S}_{\leq \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\Box_{\cong \eta}\, \psi)$	$\{\Diamond_{\cong \eta}\, \neg\psi\}$	
$\neg(\Diamond_{\cong \eta}\, \psi)$	$\{\Box_{\cong \eta}\, \neg\psi\}$	
$\neg(\xi\ \mathsf{U}_{\cong \eta}\ \psi)$	$\{\neg\psi\ \mathsf{A}_{\cong \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{A}_{\cong \eta}\ \psi)$	$\{\neg\psi\ \mathsf{U}_{\cong \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\boxminus_{\cong \eta}\, \psi)$	$\{\diamondminus_{\cong \eta}\, \neg\psi\}$	
$\neg(\diamondminus_{\cong \eta}\, \psi)$	$\{\boxminus_{\cong \eta}\, \neg\psi\}$	
$\neg(\xi\ \mathsf{S}_{\cong \eta}\ \psi)$	$\{\neg\psi\ \mathsf{B}_{\cong \eta}\ (\neg\xi \wedge \neg\psi)\}$	
$\neg(\xi\ \mathsf{B}_{\cong \eta}\ \psi)$	$\{\neg\psi\ \mathsf{S}_{\cong \eta}\ (\neg\xi \wedge \neg\psi)\}$	

The next class of constraints on the graph is the class of **neighbor-constraints**. These constraints, which each state-sketch s imposes on each of its edges to neighbor state-sketches n, are as follows:

- If n is a successor of s, then for each formula $\varphi \in s$ having the form $(\bigcirc^{\langle\eta+1\rangle}\psi)$, state-sketch n must contain the formula $(\bigcirc^{\eta}\psi)$.
- If n is a predecessor of s, then for each formula $\varphi \in s$ having the form $(\ominus^{\langle\eta+1\rangle}\psi)$, state-sketch n must contain the formula $(\ominus^{\eta}\psi)$.

If the edge between s and n satisfies s's neighbor-constraints, then it is **accurate in the direction** `future` or `past`, respectively as n is either a successor or a predecessor of s. (The phrase "in the direction" here may be abbreviated as "toward".) An edge which is not accurate in a direction is **inaccurate** in that direction.

The final class of constraints on the graph is the class of **fulfillment-constraints**: If any state-sketch s contains a formula φ having the form $\diamond\psi$, $(\xi \ \mathsf{U}\ \psi)$, $(\diamond_{\cong\eta}\ \psi)$, or $(\xi\ \mathsf{U}_{\cong\eta}\ \psi)$, then some eventual successor u of s must contain formula ψ — and, in the cases with "$\cong_\eta$", the path P from s to u must be such that its length $|P| \cong \eta$.[4]

I.1.2 An Introductory Example

The algorithm here operates by actually letting all those constraints drive the processing: state-sketches (which are used as vertices in the graph that is constructed) are built as aggregations of formulas specified by immediate constraints, and when an edge does not satisfy a neighbor-constraint, it is replaced by edges connected to new state-sketches which are created to satisfy the neighbor-constraint. Before presenting all the details completely, it's convenient to outline the algorithm and show an example.

The overall algorithm's (pseudo-)code is as follows:

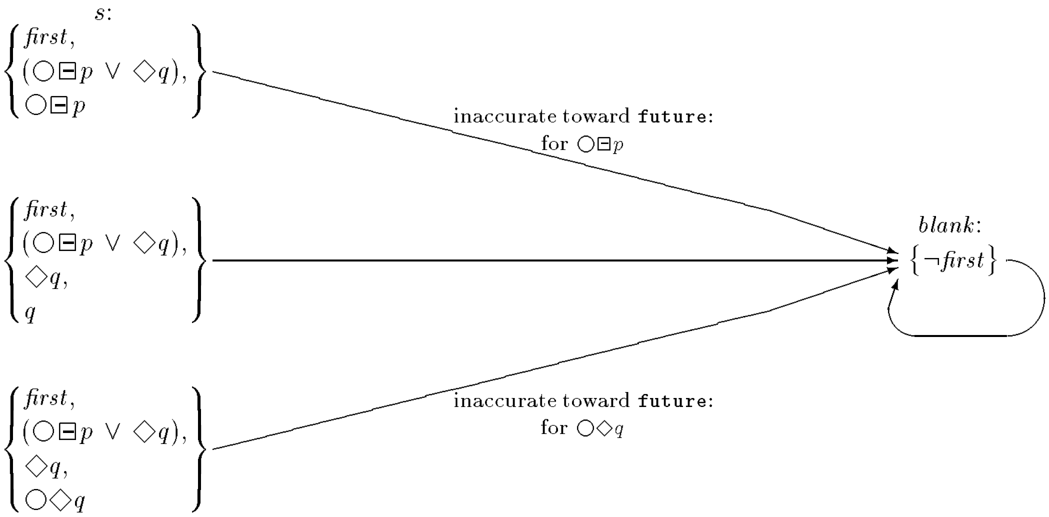
```
satisfy(formula φ_checking):
    init_graph();
    elaborate_graph();
    search_graph().
```

Procedure `init_graph()` initializes the graph, and procedure `elaborate_graph()` repeatedly adds

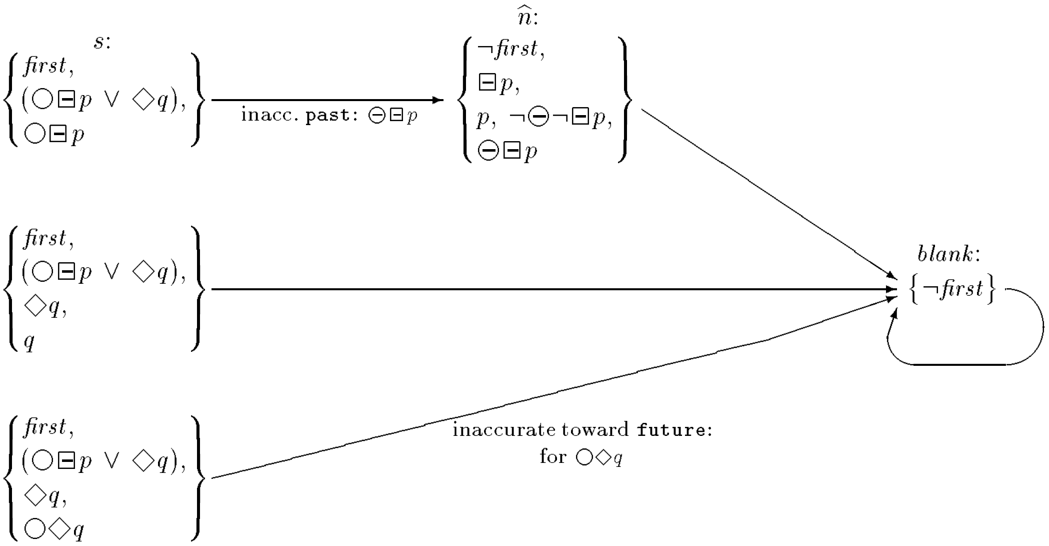
[4] Formulas having the forms $\neg(\xi\ \mathsf{A}\ \psi)$ and $\neg(\xi\ \mathsf{A}_{\cong\eta}\ \psi)$ would also engender fulfillment-constraints if their options of Table I.1 were different. For example, suppose the options for $\neg(\xi\ \mathsf{A}\ \psi)$ were $\{\neg\xi, \neg\psi\}$ and $\{\neg\psi, \bigcirc\neg(\xi\ \mathsf{A}\ \psi)\}$. Then, the fulfillment-constraint for formula $\neg(\xi\ \mathsf{A}\ \psi)$ contained in state-sketch s would be that some eventual successor of s must contain both $\neg\xi$ and $\neg\psi$. With "$\cong_\eta$", there would be a further condition re the length of the path to the fulfilling eventual successor. Having the options as they are seems simpler than having these additional fulfillment-constraints.

new vertices and edges to satisfy neighbor-constraints; each of these procedures complies with immediate constraints. Indeed, to ensure completeness, all 'consistent' options for satisfying immediate constraints are used. The function `search_graph()` seeks a path (in the final graph) which can be used to delineate a model; since any model's domain for time is infinite, strongly connected components (SCCs) in the graph are considered since they can be used to yield infinite paths. Procedure `search_graph()` checks whether SCCs satisfy fulfillment-constraints. If it finds such an SCC, procedure `search_graph()` derives a model from the satisfactory SCC, and all processing halts. Otherwise, the algorithm reports that the given formula $\varphi_{\text{checking}}$ is unsatisfiable.

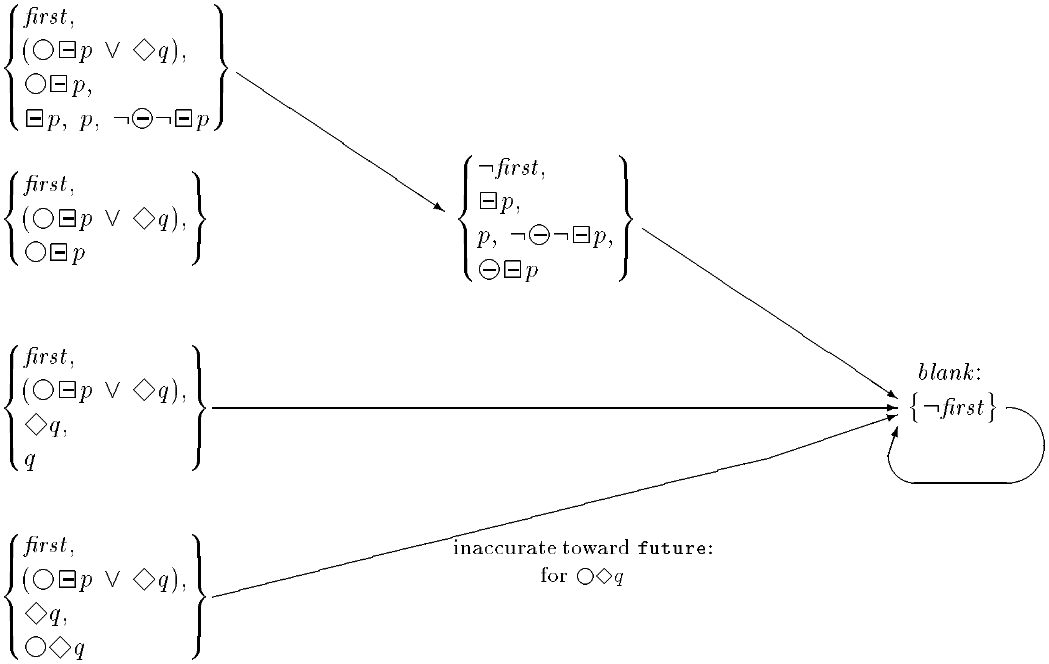
For example, suppose the formula $\varphi_{\text{checking}}$ is $(\bigcirc\boxminus p \vee \Diamond q)$. Then procedure `init_graph()` produces the following graph (with state-sketches, which are sets of formulas, as vertices). The three state-sketches containing the given formula $\varphi_{\text{checking}} = (\bigcirc\boxminus p \vee \Diamond q)$ correspond to the multiplicity of options for satisfying immediate constraints derived from this formula. The desired condition that some model $\mathcal{M}$ satisfies $\varphi_{\text{checking}}$ would actually have $\mathcal{M}$'s first state $\mathcal{M}@0 \models \varphi_{\text{checking}}$; the formula *first* which is included in each of these three state-sketches here reflects this (desired) condition. Then, with those state-sketches which contain *first* corresponding to possible first states of a model of $\varphi_{\text{checking}}$, the self-succeeding state-sketch *blank* corresponds to the sequence of as yet unconstrained states which follow any such first state.



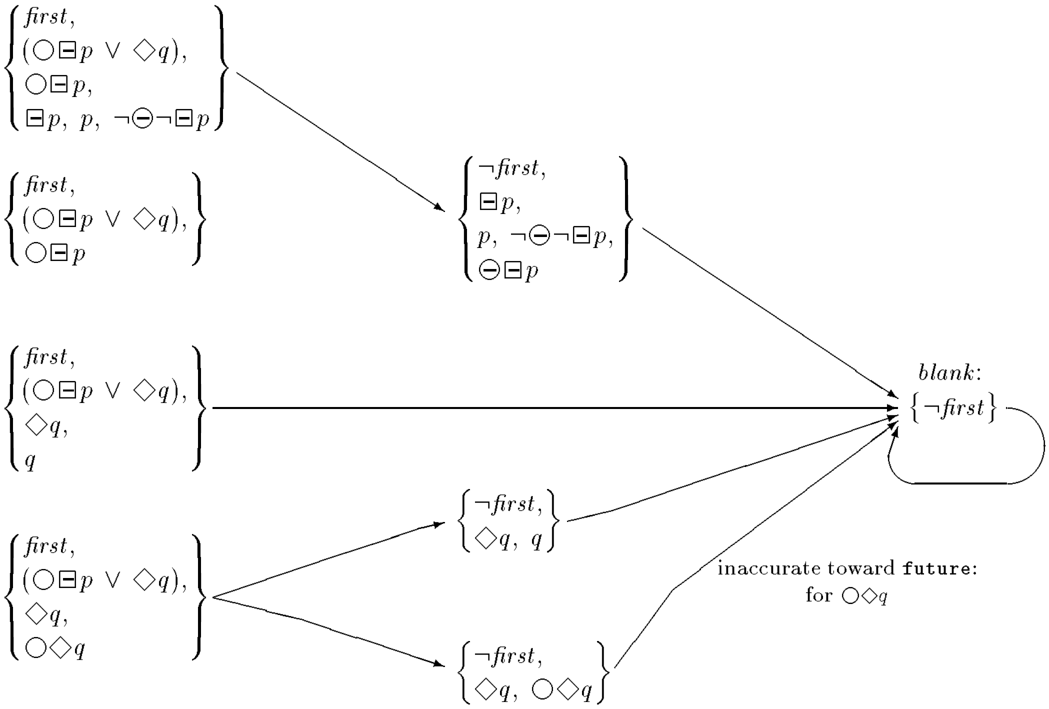
After execution of procedure `init_graph()`, processing continues with procedure `elaborate_graph()`, which addresses edges that are inaccurate in some direction. For example, here, the above diagram's topmost edge — from state-sketch s to *blank* — is inaccurate because s contains the formula $\bigcirc\boxminus p = \left(\bigcirc^{\langle 0+1\rangle}\boxminus p\right)$ whereas the successor-state-sketch *blank* does not contain the formula $\left(\bigcirc^{0}\boxminus p\right) = \boxminus p$. This situation gets corrected via replacement of the inaccurate edge by one connected to a new state-sketch $\widehat{n}$ which contains the desired formula $\boxminus p$. Formulas in addition to $\boxminus p$, e.g. p, are included in $\widehat{n}$ to satisfy immediate constraints. One such additional formula would be $\ominus\neg\neg\boxminus p$, but it is automatically simplified to $\ominus\boxminus p$. Some options are rejected, e.g. one that would include *first* in this state-sketch $\widehat{n}$ which has a predecessor (s) and which is therefore improperly placed for a 'first' state. The successor-states of $\widehat{n}$ are as yet unconstrained, so an edge from $\widehat{n}$ to *blank* is added. This processing yields the following graph:



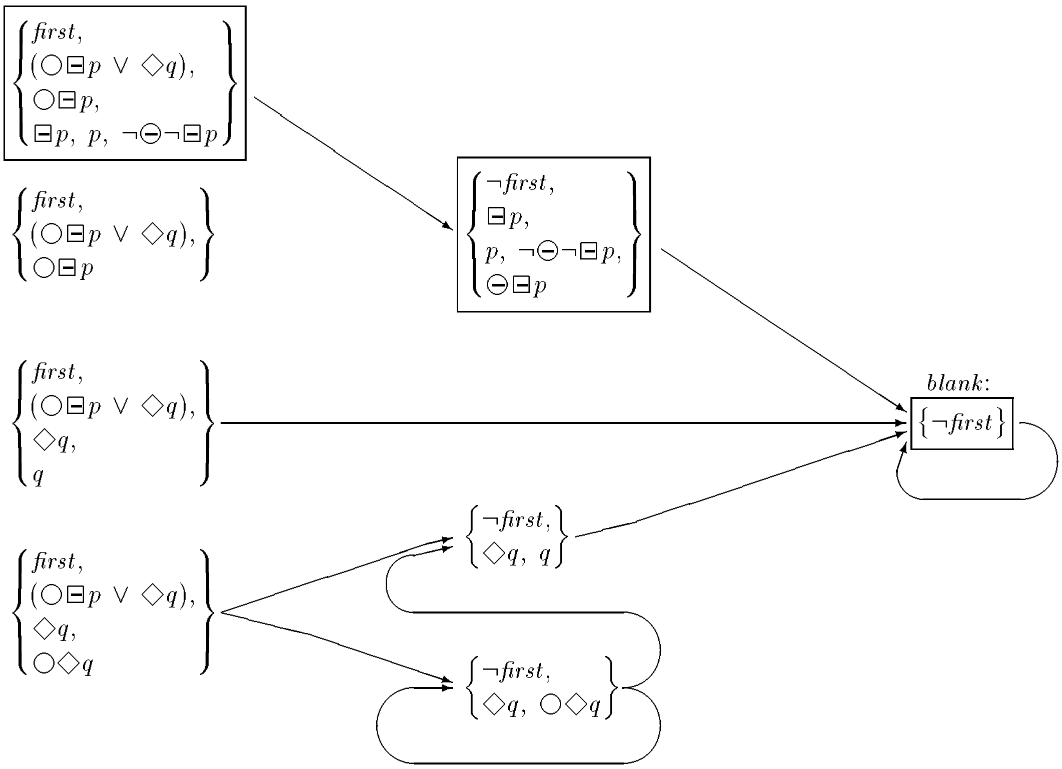
Unfortunately, the just-introduced edge from s to $\widehat{n}$ is inaccurate in the direction `past` because $\widehat{n}$ contains the formula $\ominus\boxminus p$ whereas s does not contain $\boxminus p$. As before, to satisfy the neighbor-constraint, processing creates a new state-sketch which contains the needed formula, plus additional formulas to satisfy immediate constraints — plus the formulas of the original state-sketch s.



Next, processing the only inaccurate edge here produces the following graph. The multiplicity of options in this case engenders the creation of two new state-sketches.



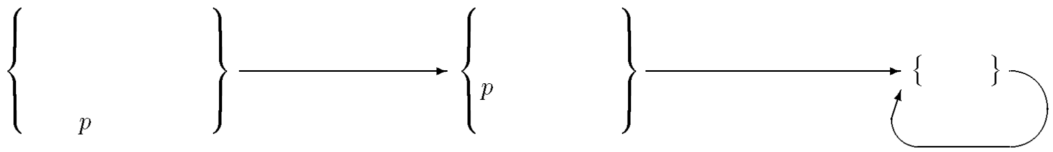
Processing the remaining inaccurate edge produces the following graph:



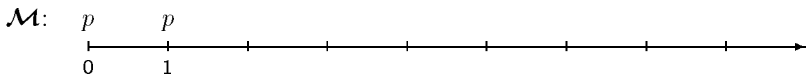
(See the following re the boxes here.)

That is the final graph.

To conclude the algorithm, procedure `search_graph()` searches this final graph for an infinite path that can be used to delineate a model. One such path here is along the top edges, i.e. along the state-sketches that are highlighted with boxes. A model derived from this path is delineated simply by the literals in the state-sketches; here, they are:



Thus, such a model — say, $\mathcal{M}$ — is determined by having its states satisfying proposition p at time 0, satisfying p again at time 1, and otherwise arbitrary (i.e. assignments to p at times 2ff. are arbitrary, and assignments to q at all times are arbitrary):



Such a model does indeed satisfy the originally given formula $\varphi_{\text{checking}} = (\bigcirc\boxminus p \vee \Diamond q)$.

I.1.3 The Algorithm

Here again is the algorithm's main procedure:

> `satisfy`(formula $\varphi_{\text{checking}}$):
>
> `init_graph();`
>
> `elaborate_graph();`
>
> `search_graph().`

The lower-level (pseudo-)code which is below employs the following elements (and notation):

- The variable V is a set containing the vertices of the graph being constructed.
- The variable E is a set containing the edges of the graph.
- The variable $\overline{E}$ is a set containing edges which have been discarded from E but which must be recorded to avoid infinitely repeated reintroduction of them.
- There is a special state-sketch *blank* which represents the as yet unconstrained future of some state-sketches; this state-sketch *blank* is the singleton set of formulas $\{\neg \mathit{first}\}$.
- An edge from a state-sketch s_1 to a state-sketch s_2 is denoted by the expression "$\langle s_1, s_2 \rangle$"; this edge may also be specified as "$\langle s_1, s_2 \rangle_{\texttt{future}}$" or "$\langle s_2, s_1 \rangle_{\texttt{past}}$".

Construction of the Graph

The initializing procedure is:

> `init_graph():`
>
> blank `:=` $\{\neg \mathit{first}\}$;
>
> V `:= add_options*`$\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}\big)$ `+` blank; (∗ "+": add element to set ∗)
>
> E `:=` $\{\}$;
>
> for each state-sketch $s \in V$:
>
> E `+=` the edge $\langle s, \mathit{blank}\rangle$; (∗ i.e. "$E$ `:=` E + $\langle s, \mathit{blank}\rangle$" ∗)
>
> $\overline{E}$ `:=` $\{\}$.

(The function `add_options*()` is below.)

Next, the central procedure is:

```
elaborate_graph():
    for each edge ⟨s, n⟩_dir ∈ E which is inaccurate — in direction dir:
        N̂ := add_options*(n, neighbor_formulas(s, dir));
        for each state-sketch n̂ ∈ N̂:
            establish_edge(s, n̂, dir);
            if n = blank:
                establish_edge(n̂, blank, future);
            else:
                for each state-sketch n̈ such that the edge ⟨n, n̈⟩_dir ∈ E ∪ Ē:
                    establish_edge(n̂, n̈, dir);
        V := V ∪ N̂;
        Ē += ⟨s, n⟩_dir;
        E -= ⟨s, n⟩_dir.        (* "-=": remove element from set *)
```

The function-call `neighbor_formulas(s, dir)` here yields:

$$\left\{\text{all the formulas } \left([\mathit{neighbor\_op}(\mathit{dir})]^{\eta}\,\psi\right) \text{ such that } \left([\mathit{neighbor\_op}(\mathit{dir})]^{\langle \eta+1\rangle}\,\psi\right) \in s\right\},$$

where $\mathit{neighbor\_op}$(`future`) is "◯" and $\mathit{neighbor\_op}$(`past`) is "⊖".

A subfunction used in both `init_graph()` and `elaborate_graph()` is:

```
add_options*(sets of formulas base, additions): set of state-sketches:
    if additions = {}:
        return {base};
    if the formula false ∈ additions:
        return {};
    (* otherwise: *)
    let the formulas φ_1, φ_2, ..., and φ_k be the elements of additions;
    options_combinations := options(φ_1) × options(φ_2) × ··· × options(φ_k);
    intermediates := { all the sets f = (base ∪ additions ∪ o_1 ∪ o_2 ∪ ··· ∪ o_k) such
                       that (o_1, o_2, ..., o_k) ∈ options_combinations and no obviously
                       contradictory pair of formulas {ψ, ¬ψ} ⊆ f };
    results := {};
    for each set of formulas f ∈ intermediates:
        results += add_options*(f, f − (base ∪ additions));
    return results.
```

This function generates state-sketches by combining *base* with *additions* while obeying immediate constraints which require additional formulas — which may in turn recursively require additional formulas, which may in turn .... All combinations of options which are superficially consistent are included. For example, the result of the function-call `add_options*`$\big(\{\neg p\},\ \{\Diamond p \vee q\}\big)$ is the set of state-sketches (i.e. set of sets of formulas) $\big\{\{\neg p,\ (\Diamond p \vee q),\ \Diamond p,\ \bigcirc\Diamond p\},\ \{\neg p,\ (\Diamond p \vee q),\ q\}\big\}$.

The subfunction `options(`φ`)` used here returns the set of sets of formulas comprising the options of additional formulas required for the immediate constraints of formula φ, as listed in Table I.1. For examples: `options(`p`)` is $\big\{\{\}\big\}$, `options(`$\Box p$`)` is $\big\{\{p,\ \bigcirc\Box p\}\big\}$, and `options(`$p \mathsf{U} q$`)` is $\big\{\{q\},\ \{p, \bigcirc(p \mathsf{U} q)\}\big\}$.

Another subprocedure (of `elaborate_graph()`) is:

> **`establish_edge`**(state-sketches $\widehat{n}$, $\ddot{n}$; direction dir)**:**
>
> if the edge $\langle \widehat{n}, \ddot{n}\rangle_{dir} \notin E \cup \overline{E}$:
>
> E **+=** $\langle \widehat{n}, \ddot{n}\rangle_{dir}$;
>
> **`unbar_edges`**`(`$\widehat{n}$, dir`);`
>
> **`unbar_edges`**$\big($$\ddot{n}$, $reverse(dir)$$\big)$.

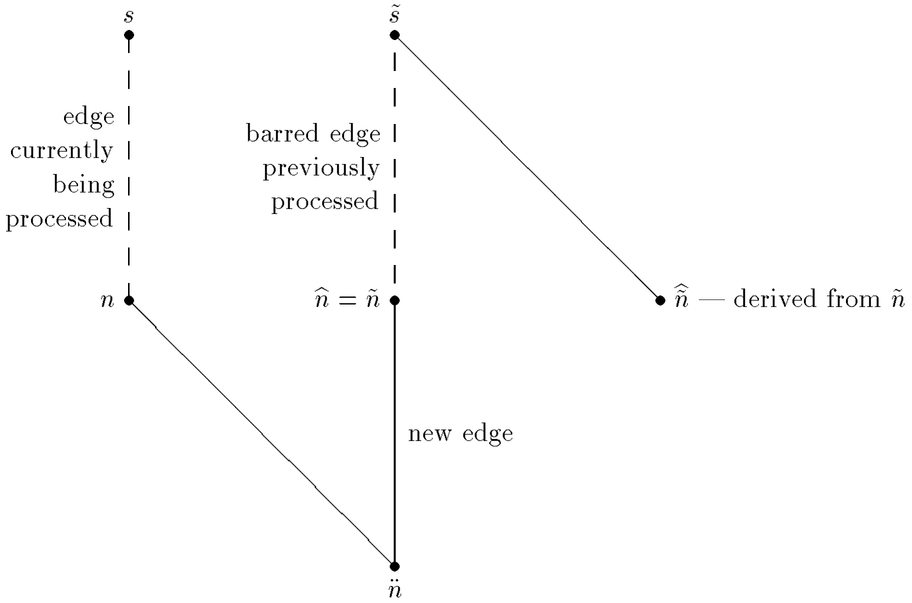
This procedure creates the edge $\langle \widehat{n}, \ddot{n}\rangle_{dir}$ if it does not already exist. The expression "$reverse(dir)$" naturally denotes **`past`** if $dir =$ **`future`** or **`future`** if $dir =$ **`past`**.

The subprocedure `unbar_edges()` is as follows:

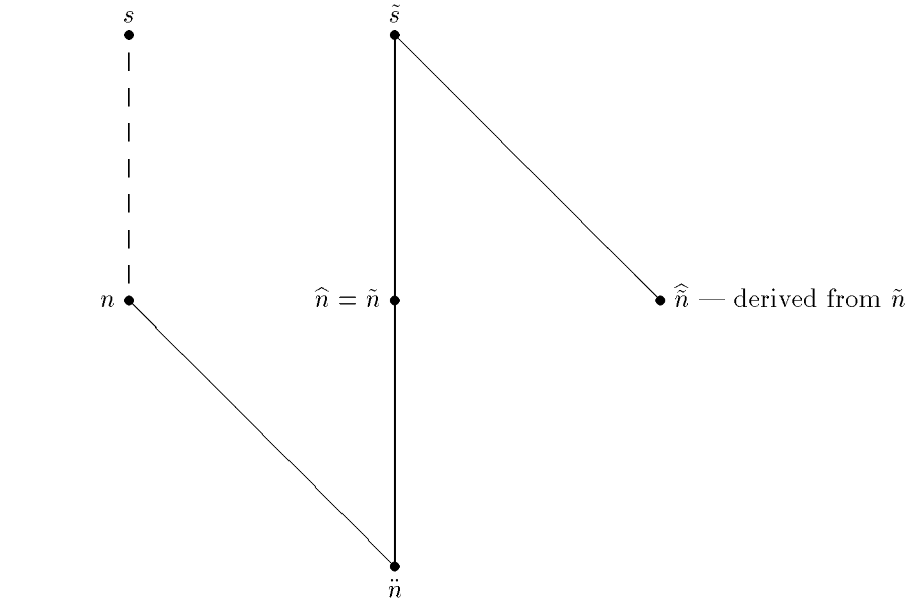
> **`unbar_edges`**(state-sketch $\tilde{n}$, direction dir)**:**
>
> for each $\tilde{s} \in V$ such that the edge $\langle \tilde{s}, \tilde{n}\rangle_{dir} \in \overline{E}$ and $\langle \tilde{s}, \tilde{n}\rangle_{dir}$ is inaccurate toward dir:
>
> E **+=** $\langle \tilde{s}, \tilde{n}\rangle_{dir}$;
>
> $\overline{E}$ **-=** $\langle \tilde{s}, \tilde{n}\rangle_{dir}$.

This procedure moves from $\overline{E}$ back to E certain edges that have vertex $\tilde{n}$ as an endpoint. This operation is necessary when an edge $\langle \widehat{n}, \ddot{n}\rangle_{dir}$ is new whereas previous processing duplicated vertex $\widehat{n}$'s then extant edges — which didn't include $\langle \widehat{n}, \ddot{n}\rangle_{dir}$ — to some vertex '$\widehat{\tilde{n}}$'; moving edges from $\overline{E}$ back to E causes reprocessing, so the new edge $\langle \widehat{n}, \ddot{n}\rangle_{dir}$ can be duplicated to $\widehat{\tilde{n}}$. The following diagram depicts this situation: The edge $\langle \tilde{s}, \tilde{n}\rangle_{dir}$ is inaccurate in the direction dir and it was previously processed; at that occasion, the state-sketch $\widehat{\tilde{n}}$ and the edge $\langle \tilde{s}, \widehat{\tilde{n}}\rangle_{dir}$ were created, and the edge $\langle \tilde{s}, \tilde{n}\rangle_{dir}$ was removed; the edge $\langle s, n\rangle_{dir}$ is inaccurate in the direction dir, and it is currently being processed; at this occasion, the state-sketch $\widehat{n} = \tilde{n}$ is found; and because the edge $\langle n, \ddot{n}\rangle_{dir}$ is present, the edge $\langle \widehat{n}, \ddot{n}\rangle_{dir}$

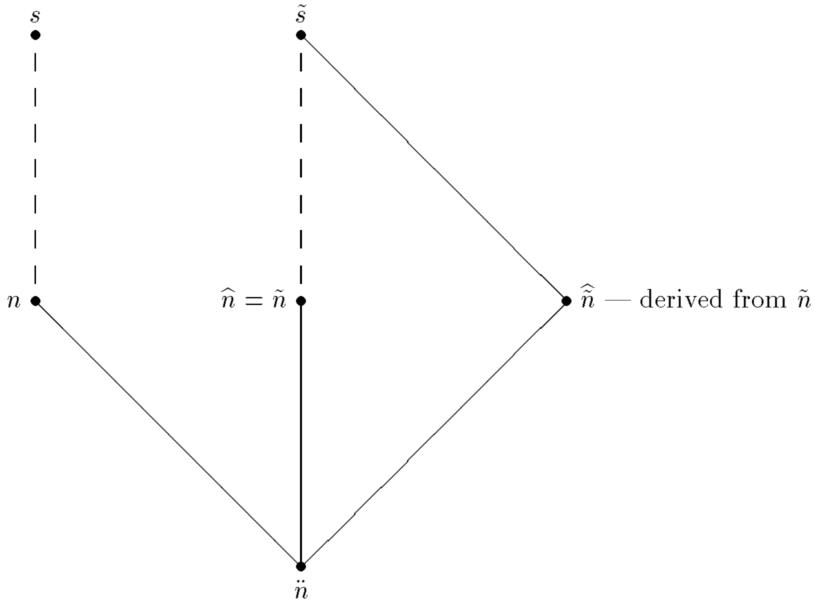
is created. (The direction *dir* is downward.) Notice how the graph is somewhat disconnected:



The function `unbar_edges()` rectifies this situation by reinstating the barred edge $\langle \tilde{s}, \tilde{n} \rangle_{dir}$:



After this reinstatement, $\langle \tilde{s}, \tilde{n} \rangle_{dir}$ will be reprocessed, engendering the edge $\langle \widehat{\widehat{n}}, \ddot{n} \rangle_{dir}$, as desired:



A formula $\varphi_{\text{checking}}$ whose processing can illustrate this situation is:

$$\bigcirc[\bigcirc\ominus(p \wedge \bigcirc\ominus\ominus p)] \vee \big[\neg p \wedge \bigcirc\big([\bigcirc\ominus(p \wedge \bigcirc\ominus\ominus p) \wedge p] \vee [\bigcirc\ominus(p \wedge \bigcirc\ominus\ominus p) \wedge p \wedge \bigcirc\ominus\ominus p]\big)\big]$$

The situation arises if edges are processed in a certain order.

Conclusion of the Algorithm

The final procedure is:

```
search_graph():
```

for each subset $mscc \subseteq V$ comprising a maximal strongly connected component:

let the set of formulas $f := \bigcup_{s \in mscc} s$;

for each formula having the form $\Diamond\psi$ or $(\xi \mathsf{U} \psi) \in f$:

if this $\psi \notin f$:

stop processing the current *mscc* (and start processing the next one, if any);

if any formula having the form $(\Diamond_{\cong\eta} \psi)$ or $(\xi \mathsf{U}_{\cong\eta} \psi) \in f$:

if `fulfilling_congruences(`*mscc*`)` returns `false`:

stop processing the current *mscc* (and start processing the next one, if any);

(∗ The current *mscc* has passed the tests here. ∗)

report that the originally given formula $\varphi_{\text{checking}}$ is satisfiable;

`delineate_model(`*mscc*`);`

terminate processing.

(∗ if the preceding loop exhausts all *mscc*s without finding a satisfactory one: ∗)

report that the originally given formula $\varphi_{\text{checking}}$ is unsatisfiable.

In this procedure, maximal strongly connected components (*mscc*s) are found via a textbook algorithm as in the work of Aho, Hopcroft, and Ullman [AhHU74].[5] Incidentally, the algorithm that finds the *mscc* produces a path from a state-sketch that contains *first* to the *mscc*.

The subfunction `fulfilling_congruences(`*mscc*`)` checks whether formulas having the form $(\Diamond_{\cong\eta} \psi)$ or $(\xi \mathsf{U}_{\cong\eta} \psi) \in f$ are properly fulfilled in *mscc*; suppose there are k such formulas $\varphi_1, \varphi_2, \ldots,$ and φ_k, each of which is either $(\Diamond_{\cong\eta_i} \psi_i)$ or $(\xi_i \mathsf{U}_{\cong\eta_i} \psi_i)$. An array of k sets of numbers is attached to each element of *mscc*; for each $s \in mscc$, element i of this array attached to s is referenced as $s.reachable[i]$.[6] As *mscc* is found via a path which ends at some state-sketch in *mscc*, this state-sketch is labeled "$s_{\bar{0}}$". Then, for each vertex $s \in mscc$, the set $s.reachable[i]$ will record each number $m \in \{0..(\eta_i - 1)\}$ such that there is a path P (within *mscc*) from vertex $s_{\bar{0}}$ to vertex s where the path's length $|P|$ is such that $|P|\ MOD\ \eta_i = m$. This information is obtained by exhaustively searching *mscc*, starting at vertex $s_{\bar{0}}$, with an array *branch_reached*[] of k numbers initialized to zeroes. At each step of this search, at a vertex s, if each element $branch\_reached[i] \in s.reachable[i]$, then this branch of

[5] But if an *mscc* is a singleton-set $\{s\}$ and the edge $\langle s, s\rangle \notin E$, then processing of this *mscc* is stopped (and processing of the next *mscc*, if any, is started).

[6] While it's specified here that the arrays *s.reachable*[] and *branch_reached*[] each comprise k elements, actually fewer elements can be used since duplicate η_is would engender duplicate processing. (But each of the arrays *needing*[] and *fulfilling*[] does require k elements.)

the search halts; otherwise, each set *s.reachable*$[i]$ `+=` *branch_reached*$[i]$, each element *branch_reached*$[i]$ is incremented modulo η_i, and with these new values in the array *branch_reached*[], the search recurses at each of vertex s's successors in *mscc*. Two 'global' arrays *needing*[] and *fulfilling*[], each comprising k sets of numbers, are also used. When a branch of the search finds $\varphi_i \in s$, the operation *needing*$[i]$ `+=` *branch_reached*$[i]$ is performed, and similarly for ψ_i and *fulfilling*$[i]$. After the search, if for any index $i \in \{1..k\}$, *needing*$[i] \not\subseteq$ *fulfilling*$[i]$, then subfunction `fulfilling_congruences()` returns `false`; otherwise, it returns `true`.

The subprocedure `delineate_model(`*mscc*`)` is straightforward: A cycle throughout *mscc* (i.e. a cycle which includes each vertex of *mscc* at least once) can be generated since *mscc* is strongly connected. Then the initial path to *mscc* plus infinite repetition of this cycle yields an infinite path. (Naturally, only one pass through the cycle is actually displayed.) The model's states are delineated by the literals that are contained in the state-sketches along this path.

If congruences are salient, the information that function `fulfilling_congruences()` has gathered affects the processing in procedure `delineate_model()`: While generating a cycle through *mscc*, starting at vertex $s_{\bar{0}}$, the search for the cycle is modified to require occurrences of state-sketches which contain fulfilling formulas at needed lengths (modulo the relevant η_i) of the path along the cycle; this restriction is done by each branch of the search for the cycle again using an array *branch_reached*[] but also getting a copy *branch_needing*[] of the previously constructed array *needing*[], and then removing fulfilled numbers from elements of array *branch_needing*[] as they are reached. (During this search, at a vertex s, if $\psi_i \in s$ then *branch_reached*$[i]$ is a 'fulfilled number'.)

Chapter I.2

Correctness of the Algorithm

A simple statement of the correctness of the algorithm here is:

> *Theorem* (Correctness): The algorithm here reports that a formula is satisfiable if and only if the formula really is satisfiable.

But this statement can be made more precise; plus, further claims hold. First:

> *Theorem 0* (Termination): Regardless of its input, whether satisfiable or unsatisfiable, this algorithm eventually finishes its processing.

This Termination theorem establishes that the algorithm here is a 'decision-algorithm'. The correctness of such a decision is guaranteed by the Correctness theorem — or, actually, theorem*s*, for Correctness is divided into two halves here. First:

> *Theorem 1* (Success): The algorithm here reports that a formula is satisfiable if the formula really is satisfiable.

The other half of the Correctness theorem is that the algorithm here reports that a formula is satisfiable only if the formula really is satisfiable. This statement is reformulated as follows:

> *Theorem 2* (Sufficiency): If the algorithm here reports that a formula is satisfiable, then the formula really is satisfiable.

The following sections provide proofs of these theorems.

I.2.0 Termination

Theorem 0 (Termination): Regardless of its input-formula $\varphi_{\text{checking}}$, whether satisfiable or unsatisfiable, procedure `satisfy()` eventually finishes its processing.

Proof: The first part of this proof shows that the graph that is being constructed is finitely bounded. Then, the process of adding new elements to it must be limited. A final consideration is that no operations should be infinitely repeatable.

The graph comprises vertices V and edges E — as well as $\overline{E}$. The edges $E \cup \overline{E}$ are ordered pairs of vertices in V, i.e. $E \cup \overline{E} \subseteq V \times V$, so if V is finite, then so is $E \cup \overline{E}$. The vertices V are state-sketches, which are sets of formulas; if f is the collection of all these formulas, then $V \subseteq \wp(f)$. Then, if f is finite, so is V, and hence so is the graph. The following Lemma shows that the collection of all the formulas (f) is finite:

Lemma 0.1: The collection of all the formulas appearing in state-sketches is finite.

Proof: Showing the finiteness of the collection of all the formulas that appear in state-sketches requires considering the algorithm's generation of formulas. Well, to begin, formula $\varphi_{\text{checking}}$ is given. Procedure `init_graph()` introduces the formulas $\neg$*first* and *first*. Procedure `elaborate_graph()` generates formulas specified as "$([neighbor\_op(dir)]^{\eta}\ \psi)$", which are really "$(\bigcirc^{\eta}\ \psi)$" or "$(\ominus^{\eta}\ \psi)$", given previously extant formulas $(\bigcirc^{\langle\eta+1\rangle}\ \psi)$ and $(\ominus^{\langle\eta+1\rangle}\ \psi)$, respectively. And lastly, function `options(`φ`)` (used in procedure `add_options*()`) generates formulas according to Table I.1's entry for formula φ.

If a formula φ has the form $([neighbor\_op(dir)]^{\langle\eta+1\rangle}\ \psi)$ (for some direction *dir*), then let the expression $neighbor\_op\_stripped(\varphi)$ denote the formula $([neighbor\_op(dir)]^{\eta}\ \psi)$. Then, for any formula φ, let the expression $top\_associated\_formulas(\varphi)$ denote the set of formulas comprising $neighbor\_op\_stripped(\varphi)$ — if applicable — plus all the formulas appearing in `options(`φ`)`.[1] Then, the generation of formulas can be summarized via the following recursively specified set:

$$associated\_formulas(\varphi) \;=\; \{\varphi, \neg first,\ first\} \;\cup \bigcup_{\psi \in t\_a\_f(\varphi)} associated\_formulas(\psi)\,,$$

where the expression "$t\_a\_f(\varphi)$" is an abbreviation for "$top\_associated\_formulas(\varphi)$".[2] What's of interest here is the finiteness of the set $associated\_formulas(\varphi_{\text{checking}})$; this desired fact follows as an instance of the finiteness of $associated\_formulas(\ldots)$ generally, which is shown in the following.

Unfortunately, naively using that definition of $associated\_formulas()$ would engender infinite recursion in some cases: e.g. the set $associated\_formulas(\Box p)$ would be constructed from the set $associated\_formulas(\bigcirc\Box p)$, but that in turn would be constructed from the set $associated\_formulas(\Box p)$. The solution here is to analyze the 'construction' of the set $associated\_formulas(\varphi)$ for each case of formula φ, showing that the set $associated\_formulas(\varphi)$ comprises a finite 'top-level' collection of formulas plus a few sets $associated\_formulas(\psi_i)$ with each formula ψ_i containing strictly fewer symbols than formula φ; then, induction on the sizes of formulas applies to the construction of the set $associated\_formulas(\varphi)$.

[1] For examples: $top\_associated\_formulas(false)$ is $\{\}$, $top\_associated\_formulas(p)$ is $\{\}$, $top\_associated\_formulas(\bigcirc p)$ is $\{p\}$, $top\_associated\_formulas(\Box p)$ is $\{p,\ \bigcirc\Box p\}$, and $top\_associated\_formulas(p\ \mathsf{U}\ q)$ is $\{q, p, \bigcirc(p\ \mathsf{U}\ q)\}$.

[2] The concept $associated\_formulas()$ here derives from the concept "closure" of [FiscL77]; see also the work of Gough [Gou84] and Barringer, Fisher, and Gough [BaFG89] re efficiency-restrictions similar to those employed here.

The base-cases of formula φ for this analysis are *false*, $\neg$*false*, the literals (p, $\neg p$, q, $\neg q$, etc.), *first*, and $\neg$*first*. For each such formula φ, the set *top_associated_formulas*(φ) is empty, so the set *associated_formulas*(φ) certainly comprises a finite top-level collection of formulas (φ itself, $\neg$*first*, and *first*) plus a few (here, zero) sets *associated_formulas*(ψ_i) with each ψ_i containing strictly fewer symbols than formula φ.

The next most tractable cases of formula φ are formulas whose outermost operators are the boolean operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\otimes$, or negations of such; each of these cases reduces easily. For example, applying the definition, *associated_formulas*$(\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$ comprises a finite top-level collection of formulas — $(\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$ itself, $\neg$*first*, and *first* — plus *associated_formulas*(ψ_i) for each $i \in \{1..k\}$, and each formula ψ_i contains strictly fewer symbols than the formula $(\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$. For another example, *associated_formulas*$\big(\neg(\psi \Rightarrow \xi)\big)$ comprises a finite top-level collection of formulas — $\neg(\psi \Rightarrow \xi)$ itself, $\neg$*first*, and *first* — plus *associated_formulas*(ψ) and *associated_formulas*$(\neg\xi)$, and each of the formulas ψ and $\neg\xi$ contains strictly fewer symbols than the formula $\neg(\psi \Rightarrow \xi)$.

The next group of cases of formula φ comprises formulas whose outermost operator is either $\Box$, $\Diamond$, U, A, $\diamondminus$, or S. For example, *associated_formulas*$(\Box\psi)$ comprises the finite top-level collection of formulas $\Box\psi$ (itself), $\neg$*first*, *first*, and $\bigcirc\Box\psi$, plus *associated_formulas*(ψ); and formula ψ contains strictly fewer symbols than the formula $\Box\psi$. Similarly, *associated_formulas*$(\xi \mathsf{S} \psi)$ comprises the finite top-level collection of formulas $(\xi \mathsf{S} \psi)$, $\neg$*first*, *first*, and $\ominus(\xi \mathsf{S} \psi)$, plus *associated_formulas*(ξ) and *associated_formulas*(ψ), and each of the formulas ξ and ψ contains strictly fewer symbols than the formula $(\xi \mathsf{S} \psi)$.

Next are formulas with $\boxminus$ or B. The set *associated_formulas*$(\boxminus\psi)$ comprises the finite top-level collection of formulas $\boxminus\psi$ (itself), $\neg$*first*, *first*, $\neg\ominus\neg\boxminus\psi$, and $\ominus\boxminus\psi$, plus *associated_formulas*(ψ); similarly for *associated_formulas*$(\xi \mathsf{B} \psi)$.

A negation $\varphi = \neg\widetilde{\varphi}$ where subformula $\widetilde{\varphi}$'s outermost operator is either $\Box$, $\Diamond$, $\boxminus$, or $\diamondminus$ reduces straightforwardly to a preceding case; for example, the set *associated_formulas*$(\neg\Box\psi)$ comprises $\neg\Box\psi$, $\neg$*first*, *first*, $\Diamond(\neg\psi)$, and $\bigcirc\Diamond(\neg\psi)$, plus *associated_formulas*$(\neg\psi)$, and formula $\neg\psi$ contains strictly fewer symbols than the formula $\neg\Box\psi$. An analogous situation holds with U, A, S, and B. For example, *associated_formulas*$\big(\neg(\xi \mathsf{U} \psi)\big)$ comprises $\neg(\xi \mathsf{U} \psi)$, $\neg$*first*, *first*, $\big(\neg\psi \mathsf{A} (\neg\xi \wedge \neg\psi)\big)$, $\bigcirc\big(\neg\psi \mathsf{A} (\neg\xi \wedge \neg\psi)\big)$, and $(\neg\xi \wedge \neg\psi)$, plus *associated_formulas*$(\neg\xi)$ and *associated_formulas*$(\neg\psi)$, and the formulas $\neg\xi$ and $\neg\psi$ each contain strictly fewer symbols than the formula $\neg(\xi \mathsf{U} \psi)$.

The cases for metric temporal operators with subscripts "$_{\leq\eta}$" and "$_{\cong\eta}$" resemble the preceding but with numbers of top-level formulas larger by approximately the value of η, $2{\cdot}\eta$, $3{\cdot}\eta$, $4{\cdot}\eta$, or $5{\cdot}\eta$, depending on the temporal operator. For example, the set *associated_formulas*$(\Box_{\leq\eta}\,\psi)$ comprises $(\Box_{\leq\eta}\,\psi)$ itself, $\neg$*first*, *first*, $\bigcirc(\Box_{\leq\langle\eta-1\rangle}\,\psi)$, $(\Box_{\leq\langle\eta-1\rangle}\,\psi)$, $\bigcirc(\Box_{\leq\langle\eta-2\rangle}\,\psi)$, $(\Box_{\leq\langle\eta-2\rangle}\,\psi)$, $\bigcirc(\Box_{\leq\langle\eta-3\rangle}\,\psi)$,

..., and $(\Box_{\leq 0}\,\psi)$, plus *associated_formulas*(ψ).

Finally, the set *associated_formulas*(φ) when formula φ is $\left(\bigcirc^{\langle\eta+1\rangle}\psi\right)$ or $\left(\ominus^{\langle\eta+1\rangle}\psi\right)$ obviously comprises formula φ itself, $\neg$*first*, *first*, and η more formulas $(\bigcirc^{\tilde{\eta}}\psi)$ or $(\ominus^{\tilde{\eta}}\psi)$ with $\tilde{\eta} \in \{1..\eta\}$, plus *associated_formulas*(ψ), and formula ψ contains strictly fewer symbols than formula φ.

Perhaps disconcertingly, when φ is $\left(\bigcirc^{\langle\eta+1\rangle}\psi\right)$ or $\left(\ominus^{\langle\eta+1\rangle}\psi\right)$, the set *associated_formulas*(ψ) may involve φ again, e.g. if φ is $\bigcirc\Box p$ or $\left(\bigcirc^{2}(\Box_{\cong 5}\,p)\right)$; but the preceding discussion actually handles this circumstance: Regardless, *associated_formulas*(ψ) may require further *associated_formulas*(ψ_i) only with each formula ψ_i containing strictly fewer symbols than formula ψ, and then *associated_formulas*(ψ_i) may require further *associated_formulas*$(\psi_{i,i'})$ only with each formula $\psi_{i,i'}$ containing strictly fewer symbols than formula ψ_i, and so on. Some of these recursive steps may add duplicate formulas such as φ — and even more blatantly, $\neg$*first* and *first* — but only a finite collection of them at each step. And by induction on the sizes of formulas, the number of steps is finite; so the overall collection of formulas is finite.

(Lemma 0.1) ■

Lemma 0.1 concludes the part of the proof of Theorem 0 showing that the elements of the graph are finitely bounded; it's further necessary to show that the processing of the graph is finitely bounded.

Well, first, all of procedure `init_graph()`'s operations except the execution of function `add_options*()` are obviously finite. In that recursive function `add_options*(`*base*, *additions*`)`, after the variable *immediates* gets set, the situation is that for each element $f \in$ *intermediates*, *additions* $\subseteq f$ and *base* $\subseteq f$. Then, the formulas in this *additions* become part of the first argument *base* of every recursive sub-call `add_options*`$\big(f,\ f - (\mathit{base} \cup \mathit{additions})\big)$. Also, with the sub-call's second argument being $f - (\mathit{base} \cup \mathit{additions})$, every sub-call's second argument *additions* cannot contain any formula φ that was previously processed as a member of an 'ancestral' call's *additions*. Thus, in any chain of recursive calls of function `add_options*()`, no formula can appear in the second argument *additions* in more than one call. Since the pool of all formulas that are available is finite according to Lemma 0.1, chains of recursive calls of `add_options*()` are uniformly finitely bounded by the size of this pool — say, k. Further, at each occasion when `add_options*()` recursively calls itself, it does so only a finite number of times.[3] Then for any depth of recursive calls of `add_options*()`, the number of further (recursive) sub-calls started at that depth must be finite. Then, with zero calls possible at depth $k+1$, the number of all calls is the sum over depths 0

[3] This number is at most $|\mathit{options\_combinations}|$, which equals: $\prod_{\varphi_i \in \mathit{additions}} |\texttt{options}(\varphi_i)|$.

through k, so the number of recursive calls of `add_options*()` engendered by the initial call in procedure `init_graph()` is finite.

Next, procedure `elaborate_graph()`'s operations within each iteration of its main loop are similarly finite. Then the crucial question is: Is the number of iterations of the loop finite? Well, as discussed immediately preceding Lemma 0.1, it follows from Lemma 0.1 that the set of edges is finitely bounded. Unfortunately, it's not clear that the loop addresses each edge only a finite number of times, for procedure `unbar_edges()` engenders reprocessing of edges (moving them from $\overline{E}$ back to E, whereupon they are processed in procedure `elaborate_graph()` as inaccurate edges in E). Fortunately, `unbar_edges()` is executed — in procedure `establish_edge()` — only when a new edge is created. Since each edge can be created at most once, and since the number of edges is finitely bounded, procedure `unbar_edges()` is executed only finitely many times, so it engenders only a finite number of occurrences of reprocessing of edges. Then, the number of times the main loop addresses an edge is finite; and therefore, the amount of processing in procedure `elaborate_graph()` is finite.

Lastly, procedure `search_graph()`'s operations are clearly finite. (The number of maximal strongly connected components of the graph is bounded by the number of subsets of V, which is finite.)

Thus, all processing here is finite. (See also Chapter I.3.)

(Theorem 0: Termination) ■

I.2.1 Success

Theorem 1 (Success): Procedure `satisfy()` reports that its given (input-)formula $\varphi_{\text{checking}}$ is satisfiable if this formula really is satisfiable.

Proof:

First, note that if the formula $\varphi_{\text{checking}}$ is satisfiable, then some model $\boldsymbol{\mathcal{M}}$ satisfies it. With such a satisfying model available, the proof here uses the following definition:

An **embedding** of model $\boldsymbol{\mathcal{M}}$ in the graph comprising vertices V and edges E is a map $M : \boldsymbol{T} \to V$ such that:

0. The formula *first* $\in M[0]$.
1. For each time-value $\mathsf{t} \in \boldsymbol{T}$, the state $\boldsymbol{\mathcal{M}}$@t satisfies the set of formulas $M[\mathsf{t}]$.
2. For each time-value $\mathsf{t} \in \boldsymbol{T}$, the edge $\langle M[\mathsf{t}],\, M[\mathsf{t}+1]\rangle$ is in E.

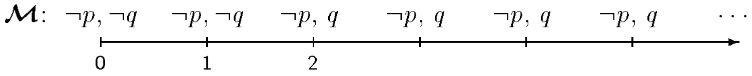
(Such an embedding in the final graph is exemplified by the boxes in the final graph at the end of Section I.1.2 above, on page 28; see also the diagrams on the next few pages.) When the algorithm's construction of the graph concludes, such an embedding provides an infinite path in the graph, considering this definition's condition 2; then, procedure `search_graph()` will find at least this path. With fulfillment-constraints also addressed in the embedding, satisfiability will be reported. The presumptions here are that an embedding exists and that fulfillment-constraints are satisfied; these presumptions are justified in the following.

The heart of this proof is the following:

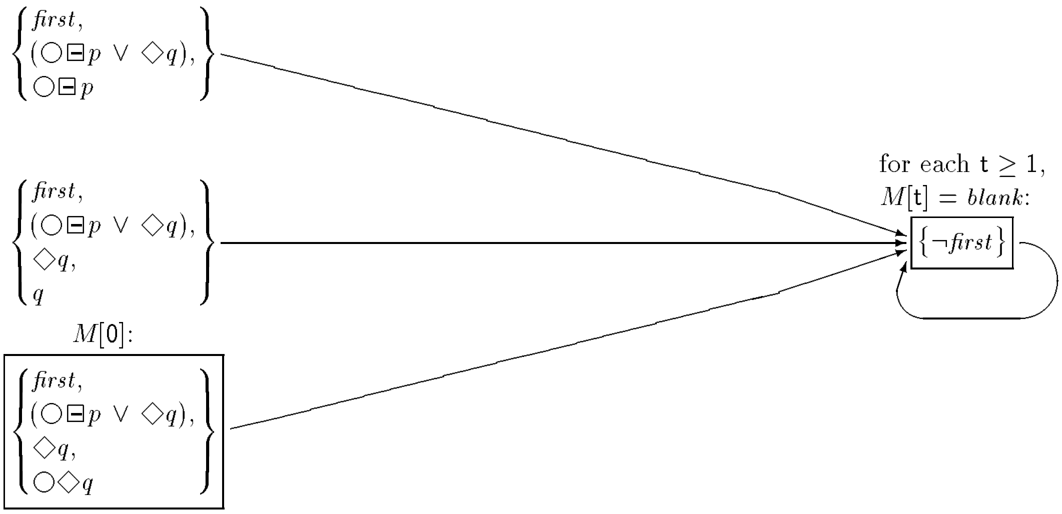
Lemma 1.1: At every stage of the algorithm's processing (after initialization), there's an embedding of model $\mathcal{M}$ in the graph.

Example

For an example, consider the example of the algorithm's processing in Section I.1.2 above (on pages 25–28), where the formula $\varphi_{\text{checking}}$ is $(\bigcirc\boxminus p \vee \Diamond q)$ — but with a different model here, to illustrate matters more thoroughly: here, let model $\mathcal{M}$ falsify p (i.e. satisfy $\neg p$) at all times, falsify q at times 0 and 1, and satisfy q at time 2 as well as at all times greater than 2:

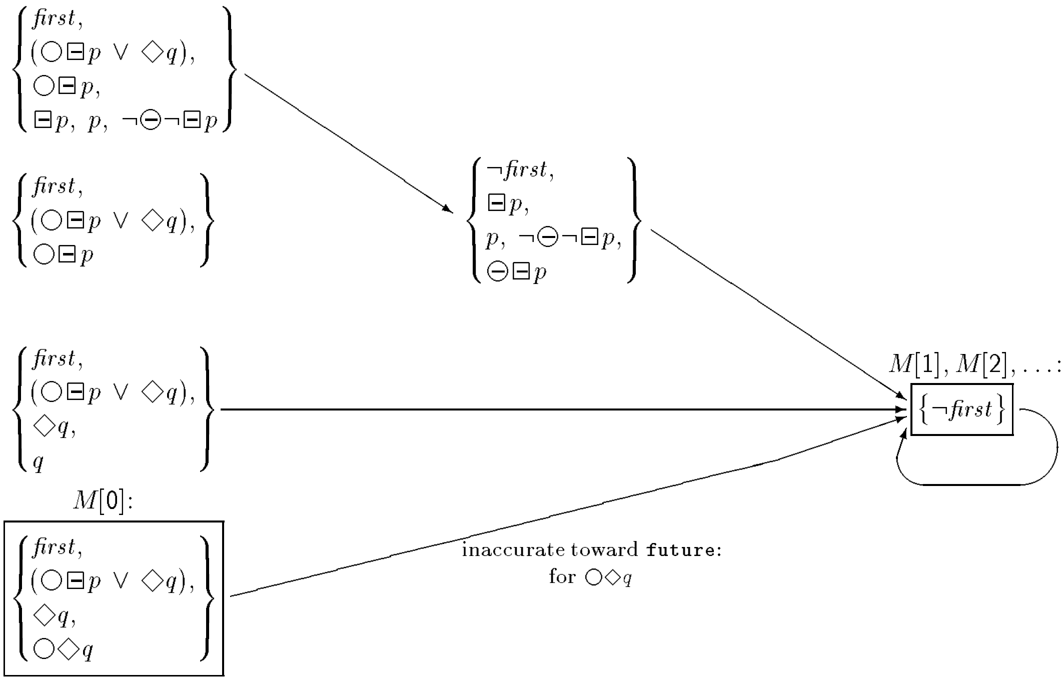


This model can be embedded in the initial graph constructed for $\varphi_{\text{checking}}$ as follows:

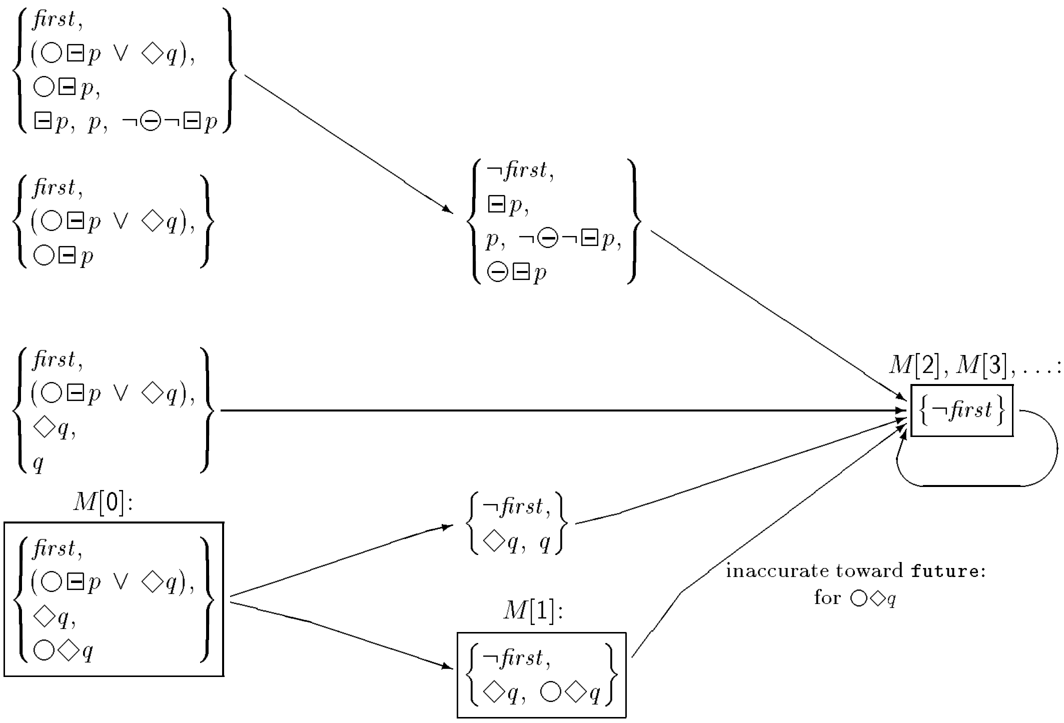


The conditions for an embedding do hold here: [0] the state-sketch $M[0]$ contains *first*; [1] for each time-value $\mathsf{t} = 0$ or $\mathsf{t} \geq 1$, the state $\mathcal{M}$@t satisfies $M[\mathsf{t}]$; and [2] for each $\mathsf{t} \in \boldsymbol{T}$, the edge $\langle M[\mathsf{t}], M[\mathsf{t}+1]\rangle$ is in E.

Naturally, the same embedding is used while the algorithm processes the top edges, until the following situation is reached:

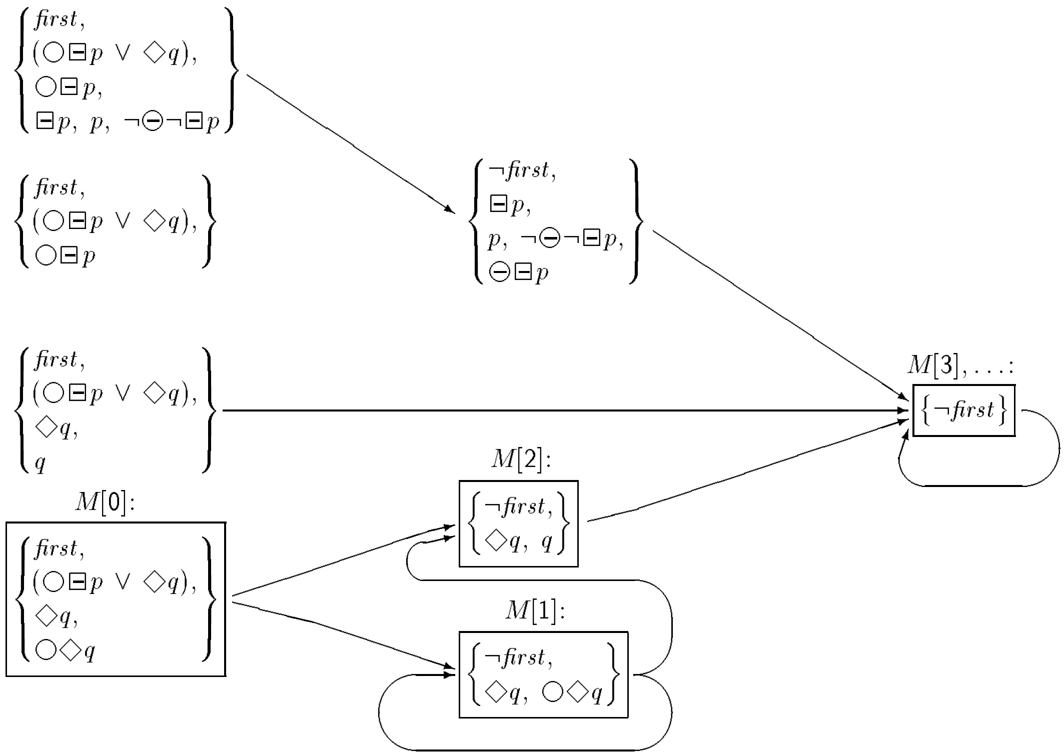


Then, as an edge used by the embedding is processed and removed from E, the embedding is updated:



In choosing between the state-sketches $\{\neg\mathit{first}, \Diamond q, q\}$ and $\{\neg\mathit{first}, \Diamond q, \bigcirc\Diamond q\}$ for the new value of $M[1]$, it's necessary to choose the latter state-sketch because $\mathcal{M}@1 \models \neg q$, which implies that $\mathcal{M}@1 \not\models \{\neg\mathit{first}, \Diamond q, q\}$.

Next, another edge used by the embedding is processed and removed from E, so again the embedding is updated:



Re choosing the new value of $M[2]$, the state $\boldsymbol{\mathcal{M}}$@2 satisfies both the state-sketch $\{\neg\mathit{first}, \Diamond q, q\}$ and the state-sketch $\{\neg\mathit{first}, \Diamond q, \bigcirc\Diamond q\}$. Then greediness to fulfill the formula $\Diamond q$ is applied: the fulfilling formula q is present in the state-sketch $\{\neg\mathit{first}, \Diamond q, q\}$ but not in the other state-sketch, so the former is chosen.

That is the final embedding.

Proof of Lemma 1.1:

The proof of the existence of an embedding at every stage of the algorithm's processing follows the scheme of the preceding example: there is an initial embedding of model $\boldsymbol{\mathcal{M}}$ in the initial graph, and subsequently, when the algorithm changes the graph, the embedding is updated. At initialization and each update of the embedding, fulfillment-constraints are addressed.

All this 'meta'-processing is expressed here as 'meta'-pseudo-code which uses $\boldsymbol{\mathcal{M}}$ as an 'oracle'. To have the meta-processing here occur at the proper moments during the real algorithm's processing, the new meta-code is simply added to procedure `satisfy()` (etc.) as follows:

First, a new procedure `init_embedding()` (which is below) is executed in procedure `satisfy()` immediately following execution of the original procedure `init_graph()`:

> `satisfy`(formula $\varphi_{\text{checking}}$)`:`
>
>   `init_graph();`
>
>   `init_embedding();` $\longleftarrow$
>
>   `elaborate_graph();`
>
>   `search_graph().`

Then, a new procedure `shift_embedding()` (which is also below) is executed at the end of (and inside) the main loop of procedure `elaborate_graph()`:

> `elaborate_graph():`
>
> $\ddots$
>
> E `-=` $\langle s, n\rangle_{dir}$;
>
>   `shift_embedding().` $\longleftarrow$

The first of these new procedures is, naturally:

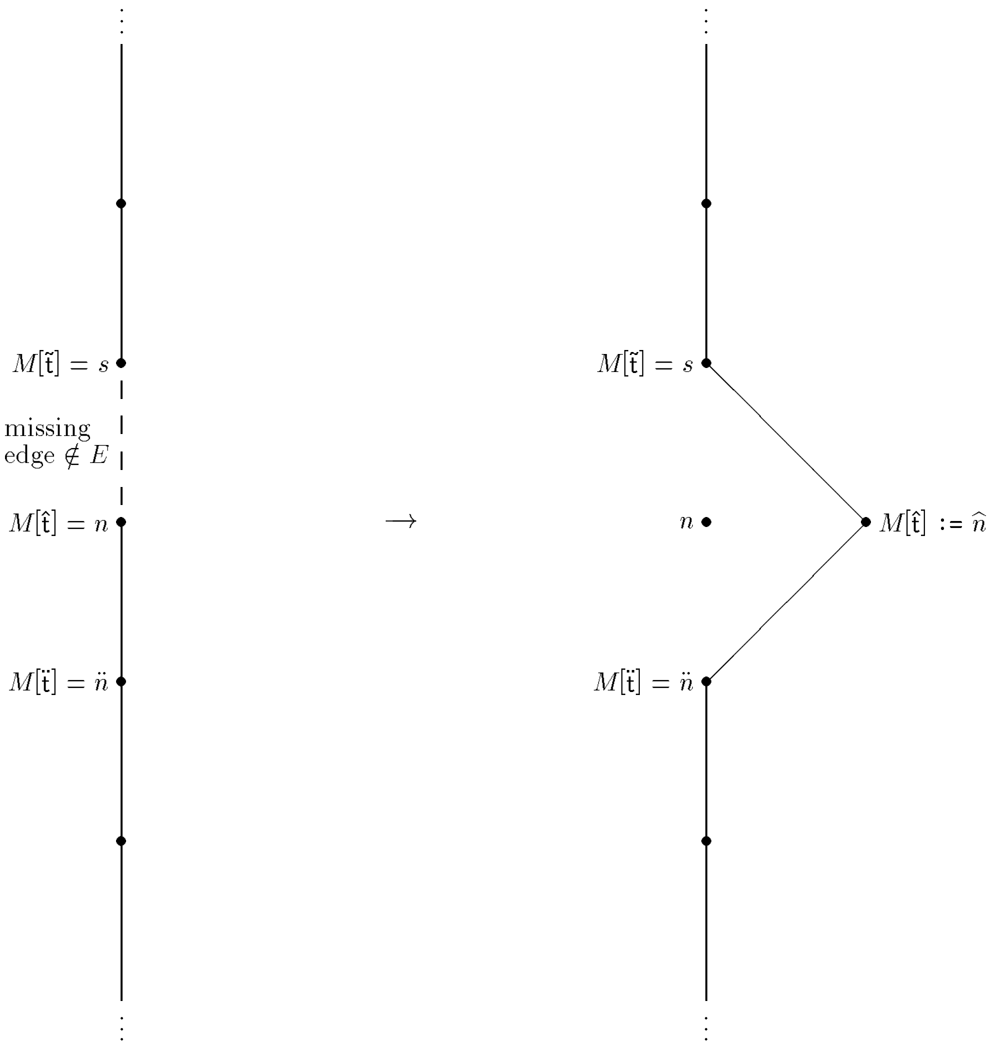
> `init_embedding():`
>
> $M[0]$ `:= selectively_add_options*`$\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}, 0\big)$;
>
>   for each time-value $\mathsf{t} \in \boldsymbol{T} - \mathsf{0}$:
>
> $M[\mathsf{t}]$ `:=` *blank*.

The purpose of this procedure should be obvious; its effect is exemplified by the initial embedding presented on page 41. (The subfunction `selectively_add_options*()` is below.)

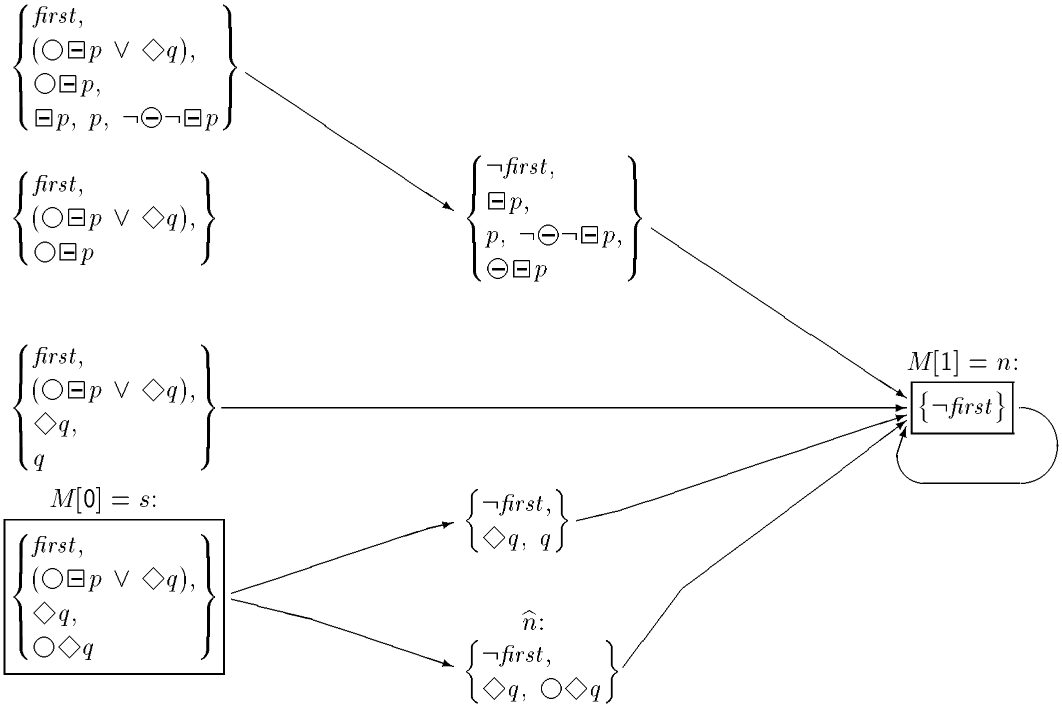
Next is:

> `shift_embedding():`
>
>   while there exists any time-value $\mathsf{t} \in \boldsymbol{T}$ such that the edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle \notin E$:
>
>   if the edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle$ is inaccurate in the direction `future`:
>
>   let dir `:= future`, let s `:=` $M[\mathsf{t}]$, let n `:=` $M[\mathsf{t}+1]$, and let $\hat{\mathsf{t}}$ `:=` $\mathsf{t}+1$;
>
>   else:
>
>   let dir `:= past`, let s `:=` $M[\mathsf{t}+1]$, let n `:=` $M[\mathsf{t}]$, and let $\hat{\mathsf{t}}$ `:=` t;
>
> $M[\hat{\mathsf{t}}]$ `:= selectively_add_options*`$\big(n$, `neighbor_formulas(`s, dir`)`, $\hat{\mathsf{t}}\big)$.

This procedure does the following, repeatedly: Wherever the embedding passes along an edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle$ which is missing from the set of edges E, the embedding is shifted to pass along a different edge by changing one of $M[\mathsf{t}]$ or $M[\mathsf{t}+1]$ ($M[\hat{\mathsf{t}}]$ is the one of $M[\mathsf{t}]$ or $M[\mathsf{t}+1]$ that is chosen to be changed). A general picture of such a shift is as follows; direction dir is downward, the path on the left is the embedding's path through the graph before the shift, and the path on the right is the embedding's path after the shift:



(If $\hat{\mathtt{t}} = \mathtt{t} + 1$, then $\tilde{\mathtt{t}} = \mathtt{t}$, else $\hat{\mathtt{t}} = \mathtt{t}$ and $\tilde{\mathtt{t}} = \mathtt{t} + 1$; "$\ddot{\mathtt{t}}$" etc. simply label further values.) For a specific example of such a shift, consider the occasion — during the sample processing just presented on the preceding pages — when the value $M[1]$ changes (as presented on page 42). At the precise moment when procedure `shift_embedding()` gets executed, the situation is actually as follows:



(The not shown values $M[2]$, $M[3]$, $M[4]$, etc. are equal to $\{\neg first\}$, i.e. the state-sketch *blank*.) Clearly, for time-value $\mathsf{t} = 0$, the edge $\langle M[\mathsf{t}], M[\mathsf{t}+1]\rangle \notin E$: it is equal to the edge $\langle s, n\rangle_{dir}$, currently being processed, which was just removed from E. The code in procedure `shift_embedding()` determines that the edge $\langle M[\mathsf{t}], M[\mathsf{t}+1]\rangle$ is inaccurate in the direction dir = `future`, so it uses the same dir, s, and n, and the variable $\hat{\mathsf{t}} := \mathsf{t}+1 = 0+1 = 1$. Then, $M[\hat{\mathsf{t}}]$ — i.e. $M[1]$ — is reset to $\widehat{n}$ = `selectively_add_options*(`n, `neighbor_formulas(`s, dir`)`, $\hat{\mathsf{t}}$`)`, yielding the situation depicted at the bottom of page 42.

A subfunction used in both `init_embedding()` and `shift_embedding()` is:

`selectively_add_options*`(sets of formulas *base*, *additions*; time-value $\hat{\mathfrak{t}}$): state-sketch:

if *additions* $= \{\}$:

return *base*;

let the formulas φ_1, φ_2, ..., and φ_k be the elements of *additions*;

$$\begin{aligned} \mathit{selected\_f} \ := \ & \mathit{base} \cup \mathit{additions} \\ & \cup \ \texttt{select\_option}(\varphi_1, \hat{\mathfrak{t}}) \\ & \cup \ \texttt{select\_option}(\varphi_2, \hat{\mathfrak{t}}) \\ & \ \vdots \\ & \cup \ \texttt{select\_option}(\varphi_k, \hat{\mathfrak{t}}); \end{aligned}$$

return `selectively_add_options*`$\big($*selected_f*, *selected_f* $-$ (*base* $\cup$ *additions*), $\hat{\mathfrak{t}}\big)$.

The (further) subfunction `select_option()` is as follows:

`select_option`(formula φ, time-value $\hat{\mathfrak{t}}$): set of formulas:

if φ has the form $\Diamond\psi$, $(\xi \mathsf{U} \psi)$, $(\Diamond_{\cong\eta}\, \psi)$, or $(\xi\ \mathsf{U}_{\cong\eta}\ \psi)$ and $\mathcal{M}@\hat{\mathfrak{t}} \models \psi$:

return $\{\psi\}$;

else:

return any element $o \in$ `options`(φ) such that $\mathcal{M}@\hat{\mathfrak{t}} \models o$.

A natural concern about these `selective` functions is: Can they fail? The following claim is used below to address this concern:

Claim 1.1.0: For any sets of formulas b and a and for any time-value $\hat{\mathfrak{t}} \in \boldsymbol{T}$, if $\mathcal{M}@\hat{\mathfrak{t}} \models b$ and $\mathcal{M}@\hat{\mathfrak{t}} \models a$, then the function-call `selectively_add_options*`$(b, a, \hat{\mathfrak{t}})$ succeeds; further, if state-sketch $\widehat{n}$ is the result, then $\widehat{n} \in$ `add_options*`(b, a) and $\mathcal{M}@\hat{\mathfrak{t}} \models \widehat{n}$.

Proof: This proof proceeds in an inductive fashion.

In the base-case, when $a = \{\}$, clearly the function-call `selectively_add_options*`$\big($*base* := b, *additions* := $a = \{\}$, $\hat{\mathfrak{t}}\big)$ succeeds, immediately returning *base* $= b$. Let $\widehat{n}$ = this result b. Then $\widehat{n} \in$ `add_options*`(b, a) because — considering that function's operations (on page 30) — the function-call `add_options*`$\big($*base* := b, *additions* := $a = \{\}\big)$ immediately returns $\{\mathit{base}\} = \{b\}$, and obviously $\widehat{n} = b \in \{b\}$. Lastly, $\mathcal{M}@\hat{\mathfrak{t}} \models \widehat{n} = b$ because, by supposition, $\mathcal{M}@\hat{\mathfrak{t}} \models b$.

Otherwise, suppose $a \neq \{\}$, and again consider the related function-call `add_options*`$\big($*base* := b, *additions* := $a = \{\}\big)$, line by line:

Lines 1–2: "if *additions* $= \{\}$, return ..."

By supposition in the current case, a = *additions* $\neq \{\}$. Consequently, `add_options*()` does not terminate here (in the current case).

Lines 3–4: "if the formula *false* $\in$ *additions*, return ..."

A premise of this claim is that $\mathcal{M}@\hat{\mathfrak{t}} \models a$, so by basic semantics, $\mathit{false} \notin a = \mathit{additions}$. Consequently, `add_options*()` does not terminate here either.

Line 5: "(∗ . . . ∗)"

This line is a comment.

Line 6: "let the formulas φ_1, φ_2, . . . , and φ_k be the elements of *additions*"

The same notation is used here.

Line 7: "$\mathit{options\_combinations}$:= `options(`φ_1`)` $\times$ `options(`φ_2`)` $\times \cdots \times$ `options(`φ_k`)`"

By supposition, $\mathcal{M}@\hat{\mathfrak{t}} \models a = \mathit{additions} = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$. Then for each $i \in \{1..k\}$, $\mathcal{M}@\hat{\mathfrak{t}} \models \varphi_i$. Then, by semantics and analysis of Table I.1, the state $\mathcal{M}@\hat{\mathfrak{t}}$ satisfies at least one of the options of Table I.1 for formula φ_i, i.e. $\mathcal{M}@\hat{\mathfrak{t}}$ satisfies at least one of the elements of `options(`φ_i`)`. Then in `selectively_add_options*()`, clearly the function-call `select_option(`φ_i, $\hat{\mathfrak{t}}$`)` will succeed in returning some option $o_i \in$ `options(`φ_i`)`.

Line 8: "$\mathit{intermediates}$:= {all the sets $f = (\mathit{base} \cup \mathit{additions} \cup o_1 \cup o_2 \cup \cdots \cup o_k)$ such that . . . }"

The variable $\mathit{selected\_f}$ in `selectively_add_options*()` clearly has `add_options*()`'s form $f = (\mathit{base} \cup \mathit{additions} \cup o_1 \cup o_2 \cup \cdots \cup o_k)$ such that $(o_1, o_2, \ldots, o_k) \in \mathit{options\_combinations}$. Further, considering the operations of `select_option()`, the state $\mathcal{M}@\hat{\mathfrak{t}}$ satisfies each o_i here. Also, by supposition, $\mathcal{M}@\hat{\mathfrak{t}} \models b = \mathit{base}$ and $\mathcal{M}@\hat{\mathfrak{t}} \models a = \mathit{additions}$. Then $\mathcal{M}@\hat{\mathfrak{t}} \models \mathit{selected\_f}$. Then, no contradictory pair of formulas $\{\psi, \neg\psi\} \subseteq \mathit{selected\_f}$, by basic semantics. Therefore, $\mathit{selected\_f} \in \mathit{intermediates}$.

Lines 9–12: ". . . *results* += `add_options*(`f, $f - (\mathit{base} \cup \mathit{additions})$`)`; return *results*"

By the preceding, one f used in this recursion in `add_options*()` is $\mathit{selected\_f}$. Consequently, the results of the current call `add_options*(`b, a`)` include the results of `add_options*(`$\mathit{selected\_f}$, $\mathit{selected\_f} - (\mathit{base} \cup \mathit{additions})$`)`. The matching recursive call in `selectively_add_options*()` is `selectively_add_options*(`$\mathit{selected\_f}$, $\mathit{selected\_f} - (\mathit{base} \cup \mathit{additions})$, $\hat{\mathfrak{t}}$`)`, which is equivalent to `selectively_add_options*(`$\mathit{selected\_f}$, $\mathit{selected\_f} - (b \cup a)$, $\hat{\mathfrak{t}}$`)`. By the preceding, $\mathcal{M}@\hat{\mathfrak{t}} \models \mathit{selected\_f}$, so this sub-call is a call `selectively_add_options*(`b', a', $\hat{\mathfrak{t}}$`)` with $\mathcal{M}@\hat{\mathfrak{t}} \models b'$ and $\mathcal{M}@\hat{\mathfrak{t}} \models a'$.

At this point in this proof of this claim, it is desirable to invoke inductive hypothesis for this sub-call `selectively_add_options*(`b', a', $\hat{\mathfrak{t}}$`)`. But what guarantees that this recursion will terminate? Well, the preceding discussion demonstrates that the operations in `selectively_add_options*()` follow one thread of the operations of function `add_options*()`. And termination of that function is guaranteed, as discussed in the proof of Theorem 0 (near the end of that proof, on page 39). Then the recursion here in function `selectively_add_options*()` is also guaranteed to terminate, so induction applies.

Then, by inductive hypothesis, the sub-call `selectively_add_options*`(b', a', $\hat{\mathsf{t}}$) succeeds, its result $\widehat{n} \in$ `add_options*`(b', a'), and $\mathcal{M}@\hat{\mathsf{t}} \models \widehat{n}$. Notice that the current call `selectively_add_options*`(b, a, $\hat{\mathsf{t}}$) simply returns that sub-result $\widehat{n}$ as the current result. Then, obviously, this current call succeeds. And this result $\widehat{n} \in$ `add_options*`(b, a) because $\widehat{n} \in$ `add_options*`(b', a') $=$ `add_options*`$\big(\mathit{selected\_f},\ \mathit{selected\_f} - (b \cup a)\big)$, and it's indicated above that the results of the current call `add_options*`(b, a) include the results of `add_options*`$\big(\mathit{selected\_f},\ \mathit{selected\_f} - (\mathit{base} \cup \mathit{additions})\big)$, which is actually `add_options*`$\big(\mathit{selected\_f},\ \mathit{selected\_f} - (b \cup a)\big)$. Lastly, the inductive hypothesis also gives $\mathcal{M}@\hat{\mathsf{t}} \models \widehat{n}$.

(Claim 1.1.0) ■

It's necessary to prove that the (meta-)pseudo-code here successfully constructs an embedding. This proof proceeds inductively through the processing, first proving that the map M constructed by procedure `init_embedding()` is an embedding and then proving that if M is an embedding just before execution of procedure `shift_embedding()`, then immediately afterward, the new map M is also an embedding.

Procedure `init_embedding()`

The goal here is to show that (immediately) after execution of procedure `init_embedding()`, the map M is an embedding.

∗. A fundamental condition is that M must be a map from $\boldsymbol{T}$ to V. Well, re $M[\mathsf{0}]$, procedure `init_embedding()` begins with the operation:

$$M[\mathsf{0}] \texttt{ := selectively\_add\_options*}\big(\{\mathit{first}\},\ \{\varphi_{\text{checking}}\},\ 0\big)\,.$$

This operation is done after procedure `init_graph()` has executed:

$$V \texttt{ := add\_options*}\big(\{\mathit{first}\},\ \{\varphi_{\text{checking}}\}\big) + \mathit{blank}\,.$$

By basic semantics, $\mathcal{M}@\mathsf{0} \models \mathit{first}$. By supposition, $\mathcal{M} \models \varphi_{\text{checking}}$, i.e. $\mathcal{M}@\mathsf{0} \models \varphi_{\text{checking}}$. Then by Claim 1.1.0, the function-call `selectively_add_options*`$\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}, 0\big)$ successfully returns a state-sketch $\widehat{n}$ such that $\widehat{n} \in$ `add_options*`$\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}\big)$ (and $\mathcal{M}@\mathsf{0} \models \widehat{n}$). Then since $V =$ `add_options*`$\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}\big) + \mathit{blank}$, $M[\mathsf{0}] = \widehat{n} \in V$. For each other time-value $\mathsf{t} \in \boldsymbol{T} - \{\mathsf{0}\}$, the value $M[\mathsf{t}] = \mathit{blank} \in V$. Thus, this fundamental condition is satisfied.

0. Condition 0 is that $M[\mathsf{0}]$ must contain the formula *first*. Well, in the recursive function `selectively_add_options*()`, *selected_f* `:=` *base* ∪ other stuff; and each sub-call's *base* `:=` the parent call's *selected_f*; and finally, at termination of the recursion, the result is

the terminating sub-call's *base*. Therefore, the result contains the original *base*. Then in `init_embedding()`'s assignment:

$$M[0] \texttt{ := selectively\_add\_options*}\big(base \texttt{ := } \{first\},\ \ldots\big),$$

the result must contain $\{first\}$. Thus $first \in M[0]$, as desired.

1. Condition 1 is that for each time-value $\mathsf{t} \in \boldsymbol{T}$, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models M[\mathsf{t}]$. Well, from the discussion in item '$*$' above, $\boldsymbol{\mathcal{M}}@0 \models M[0]$. For each other time-value $\mathsf{t} \in \boldsymbol{T} - \{0\}$, the value $M[\mathsf{t}] = blank = \{\neg first\}$, so by semantics $\boldsymbol{\mathcal{M}}@\mathsf{t} \models M[\mathsf{t}]$.
2. Condition 2 is that for each $\mathsf{t} \in \boldsymbol{T}$, the edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle \in E$. Well, for every time-value $\mathsf{t} \in \boldsymbol{T}$, $M[\mathsf{t}+1] = blank$, and procedure `init_graph()` adds the edge $\langle s, blank\rangle$ to E for every state-sketch $s \in V$; so, given the preceding condition '$*$', for each time-value $\mathsf{t} \in \boldsymbol{T}$ the edge $\big\langle M[\mathsf{t}],\ M[\mathsf{t}+1]\big\rangle = \big\langle M[\mathsf{t}],\ blank\big\rangle \in E$.

Considering the code of procedure `elaborate_graph()`, this initial map M remains an embedding until just before the first execution of procedure `shift_embedding()` — more precisely, until the operation "E `-=` $\langle s, n\rangle_{dir}$" — since the processing that occurs before then is a calculation of $\widehat{N}$ and some additions to E and V, and those operations do not affect M's status as an embedding.

Procedure `shift_embedding()`

Next, suppose M is an embedding just before execution of procedure `shift_embedding()` — more precisely, immediately before procedure `elaborate_graph()`'s operation "E `-=` $\langle s, n\rangle_{dir}$"; the goal here is to prove that immediately after execution of procedure `shift_embedding()`, the new map M will also be an embedding. The following two claims facilitate this proof:

Claim 1.1.1: At every stage of the processing (after execution of procedure `init_embedding()`), for each time-value $\mathsf{t} \in \boldsymbol{T}$, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models M[\mathsf{t}]$.

Proof: This proof proceeds inductively through the processing.

This claim's condition clearly holds immediately after execution of procedure `init_embedding()` because at that moment M is an embedding (by the preceding) and this claim's condition coincides with condition 1 of the definition of embedding.

Subsequently, this condition can be invalidated only when a value $M[\hat{\mathsf{t}}]$ changes. The only occasion for such change is procedure `shift_embedding()`'s operation:

$$M[\hat{\mathsf{t}}] \texttt{ := selectively\_add\_options*}\big(n,\ \texttt{neighbor\_formulas}(s,\ dir),\ \hat{\mathsf{t}}\big).$$

At such an occasion, according to procedure `shift_embedding()`, there is a time-value $\mathsf{t} \in \boldsymbol{T}$ which is involved and either dir = `future`, $s = M[\mathsf{t}]$, $n = M[\mathsf{t}+1]$, and $\hat{\mathsf{t}} = \mathsf{t}+1$, or else dir = `past`,

$s = M[\mathtt{t}+1]$, $n = M[\mathtt{t}]$, and $\hat{\mathtt{t}} = \mathtt{t}$. Here, in the first case, let time-value $\tilde{\mathtt{t}} = \mathtt{t}$, else let $\tilde{\mathtt{t}} = \mathtt{t}{+}1$; so in either case, $M[\tilde{\mathtt{t}}] = s$.

Now, `neighbor_formulas`(s, dir) uses $\mathit{neighbor\_op}(\mathit{dir})$, which is either "$\bigcirc$" or "$\ominus$". Let:

$$\mathit{neighbor\_constraint\_formulas}(s, \mathit{dir}) = \\ \big\{\text{all the formulas } \big([\mathit{neighbor\_op}(\mathit{dir})]^{\langle\eta+1\rangle}\,\psi\big) \text{ that are in } s\big\}.$$

By inductive hypothesis, $\mathcal{M}@\tilde{\mathtt{t}} \models M[\tilde{\mathtt{t}}] = s$. Then with $\mathit{neighbor\_constraint\_formulas}(s, \mathit{dir}) \subseteq s$, clearly $\mathcal{M}@\tilde{\mathtt{t}} \models \mathit{neighbor\_constraint\_formulas}(s, \mathit{dir})$. If the direction dir is **`future`**, then $\mathit{neighbor\_op}(\mathit{dir})$ is "$\bigcirc$", so each formula in $\mathit{neighbor\_constraint\_formulas}(s, \mathit{dir})$ has the form $\big(\bigcirc^{\langle\eta+1\rangle}\psi\big)$; thus $\mathcal{M}@\tilde{\mathtt{t}} \models \big(\bigcirc^{\langle\eta+1\rangle}\psi\big)$. Then by semantics, $\mathcal{M}@(\tilde{\mathtt{t}}+1) \models \big(\bigcirc^{\eta}\psi\big)$. But in this case with $\mathit{dir} =$ `future`, the time-value $\hat{\mathtt{t}}$ is $\tilde{\mathtt{t}}+1$; so $\mathcal{M}@\hat{\mathtt{t}} \models \big(\bigcirc^{\eta}\psi\big)$. The collection of such relevant formulas $\big(\bigcirc^{\eta}\psi\big)$ happens to be `neighbor_formulas`(s, dir). Then $\mathcal{M}@\hat{\mathtt{t}} \models$ `neighbor_formulas`(s, dir). Similarly, if $\mathit{dir} =$ `past` so $\mathit{neighbor\_op}(\mathit{dir})$ is "$\ominus$" and $\hat{\mathtt{t}} = \tilde{\mathtt{t}} - 1$, again $\mathcal{M}@\hat{\mathtt{t}} \models$ `neighbor_formulas`(s, dir).

By inductive hypothesis, $\mathcal{M}@\hat{\mathtt{t}} \models M[\hat{\mathtt{t}}] = n$. Then, by Claim 1.1.0, the function-call:

`selectively_add_options*`$\big(n,$ `neighbor_formulas`$(s, \mathit{dir}),\ \hat{\mathtt{t}}\big)$

returns a state-sketch $\widehat{n}$ such that $\mathcal{M}@\hat{\mathtt{t}} \models \widehat{n}$. Therefore, the assignment $M[\hat{\mathtt{t}}]$ `:=` $\widehat{n}$ being considered here maintains this claim's condition, as desired.

(Claim 1.1.1) ■

Claim 1.1.2: At every stage of the processing (after execution of procedure `init_embedding()`), for each time-value $\mathtt{t} \in \boldsymbol{T}$, the edge $\big\langle M[\mathtt{t}], M[\mathtt{t}+1]\big\rangle \in E \cup \overline{E}$.

Proof: The proof of this claim proceeds inductively through the processing.

This claim's condition clearly holds immediately after execution of procedure `init_embedding()` because at that moment M is an embedding (by the preceding) and condition 2 of the definition of embedding specifies that each edge $\big\langle M[\mathtt{t}], M[\mathtt{t}+1]\big\rangle \in E$ (for all $\mathtt{t} \in \boldsymbol{T}$).

Otherwise, suppose $E \cup \overline{E}$ contains all the edges that are specified by M and procedure `elaborate_graph()`'s operation "E `-=` $\langle s, n\rangle_{\mathit{dir}}$" occurs. Well, immediately before that, the operation "$\overline{E}$ `+=` $\langle s, n\rangle_{\mathit{dir}}$" occurred. Then clearly $E \cup \overline{E}$ continues to contain all the edges specified by M.

Otherwise, suppose $E \cup \overline{E}$ contains all the edges that are specified by M and procedure `shift_embedding()` changes M, resetting an entry $M[\hat{\mathtt{t}}]$ from a value n to a new value $\widehat{n} =$ `selectively_add_options*`$\big(n,$ `neighbor_formulas`$(s, \mathit{dir}),\ \hat{\mathtt{t}}\big)$. After such a change, does $E \cup \overline{E}$ still contain all the edges that are specified by M? The existence of the one or two new edges connected to $M[\hat{\mathtt{t}}]$'s new value $\widehat{n}$ must be confirmed. (This situation is illustrated above on page 45.) As in the preceding proof (of Claim 1.1.1), there is a time-value $\mathtt{t} \in \boldsymbol{T}$ which is involved and either

$dir = \texttt{future}$, $s = M[\mathtt{t}]$, $n = M[\mathtt{t}+1]$, and $\hat{\mathtt{t}} = \mathtt{t{+}1}$, or else $dir = \texttt{past}$, $s = M[\mathtt{t}+1]$, $n = M[\mathtt{t}]$, and $\hat{\mathtt{t}} = \mathtt{t}$. Here, in the first case, let time-value $\tilde{\mathtt{t}} = \mathtt{t}$, else let $\tilde{\mathtt{t}} = \mathtt{t{+}1}$; so in either case, $M[\tilde{\mathtt{t}}] = s$.

Now, this point in the code ("$M[\hat{\mathtt{t}}]$ `:=` ...") is reached only if $\langle M[\mathtt{t}], M[\mathtt{t}+1]\rangle \notin E$. But by inductive hypothesis, $E \cup \overline{E}$ contained every such edge before this change of $M[\hat{\mathtt{t}}]$. Then $\langle M[\mathtt{t}], M[\mathtt{t}+1]\rangle \in \overline{E}$. Considering the algorithm's code, an edge can enter $\overline{E}$ only if it is inaccurate in some direction and it gets processed in procedure `elaborate_graph()`. Then such is true of the edge $\langle M[\mathtt{t}], M[\mathtt{t}+1]\rangle$. Thus, the edge $\langle s, n\rangle_{dir} = \langle M[\mathtt{t}], M[\mathtt{t}+1]\rangle$ is inaccurate (in some direction), and it has been processed in procedure `elaborate_graph()`. That processing used this same direction dir because no edge can be inaccurate in both directions: all initial edges are accurate in the direction `past`, and subsequent creation of edges uses `neighbor_formulas(`..., $\widetilde{dir}$`)`, which ensures that edges are accurate in the direction $\widetilde{dir}$.[4]

The first edge whose existence must be confirmed here is $\langle M[\tilde{\mathtt{t}}], M[\hat{\mathtt{t}}]\rangle_{dir} = \langle s, \widehat{n}\rangle_{dir}$. But clearly, when the edge $\langle s, \widehat{n}\rangle_{dir}$ was processed in procedure `elaborate_graph()`, the subprocedure-call `establish_edge(`s, $\widehat{n}$, dir`)` established this edge $\langle s, \widehat{n}\rangle_{dir}$ in $E \cup \overline{E}$ (if $\langle s, \widehat{n}\rangle_{dir} \notin E \cup \overline{E}$, the operation E `+=` $\langle s, \widehat{n}\rangle_{dir}$ occurred), so thenceforth — in particular, now — $\langle s, \widehat{n}\rangle_{dir} \in E \cup \overline{E}$, since after creation an edge is only moved from E to $\overline{E}$ or back, never truly expunged. Consequently, the edge $\langle s, \widehat{n}\rangle_{dir}$ remains in $E \cup \overline{E}$, as desired.

If $dir = \texttt{future}$ or $\hat{\mathtt{t}} > 0$, the other edge whose existence must be confirmed here is $\langle M[\hat{\mathtt{t}}],\, M[\ddot{\mathtt{t}}]\rangle_{dir}$, where $\ddot{\mathtt{t}} = \hat{\mathtt{t}} + (\hat{\mathtt{t}} - \tilde{\mathtt{t}})$, i.e. $\ddot{\mathtt{t}}$ is the time-value beyond $\hat{\mathtt{t}}$ in the direction dir; let $\ddot{n} = M[\ddot{\mathtt{t}}]$.

Well, if $n = blank$, then $dir = \texttt{future}$ because the edge $\langle s, blank\rangle$ is not inaccurate in the direction `past`. Further, $\ddot{n}$ must be $blank$.[5] Thus the desired edge $\langle M[\hat{\mathtt{t}}],\, M[\ddot{\mathtt{t}}]\rangle_{dir} = \langle \widehat{n}, \ddot{n}\rangle_{dir}$ is $\langle \widehat{n}, \ddot{n}\rangle_{\texttt{future}}$. Coincidentally, if $n = blank$ then procedure `elaborate_graph()` executed `establish_edge(`$\widehat{n}$, $blank$, `future)`, and thenceforth this edge has remained in $E \cup \overline{E}$.

Otherwise, suppose $n \neq blank$. By inductive hypothesis, the edge $\langle n, \ddot{n}\rangle_{dir} \in E \cup \overline{E}$. Then consider the most recent occasion when the edge $\langle s, n\rangle_{dir}$ was processed. That processing of it included moving it to $\overline{E}$. And it is now in $\overline{E}$. Then it must have remained in $\overline{E}$ continuously since that most recent processing: if not, it could only have been moved from $\overline{E}$ to E and then back, and such movement back — i.e. from E to $\overline{E}$ — can occur only during processing of an edge, in which case there would be more recent processing of $\langle s, n\rangle_{dir}$. Considering all that, the edge $\langle n, \ddot{n}\rangle_{dir}$ cannot have been created after this most recent processing of $\langle s, n\rangle_{dir}$, for in the procedure-call `establish_edge(`n, $\ddot{n}$, dir`)` or

[4] (See the definitions of "accurate" and `neighbor_formulas()` on pages 24 and 30, respectively.)

[5] This statement is justifiable by a claim that at every stage of the processing, the state-sketch $blank$ has no successors other than itself. A proof of this claim proceeds inductively through the processing, considering new edges $\langle s, \hat{n}\rangle_{dir}$, $\langle \hat{n}, blank\rangle$, and $\langle \hat{n}, \ddot{n}\rangle_{dir}$ which arise: $\langle s, n\rangle_{dir}$ is inaccurate so $s \neq blank$ and `neighbor_formulas(`s, dir`)` is not vacuous (or trivial), `neighbor_formulas(`s, dir`)` $\in \hat{n}$ so $\hat{n} \neq blank$, and $\langle n, \ddot{n}\rangle_{dir} \neq \langle n, blank\rangle_{\texttt{past}}$ (by inductive hypothesis) so $\langle \hat{n}, \ddot{n}\rangle_{dir} \neq \langle \hat{n}, blank\rangle_{\texttt{past}}$.

`establish_edge`$\big(\ddot{n},\ n,\ \mathit{reverse}(dir)\big)$ that created $\langle n, \ddot{n}\rangle_{dir}$, the operation `unbar_edges`$(n,\ dir)$ was executed; if this operation occurred after the most recent processing of $\langle s, n\rangle_{dir}$, then $\langle s, n\rangle_{dir} \in \overline{E}$ at that moment and `unbar_edges()` would have moved it from $\overline{E}$ to E, contradicting the preceding. Therefore, at the most recent occasion when the edge $\langle s, n\rangle_{dir}$ was processed, the edge $\langle n, \ddot{n}\rangle_{dir}$ was present in $E \cup \overline{E}$. Then at that occasion, procedure `elaborate_graph()` executed the operation `establish_edge`$(\widehat{n},\ \ddot{n},\ dir)$. Then, the edge $\langle \widehat{n}, \ddot{n}\rangle_{dir}$ is now present in $E \cup \overline{E}$, as desired.

(Claim 1.1.2) ■

One issue here is that meta-procedure `shift_embedding()` specifies an amount of processing which is infinite. Such infiniteness is necessary since an infinite object, model $\boldsymbol{\mathcal{M}}$, is involved. Nonetheless, each state-sketch $M[\mathrm{t}]$ (for $\mathtt{t} \in \boldsymbol{T}$) is well-defined — not undergoing reassignment infinitely many times. Consider a reassignment:

$$\begin{aligned} M[\hat{\mathrm{t}}] \ &:= \mathtt{selectively\_add\_options*}(n, \ldots) \\ &= \mathtt{selectively\_add\_options*}\big(M[\hat{\mathrm{t}}], \ldots\big), \end{aligned}$$

where the "$M[\hat{\mathrm{t}}]$" appearing as an argument is the old value of $M[\hat{\mathrm{t}}]$. The function-call here yields a combination of its first argument with further formulas, so the old value of $M[\hat{\mathrm{t}}]$ is contained in the result. Thus the progression of state-sketches which are values of $M[\hat{\mathrm{t}}]$ during processing is monotonic, each containing the preceding. Then this progression is finitely bounded — and therefore ultimately stable — since state-sketches are finitely bounded according to Lemma 0.1.

The outstanding task is to prove that (immediately) after each execution of procedure `shift_embedding()`, M is indeed an embedding.

$*$. A fundamental condition is that M must be a map from $\boldsymbol{T}$ to V. Naturally, this condition might be violated only when V loses elements — but such loss never occurs — or when values of $M[]$ change. Such changes of $M[]$ occur at Line 7 of procedure `shift_embedding()`:

$$M[\hat{\mathrm{t}}] := \mathtt{selectively\_add\_options*}\big(n,\ \mathtt{neighbor\_formulas}(s,\ dir),\ \hat{\mathtt{t}}\big).$$

Considering what precedes this operation, the edge $\langle s, n\rangle_{dir} = \big\langle M[\mathrm{t}], M[\mathrm{t}+1]\big\rangle \notin E$ (where $\mathtt{t} \in \boldsymbol{T}$). Then by Claim 1.1.2 — and further consulting its proof — the edge $\langle s, n\rangle_{dir}$ was previously processed in procedure `elaborate_graph()`, being inaccurate in the direction dir. Now, according to the proof of Claim 1.1.1, Claim 1.1.0 applies here: $M[\hat{\mathrm{t}}]$'s new value $\widehat{n} \in$ `add_options*`$\big(n,\ \mathtt{neighbor\_formulas}(s,\ dir)\big)$. Coincidentally, when the edge $\langle s, n\rangle_{dir}$ was processed in procedure `elaborate_graph()`, the operations $\widehat{N}$:= `add_options*`$\big(n,\ \mathtt{neighbor\_formulas}(s,\ dir)\big)$ and then $V := V \cup \widehat{N}$ occurred. Consequently, $M[\hat{\mathrm{t}}]$'s new value $\widehat{n} \in V$. Thus, inclusion of values of $M[]$ in V is maintained, as desired.

0. At any occasion when the entry $M[0]$ changes, its reassignment is as follows:
$$M[0] = M[\hat{t}] \;\texttt{:=}\; \texttt{selectively\_add\_options*}\big(M[\hat{t}], \ldots\big)$$
$$= \texttt{selectively\_add\_options*}\big(M[0], \ldots\big),$$
where the "$M[0]$" appearing as an argument is the old value of $M[0]$. Since all formulas of $M[0]$ are thus (considering how function `selectively_add_options*()` proceeds) kept in successive values of $M[0]$, then in particular the formula *first* — which is present before this reassignment, inductively — is kept. Thus the condition that $\mathit{first} \in M[0]$ is preserved, as desired.
1. The condition that for each time-value $\mathsf{t} \in \boldsymbol{T}$, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models M[\mathsf{t}]$ is Claim 1.1.1.
2. The condition that for each time-value $\mathsf{t} \in \boldsymbol{T}$, the edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle \in E$ is obvious after execution of procedure `shift_embedding()` since such execution terminates only when there no longer exists any time-value $\mathsf{t} \in \boldsymbol{T}$ such that the edge $\big\langle M[\mathsf{t}], M[\mathsf{t}+1]\big\rangle \notin E$.

The preceding shows that at every stage of the processing (after initialization), there's an embedding of model $\boldsymbol{\mathcal{M}}$ in the graph, as desired for Lemma 1.1.

(Lemma 1.1) ■

Lemma 1.2: After procedure `elaborate_graph()` finishes, along the graph's path specified by the embedding that is constructed as in Lemma 1.1, all fulfillment-constraints are explicitly satisfied.

Proof: Let V_{final}, E_{final}, and M_{final} be the vertices, edges (excluding $\overline{E}$), and embedding — respectively — after the construction finishes, let $\mathsf{t} \in \boldsymbol{T}$ be arbitrary, and let φ be any formula having the form $\Diamond\psi$, $(\xi \,\mathsf{U}\, \psi)$, $(\Diamond_{\cong\eta}\, \psi)$, or $(\xi \,\mathsf{U}_{\cong\eta}\, \psi)$ such that $\varphi \in M_{\text{final}}[\mathsf{t}]$. The goal here is to show that φ's fulfillment-constraint is satisfied — along the path specified by M. Well, since M_{final} is an embedding, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \varphi$. Then since $\boldsymbol{\mathcal{M}}$ is a model, there must be some least time-delay $\mathsf{d}_{\text{fulfill}} \in \boldsymbol{T}$ such that $\boldsymbol{\mathcal{M}}@(\mathsf{t}+\mathsf{d}_{\text{fulfill}}) \models \psi$ — and, for the cases with "$\cong_\eta$", $\mathsf{d}_{\text{fulfill}} \cong \eta$. Since $\mathsf{d}_{\text{fulfill}}$ is minimal, $\boldsymbol{\mathcal{M}}@(\mathsf{t}+\mathsf{d}) \not\models \psi$ for each $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{d} < \mathsf{d}_{\text{fulfill}}$ (and, if appropriate, $\mathsf{d} \cong \eta$); so since M_{final} is an embedding, $\psi \notin M_{\text{final}}[\mathsf{t}+\mathsf{d}]$ for each such d.

Claim 1.2.1: For each time-delay $\mathsf{d} \in \boldsymbol{T}$ such that $\mathsf{d} \leq \mathsf{d}_{\text{fulfill}}$ — and, if φ is $(\Diamond_{\cong\eta}\, \psi)$ or $(\xi \,\mathsf{U}_{\cong\eta}\, \psi)$, $\mathsf{d} \cong \eta$ — $\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}]$.

Proof: This proof proceeds by induction on natural numbers ($\boldsymbol{T}$).

For the base-case when $\mathsf{d} = 0$, the condition that $\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}] = M_{\text{final}}[\mathsf{t}]$ is given.

Otherwise, let arbitrary time-delay $\mathsf{d} \in \boldsymbol{T}$ be such that $\mathsf{d} < \mathsf{d}_{\text{fulfill}}$, and suppose by inductive hypothesis that $\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}]$ — and, if φ is $(\Diamond_{\cong\eta}\, \psi)$ or $(\xi \,\mathsf{U}_{\cong\eta}\, \psi)$, $\mathsf{d} \cong \eta$; it's necessary to show that $\varphi \in M_{\text{final}}[\mathsf{t}+(\mathsf{d}+1)]$ — or, if φ is $(\Diamond_{\cong\eta}\, \psi)$ or $(\xi \,\mathsf{U}_{\cong\eta}\, \psi)$, that $\varphi \in M_{\text{final}}[\mathsf{t}+(\mathsf{d}+\eta)]$. By the paragraph preceding this claim, the supposition that $\mathsf{d} < \mathsf{d}_{\text{fulfill}}$ implies that $\psi \notin M_{\text{final}}[\mathsf{t}+\mathsf{d}]$. With

ψ excluded from $M_{\text{final}}[\mathsf{t}+\mathsf{d}]$, the options for additional formulas required to satisfy the immediate constraints of formula $\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}]$ diminish to just one option — for each case of φ: $\{\bigcirc\Diamond\psi\}$ if φ is $\Diamond\psi$, $\{\xi, \bigcirc(\xi \mathsf{U} \psi)\}$ if φ is $(\xi \mathsf{U} \psi)$, $\{\bigcirc^{\eta}(\Diamond_{\cong\eta} \psi)\}$ if φ is $(\Diamond_{\cong\eta} \psi)$, or $\{(\Box_{<\eta} \xi), (\bigcirc^{\eta}(\xi \mathsf{U}_{\cong\eta} \psi))\}$ if φ is $(\xi \mathsf{U}_{\cong\eta} \psi)$. These options may be expressed as $\{\bigcirc\varphi\}$, $\{\ldots, \bigcirc\varphi\}$, $\{\bigcirc^{\eta}\varphi\}$, or $\{\ldots, (\bigcirc^{\eta}\varphi)\}$, respectively. The construction of state-sketches ensures that the state-sketch $M_{\text{final}}[\mathsf{t}+\mathsf{d}]$ contains the appropriate option for φ.[6] Thus, $M_{\text{final}}[\mathsf{t}+\mathsf{d}]$ contains $\bigcirc\varphi$ or $(\bigcirc^{\eta}\varphi)$, depending on φ. With η (if relevant) nonzero, this situation may be expressed in either case as $(\bigcirc^{\langle\tilde{\eta}+1\rangle}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}]$, where the numeral $\tilde{\eta}$ is either 0 or $\eta - 1$, depending on φ.

Since M_{final} is an embedding, the edge $\langle M_{\text{final}}[\mathsf{t}+\mathsf{d}], M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]\rangle \in E_{\text{final}}$. Since procedure `elaborate_graph()` has finished, this edge is accurate in the direction `future`. Then with $(\bigcirc^{\langle\tilde{\eta}+1\rangle}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}]$, $(\bigcirc^{\tilde{\eta}}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]$.

If φ is $\Diamond\psi$ or $(\xi \mathsf{U} \psi)$, then the numeral $\tilde{\eta}$ is 0, so the statement "$(\bigcirc^{\tilde{\eta}}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]$" reduces to "$\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]$", which is as desired in these cases for φ.

Otherwise, if φ is $(\Diamond_{\cong\eta} \psi)$ or $(\xi \mathsf{U}_{\cong\eta} \psi)$, in which case the numeral $\tilde{\eta} = \eta - 1$, then "$(\bigcirc^{\tilde{\eta}}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]$" may be expressed as "$(\bigcirc^{\langle\eta-1\rangle}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+1]$". (The meaning of "$\langle\eta-1\rangle$" in the formula here should be clear.) If $\eta > 1$, then again since `elaborate_graph()` has finished, $(\bigcirc^{\langle\eta-2\rangle}\varphi) \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+2]$. Continuing, clearly $(\bigcirc^{\langle\eta-\eta\rangle}\varphi) = \varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}+\eta]$, which is as desired in these cases for φ.

(Claim 1.2.1) ■

By Claim 1.2.1, $\varphi \in M_{\text{final}}[\mathsf{t}+\mathsf{d}_{\text{fulfill}}]$.

Then consider the moment during processing when the entry $M[\mathsf{t}+\mathsf{d}_{\text{fulfill}}]$ was first set to a value containing φ. Considering the pseudo-code for the construction, this moment is either at procedure `init_embedding()`'s line:

$$M[0] \mathtt{\ :=\ selectively\_add\_options*}\big(\{\mathit{first}\}, \{\varphi_{\text{checking}}\}, 0\big)$$

or at procedure `shift_embedding()`'s line:

$$M[\hat{\mathsf{t}}] \mathtt{\ :=\ selectively\_add\_options*}\big(n, \mathtt{neighbor\_formulas}(s, \mathit{dir}), \hat{\mathsf{t}}\big);$$

in the latter case, the "n" appearing as an argument is the old value of $M[\hat{\mathsf{t}}]$. In the first case, obviously $\varphi \notin \{\mathit{first}\}$; in the other case, by the choice of this moment during the construction, the old value of $M[\hat{\mathsf{t}}]$ does not contain φ; so, these two cases reduce to:

$$M[\mathsf{t}+\mathsf{d}_{\text{fulfill}}] \mathtt{\ :=\ selectively\_add\_options*}(b, a, \mathsf{t}+\mathsf{d}_{\text{fulfill}}),$$

[6] This statement can be justified either by Lemma 1.3 (which is on the page after the next one) or, actually, directly via an argument about a chain of calls of `selectively_add_options*()` (i.e. an argument as is used later in this proof), involving considering how φ can be present while ψ is not.

where b and a are sets of formulas and $\varphi \notin b$. Then how can φ appear in the result of this operation? The result is constructed via a chain of recursive calls, say numbered $1..j$, as follows (abbreviating "`selectively_add_options*`" slightly):

$$\begin{array}{ll}
\texttt{selectively_add_opts*}(\mathit{base}_1 := b,\ \mathit{additions}_1 := a,\ \hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}) & \text{call \#1} \\
\quad\texttt{selectively_add_opts*}\big(\mathit{base}_2 := \mathit{selected_f}_1, & \\
\qquad \mathit{additions}_2 := \mathit{selected_f}_1 - (\mathit{base}_1 \cup \mathit{additions}_1), & \text{call \#2} \\
\qquad \hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}\big) & \\
\qquad\texttt{selectively_add_opts*}\big(\mathit{base}_3 := \mathit{selected_f}_2, & \\
\qquad\quad \mathit{additions}_3 := \mathit{selected_f}_2 - (\mathit{base}_2 \cup \mathit{additions}_2), & \text{call \#3} \\
\qquad\quad \hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}\big) & \\
\qquad\quad \ddots & \\
\qquad\qquad\texttt{selectively_add_opts*}\big(\mathit{base}_j := \mathit{selected_f}_{j-1}, & \\
\qquad\qquad\quad \mathit{additions}_j := \mathit{selected_f}_{j-1} - (\mathit{base}_{j-1} \cup \mathit{additions}_{j-1}), & \text{call \#}j \\
\qquad\qquad\quad \hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}\big) &
\end{array}$$

The new value of $M[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}]$ can be the result of this chain only if $\mathit{additions}_j = \{\}$ and $M[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}] = \mathit{base}_j$. Then $\varphi \in \mathit{base}_j = M[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}]$.

For each $i \in \{1..(j-1)\}$, $\mathit{selected\_f}_i := \mathit{base}_i \cup \mathit{additions}_i \cup (\text{selected options})$ and $\mathit{base}_{i+1} := \mathit{selected\_f}_i$; so formulas accumulate in the *base*s here, i.e. each base_{i+1} contains all the formulas of every 'ancestral' call's *base* — and *additions*. Considering that $\varphi \in M[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}] = \mathit{base}_j$, let i_{b} be the least $i \in \{1..j\}$ such that $\varphi \in \mathit{base}_i$. Since $\varphi \notin b = \mathit{base}_1$, $i_{\mathrm{b}} > 1$. Then:

$$\mathit{additions}_{i_{\mathrm{b}}} := \mathit{selected\_f}_{i_{\mathrm{b}}-1} - (\mathit{base}_{i_{\mathrm{b}}-1} \cup \mathit{additions}_{i_{\mathrm{b}}-1}) = \mathit{base}_{i_{\mathrm{b}}} - (\mathit{base}_{i_{\mathrm{b}}-1} \cup \mathit{additions}_{i_{\mathrm{b}}-1})\,;$$

hence $\mathit{base}_{i_{\mathrm{b}}} - \mathit{base}_{i_{\mathrm{b}}-1} = \mathit{additions}_{i_{\mathrm{b}}} \cup \mathit{additions}_{i_{\mathrm{b}}-1}$. By choice of i_{b}, $\varphi \in \mathit{base}_{i_{\mathrm{b}}}$ and $\varphi \notin \mathit{base}_{i_{\mathrm{b}}-1}$, i.e. $\varphi \in \mathit{base}_{i_{\mathrm{b}}} - \mathit{base}_{i_{\mathrm{b}}-1}$. Then $\varphi \in \mathit{additions}_{i_{\mathrm{b}}} \cup \mathit{additions}_{i_{\mathrm{b}}-1}$; let i_{a} be i_{b} or $i_{\mathrm{b}}-1$ so that $\varphi \in \mathit{additions}_{i_{\mathrm{a}}}$.

Then during call $\#i_{\mathrm{a}}$, $\mathit{selected\_f}_{i_{\mathrm{a}}} := \big(\mathit{base}_{i_{\mathrm{a}}} \cup \mathit{additions}_{i_{\mathrm{a}}} \cup (\text{selected options})\big)$, where with $\varphi \in \mathit{additions}_{i_{\mathrm{a}}}$, one of these selected options is `select_option(`φ, $\hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}$`)`. By supposition, $\boldsymbol{\mathcal{M}}@(\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}) \models \psi$, i.e. $\boldsymbol{\mathcal{M}}@\hat{\mathsf{t}} \models \psi$. Then considering the pseudo-code of function `select_option()`, the result of `select_option(`φ, $\hat{\mathsf{t}} := \mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}$`)` is $\{\psi\}$. Thus, $\psi \in \mathit{selected\_f}_{i_{\mathrm{a}}}$.

Consequently, $\psi \in \mathit{base}_{i_{\mathrm{a}}+1} := \mathit{selected\_f}_{i_{\mathrm{a}}}$; and further, as formulas accumulate in $\mathit{base}_{i_{\mathrm{a}}+1}$, $\mathit{base}_{i_{\mathrm{a}}+2}, \ldots, \mathit{base}_j$, $\psi \in \mathit{base}_j$. Then $\psi \in M[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}] = \mathit{base}_j$.

Then, as formulas are retained in successive values of $M[]$, $\psi \in M_{\mathrm{final}}[\mathsf{t} + \mathsf{d}_{\mathrm{fulfill}}]$. Thus, for this given formula $\varphi \in$ state-sketch $M_{\mathrm{final}}[\mathsf{t}]$, its fulfillment-constraint is satisfied along the path comprising values of $M[]$, as desired.

(Lemma 1.2) ■

To conclude the proof of Theorem 1, it's necessary to show that procedure `satisfy()`'s sub-procedure `search_graph()` does indeed report satisfiability. As procedure `search_graph()` needs to find certain formulas present in state-sketches, the following lemma is useful here:

Lemma 1.3: At every stage of the algorithm's processing, all state-sketches' immediate constraints are satisfied.

Proof: This proof proceeds inductively through the processing.

One initial state-sketch is $\mathit{blank} = \{\neg \mathit{first}\}$. The immediate constraints are that for each formula $\varphi \in \mathit{blank}$, φ must not be false, blank must not contain $\neg\varphi$, and blank must contain certain formulas specified in Table I.1 (there may be options). Well, clearly these constraints are satisfied.

Every other state-sketch s results from a function-call `add_options*(`b, a`)`, where b and a are sets of formulas. In one case, b is $\{\mathit{first}\}$; in every other case, b is an already extant state-sketch n. Considering the set of formulas $\{\mathit{first}\}$ as a state-sketch, clearly its immediate constraints are satisfied. In the other cases, already extant state-sketch n should have its immediate constraints satisfied by inductive hypothesis. Thus, every state-sketch $s \in$ `add_options*(`b, a`)` (for some b and a), where the immediate constraints for b are satisfied.

Consider any state-sketch $s \neq \mathit{blank}$ and any formula $\varphi \in s$; are φ's immediate constraints satisfied? Well, how can φ appear in this result s of a function-call `add_options*(`b, a`)`? But then, how is s obtained as a result? This result s is constructed via a chain of `add_options*()`'s recursive calls, say numbered $1..j$, as follows:

$$
\begin{array}{ll}
\texttt{add\_options*(}\mathit{base}_1 \texttt{ := } b,\ \mathit{additions}_1 \texttt{ := } a\texttt{)} & \text{call \#1} \\
\quad \texttt{add\_options*}\big(\mathit{base}_2 \texttt{ := } f_1,\ \mathit{additions}_2 \texttt{ := } f_1 - (\mathit{base}_1 \cup \mathit{additions}_1)\big) & \text{call \#2} \\
\quad\quad \texttt{add\_options*}\big(\mathit{base}_3 \texttt{ := } f_2,\ \mathit{additions}_3 \texttt{ := } f_2 - (\mathit{base}_2 \cup \mathit{additions}_2)\big) & \text{call \#3} \\
\quad\quad\quad \ddots & \\
\quad\quad\quad\quad \texttt{add\_options*}\big(\mathit{base}_j \texttt{ := } f_{j-1},\ \mathit{additions}_j \texttt{ := } f_{j-1} - (\mathit{base}_{j-1} \cup \mathit{additions}_{j-1})\big) & \text{call \#}j
\end{array}
$$

State-sketch s is the result of this particular chain of calls only if $\mathit{additions}_j = \{\}$, call $\#j$ returns $\{\mathit{base}_j\}$, and $s = \mathit{base}_j$. Then $\varphi \in \mathit{base}_j = s$.

If $j = 1$, then $s = \mathit{base}_j = \mathit{base}_1 = b$. Then, with b satisfying the immediate constraints for φ, so does $s = b$.

Otherwise, there is a call $\#(j-1)$ which provides arguments for call $\#j$: $\mathit{base}_j \texttt{ := } f_{j-1}$. In call $\#(j-1)$, this f_{j-1} would have been rejected from $\mathit{intermediates}_{j-1}$ if it contained any obviously contradictory pair of formulas $\{\psi, \neg\psi\}$. Therefore, $s = \mathit{base}_j = f_{j-1}$ does not contain $\neg\varphi$.

For each $i \in \{1..(j-1)\}$, $f_i \texttt{ := } \mathit{base}_i \cup \mathit{additions}_i \cup \text{(options)}$ and $\mathit{base}_{i+1} \texttt{ := } f_i$; so formulas accumulate in the *bases* here, i.e. each base_{i+1} contains all the formulas of every 'ancestral' call's *base* — and *additions*. Considering that $\varphi \in s = \mathit{base}_j$, let i_b be the least $i \in \{1..j\}$ such that $\varphi \in \mathit{base}_i$. If $i_\mathrm{b} = 1$, then $\varphi \in \mathit{base}_1 = b$. In this case, with the immediate constraints for b satisfied by the above, $\varphi \neq \mathit{false}$ and b contains appropriate additional formulas from Table I.1; then

s contains these additional formulas also, since $b = base_1 \subset base_j = s$. Otherwise, if $i_{\rm b} > 1$, then:

$$additions_{i_{\rm b}} := f_{i_{\rm b}-1} - (base_{i_{\rm b}-1} \cup additions_{i_{\rm b}-1}) = base_{i_{\rm b}} - (base_{i_{\rm b}-1} \cup additions_{i_{\rm b}-1});$$

hence $base_{i_{\rm b}} - base_{i_{\rm b}-1} = additions_{i_{\rm b}} \cup additions_{i_{\rm b}-1}$. By choice of $i_{\rm b}$ (and with $i_{\rm b} > 1$), $\varphi \in base_{i_{\rm b}}$ and $\varphi \notin base_{i_{\rm b}-1}$, i.e. $\varphi \in base_{i_{\rm b}} - base_{i_{\rm b}-1}$. Then $\varphi \in additions_{i_{\rm b}} \cup additions_{i_{\rm b}-1}$; let $i_{\rm a}$ be $i_{\rm b}$ or $i_{\rm b}-1$ so that $\varphi \in additions_{i_{\rm a}}$.

Then during call $\#i_{\rm a}$, processing checks $false \in additions_{i_{\rm a}}$, and if this condition is true, there would have been no result here; so $false \notin additions_{i_{\rm a}}$, which implies that $\varphi \neq false$. Also during call $\#i_{\rm a}$, $f_{i_{\rm a}} := (base_{i_{\rm a}} \cup additions_{i_{\rm a}} \cup (\text{some options}))$, where with $\varphi \in additions_{i_{\rm a}}$, one of these options is $o \in$ `options(`φ`)`. Thus $f_{i_{\rm a}}$ contains one of the options for additional formulas required to satisfy φ's immediate constraints. Then, as each $base_i$ contains all the formulas of every 'ancestral' call's *base* and *additions* here, $base_j = s$ also contains the desired formulas.

Thus, all immediate constraints are satisfied.

(Lemma 1.3) ■

Now, procedure `search_graph()` can be addressed.

By Theorem 0, the final graph $(V_{\rm final}, E_{\rm final})$ is finite. However, $\boldsymbol{T}$, the domain for time, is infinite. Then with the embedding $M_{\rm final}$ (of Lemma 1.1) being a map from $\boldsymbol{T}$ to $V_{\rm final}$, some values of $M_{\rm final}[]$ must be repeated infinitely. Since $M_{\rm final}$ specifies a path through the graph, there is a path from each of these infinitely repeated vertices to each of the others. Thus these infinitely repeated vertices form a strongly connected component $scc_{M_{\rm f}}$ of the graph. Then consider `search_graph()`'s processing:

Line 1: "for each subset $mscc \subseteq V_{\rm final}$ comprising a maximal strongly connected component"

The presentation of procedure `search_graph()` specifies that such search for an *mscc* uses paths starting at state-sketches that contain *first*. The existence of at least one such state-sketch is guaranteed by condition 0 of the definition of embedding, that $first \in M_{\rm final}[0]$ (plus condition '$*$', that $M_{\rm final}[0] \in V$). Then the search here can find the graph's maximal strongly connected component that contains $scc_{M_{\rm f}}$.

Line 2: "let the set of formulas $f := \bigcup_{s \in mscc} s$"

This operation gathers into set f all the formulas that appear in *mscc*.

Lines 3–5: "for each formula having the form $\Diamond\psi$ or $(\xi \,\mathsf{U}\, \psi) \in f$: if this $\psi \notin f$, stop ..."

Let φ be any formula in f having the form $\Diamond\psi$ or $(\xi \,\mathsf{U}\, \psi)$. Considering how f is derived from *mscc*, *mscc* must contain a state-sketch s_1 such that $\varphi \in s_1$. Then since *mscc* is strongly connected, it contains a path from s_1 to any element of *mscc*'s subset $scc_{M_{\rm f}}$; let $(s_1, s_2, s_3, \ldots, s_k)$ be such a path, with $s_k \in scc_{M_{\rm f}}$. If any s_i here contains ψ, then with $s_i \in mscc$, $\psi \in f$, so `search_graph()` would not stop processing *mscc* at Line 5. Otherwise, with $\varphi \in s_1$ and $\psi \notin s_1$, immediate constraints for

φ (which is either $\Diamond\psi$ or $(\xi \mathsf{U} \psi)$) entail that $\bigcirc\varphi \in s_1$, by Lemma 1.3. Then with the edge $\langle s_1, s_2\rangle$ clearly accurate at this point, $\varphi \in s_2$. Similarly, with $\psi \notin s_2$, $\bigcirc\varphi \in s_2$ and $\varphi \in s_3$. Continuing, $\varphi \in s_k$.

Now, $s_k \in scc_{M_\mathrm{f}}$, which comprises the values of $M_\mathrm{final}[]$ that are repeated infinitely. Let the time-value $\mathsf{t} \in \boldsymbol{T}$ be such that (1) $M_\mathrm{final}[\mathsf{t}] = s_k$ and (2) for all $\mathsf{t}' \geq \mathsf{t}$, $M_\mathrm{final}[\mathsf{t}'] \in scc_{M_\mathrm{f}}$. As in the proof of Lemma 1.2 above, some $M_\mathrm{final}[\mathsf{t} + \mathsf{d}_\mathsf{fulfill}]$ contains ψ. Since $\mathsf{t} + \mathsf{d}_\mathsf{fulfill} \geq \mathsf{t}$, $M_\mathrm{final}[\mathsf{t} + \mathsf{d}_\mathsf{fulfill}] \in scc_{M_\mathrm{f}}$. But $scc_{M_\mathrm{f}} \subseteq mscc$, so $M_\mathrm{final}[\mathsf{t} + \mathsf{d}_\mathsf{fulfill}] \in mscc$. Thus a state-sketch $s = M_\mathrm{final}[\mathsf{t} + \mathsf{d}_\mathsf{fulfill}]$ contains ψ and $s \in mscc$. Then $\psi \in f$. Then, again, procedure `search_graph()` does not stop processing *mscc* at Line 5.

Lines 6–8: "if ... $(\Diamond_{\cong\eta}\, \psi)$ or $(\xi\ \mathsf{U}_{\cong\eta}\ \psi) \in f$: if `fulfilling_congs(`*mscc*`)` $=$ `false`, stop ..."

Again, even if such a formula φ isn't fulfilled in $mscc - scc_{M_\mathrm{f}}$, it is fulfilled (appropriately) in scc_{M_f}. Such fulfillment implies that the function `fulfilling_congruences()` would return `true`: Using the notation specified in the presentation of this function (starting on page 34), consider a state-sketch $s_\mathrm{n} \in mscc$ ("n" for "needing") such that $\varphi_i \in s_\mathrm{n}$ and a path (in *mscc*) from $s_{\bar{0}}$ to s_n is P_n; then the value $|P_\mathrm{n}|\ MOD\ \eta_i = m$ is included in the set $needing[i]$. Now, the set *mscc* contains a path from s_n to scc_{M_f}. If $\varphi_i \in s_n$ isn't straightforwardly fulfilled along this path, then a state-sketch $\tilde{s}_n \in scc_{M_\mathrm{f}}$ contains φ_i (with $\tilde{s}_n$ at a distance $\cong \eta_i$ from s_n along the path). As in the proof of Lemma 1.2 above, $\tilde{s}_n$ is $M_\mathrm{final}[\mathsf{t}]$ for some t, and some $M_\mathrm{final}[\mathsf{t} + \mathsf{d}_\mathsf{fulfill}]$ contains ψ_i (with $\mathsf{d}_\mathsf{fulfill} \cong \eta_i$). Thus there is a state-sketch $s_\mathrm{f} \in mscc$ ("f" for "fulfilling") such that $\psi_i \in s_\mathrm{f}$, a path from s_n to s_f (in *mscc*) is P_nf, and $|P_\mathrm{nf}| \cong \eta_i$ — i.e. $|P_\mathrm{nf}|\ MOD\ \eta_i = 0$. Then conjoining the paths P_n and P_nf yields a path P_f from $s_{\bar{0}}$ to s_f, and modulo η_i, $|P_\mathrm{f}| = |P_\mathrm{n}| + |P_\mathrm{nf}| = m + 0 = m\ \ (MOD\ \eta_i)$. Thus there is a path from $s_{\bar{0}}$ to s_f whose length modulo η_i is m, so the value m is included in the set $fulfilling[i]$. Since the specifics here were arbitrary, $needing[i] \subseteq fulfilling[i]$ for each $i \in \{1..k\}$, so `fulfilling_congruences()` returns `true`.

Then processing of *mscc* does not stop at this point in procedure `search_graph()`.

Line 9: "(* ... *)"

This line is a comment.

Line 10: "report that the originally given formula $\varphi_\mathrm{checking}$ is satisfiable"

This is what was desired.

(Theorem 1: Success) ■

I.2.2 Sufficiency

Theorem 2 (Sufficiency): If procedure `satisfy()` reports that its given (input-) formula $\varphi_{\text{checking}}$ is satisfiable, then this formula really is satisfiable.

Proof:

Clearly, procedure `satisfy()` reports satisfiability only after constructing a graph etc. It would be complacent to say that a model of $\varphi_{\text{checking}}$ is then simply given by the state-sketches of the relevant *mscc* etc. Unfortunately, it's not obvious that these sets of formulas are sufficiently compatible for any model to satisfy all of them together. Hence the necessity of the work here.

The guiding principle is that the output of procedure `delineate_model()` actually is sufficient to specify a model that satisfies $\varphi_{\text{checking}}$. Now, basically, `delineate_model()` determines an infinite path of state-sketches $s_0, s_1, s_2, s_3, \ldots$ and reports their literals, say as sets denoted by $l_0, l_1, l_2, l_3, \ldots$, where each l_{t} (for $\mathsf{t} \in \boldsymbol{T}$) comprises all the literals of s_{t}. Then for each $\mathsf{t} \in \boldsymbol{T}$, a state $\boldsymbol{\mathcal{M}}$@t of a model $\boldsymbol{\mathcal{M}}$ can be specified by having $\boldsymbol{\mathcal{M}}$@t satisfy a proposition π if $\pi \in l_{\mathsf{t}}$, falsify π if $\neg\pi \in l_{\mathsf{t}}$, and otherwise — if the set l_{t} contains neither π nor $\neg\pi$ — allowing $\boldsymbol{\mathcal{M}}$@t to arbitrarily satisfy or falsify π. This specification cannot be contradictory, requiring $\boldsymbol{\mathcal{M}}$@t to both satisfy and falsify some proposition π, because Lemma 1.3 implies that state-sketch s_{t} does not contain both formulas π and $\neg\pi$, so with $l_{\mathsf{t}} \subseteq s_{\mathsf{t}}$, the set l_{t} also does not contain both π and $\neg\pi$.

Thus, a model $\boldsymbol{\mathcal{M}}$ is delineated. The desired condition $\boldsymbol{\mathcal{M}} \models \varphi_{\text{checking}}$ is derivable from the following lemma:

Lemma: For each formula φ: for each time-value $\mathsf{t} \in \boldsymbol{T}$: if $\varphi \in s_{\mathsf{t}}$, then $\boldsymbol{\mathcal{M}}$@t $\models \varphi$.

Proof: The proof of this lemma proceeds via induction on formulas, using a well-founded ordering "$\prec$" (for formulas) which is defined as follows:

$$\text{“}\upsilon \prec \varphi\text{”} \quad \text{means} \quad \textit{associated\_formulas}(\upsilon) \subsetneqq \textit{associated\_formulas}(\varphi)\,.$$

This ordering is well-founded because proper set-inclusion for finite sets is well-founded and the proof of Lemma 0.1 shows that *associated_formulas*$(\ldots)$ is a finite set.

The base-case for a formula φ in this inductive proof is when there is no formula $\upsilon \prec \varphi$. Then φ must be either *first* or $\neg$*first*. The following claim facilitates treatment of these cases for φ:

Claim: Continually, throughout procedure `satisfy()`'s construction of the graph (V, E), every state-sketch contains either *first* or $\neg$*first*; and a state-sketch contains *first* only if the state-sketch has no predecessors.

Proof: This proof proceeds inductively through the processing.

The initial state-sketches comprise *blank* and the results of procedure `init_graph()`'s subfunction-call `add_options*(`$\mathit{base} := \{\mathit{first}\}, \ldots$`)`. Obviously, $\neg\mathit{first} \in \{\neg\mathit{first}\} = \mathit{blank}$. And `add_options*()` accumulates *base*-formulas and returns them, so every initial state-sketch other than *blank* contains *first*. Thus, the condition re containment of *first* or $\neg$*first* holds initially.

Next, each initial edge is $\langle s, \mathit{blank}\rangle$ for some state-sketch $s \in V$. Obviously, $\mathit{first} \notin \mathit{blank} = \{\neg\mathit{first}\}$. No other state-sketch has predecessors here. Thus, the condition that a state-sketch s contains *first* only if it has no predecessors also holds initially.

Subsequently, creation of a state-sketch $\widehat{n}$ occurs via procedure `elaborate_graph()`'s subfunction-call `add_options*(`$\mathit{base} := n, \ldots$`)`, where n is a previously existing state-sketch. Then the result $\widehat{n}$ must contain n, so as n contains either *first* or $\neg$*first*, so does $\widehat{n}$.

Then, any edge that is created has the form $\langle s, \widehat{n}\rangle_{\mathit{dir}}$, $\langle \widehat{n}, \mathit{blank}\rangle_{\mathtt{future}}$, or $\langle \widehat{n}, \ddot{n}\rangle_{\mathit{dir}}$. (The set of edges E gains elements also when procedure `unbar_edges()` moves edges from $\overline{E}$ to E, but these are old edges, not newly created ones, so they are presumably compliant.) As with $\langle s, \mathit{blank}\rangle$ above, an edge $\langle \widehat{n}, \mathit{blank}\rangle_{\mathtt{future}}$ easily complies with this claim's condition since $\mathit{first} \notin \mathit{blank}$. The other cases for an edge, $\langle s, \widehat{n}\rangle_{\mathit{dir}}$ and $\langle \widehat{n}, \ddot{n}\rangle_{\mathit{dir}}$, expand to $\langle s, \widehat{n}\rangle$, $\langle \widehat{n}, s\rangle$, $\langle \widehat{n}, \ddot{n}\rangle$, or $\langle \ddot{n}, \widehat{n}\rangle$, depending on the direction *dir*. In each of these cases, there is a previously existing similar edge: $\langle s, n\rangle$, $\langle n, s\rangle$, $\langle n, \ddot{n}\rangle$, or $\langle \ddot{n}, n\rangle$, respectively. If $\langle n, s\rangle$ or $\langle n, \ddot{n}\rangle$ exists previously, then by inductive hypothesis $\mathit{first} \notin s$ or $\mathit{first} \notin \ddot{n}$, as desired for the new edge $\langle \widehat{n}, s\rangle$ or $\langle \widehat{n}, \ddot{n}\rangle$, respectively. If $\langle s, n\rangle$ or $\langle \ddot{n}, n\rangle$ exists previously, then by inductive hypothesis $\mathit{first} \notin n$. Then further by inductive hypothesis, $\neg\mathit{first} \in n$. Then $\neg\mathit{first} \in \widehat{n}$. Then by Lemma 1.3 (which is on page 57), $\neg(\neg\mathit{first}) = \mathit{first} \notin \widehat{n}$, as desired for an edge $\langle s, \widehat{n}\rangle$ or $\langle \ddot{n}, \widehat{n}\rangle$.

(Claim) ■

The proof of the lemma proceeds, considering cases of formula φ:

- If the formula φ is *first*, then $\mathcal{M}@\mathtt{0} \models \varphi$ by semantics. Re each time-value $\mathtt{t} \in \boldsymbol{T}$ other than $\mathtt{0}$, recall that the state-sketches s_0, s_1, s_2, s_3, … comprise a path. Then with $\mathtt{t} \neq \mathtt{0}$, the state-sketch $s_{\mathtt{t}}$ has $s_{\mathtt{t}-1}$ as a predecessor. Then by the immediately preceding Claim, $\mathit{first} \notin s_{\mathtt{t}}$. Thus, for each $\mathtt{t} \in \boldsymbol{T}$, if $\varphi \in s_{\mathtt{t}}$, then $\mathcal{M}@\mathtt{t} \models \varphi$, as desired.
- If φ is $\neg$*first*, then for $\mathtt{t} \neq \mathtt{0}$, $\mathcal{M}@\mathtt{t} \models \varphi$ by semantics. For $\mathtt{t} = \mathtt{0}$, the presentation of procedure `delineate_model()` implies that state-sketch s_0 contains the formula *first*, so by Lemma 1.3, $\neg\mathit{first} \notin s_0$. Thus, for each $\mathtt{t} \in \boldsymbol{T}$, if $\varphi \in s_{\mathtt{t}}$, then $\mathcal{M}@\mathtt{t} \models \varphi$, as desired.

To address other cases for formula φ, suppose there is at least one formula $\upsilon \prec \varphi$. But additionally, suppose there is no formula $\widetilde{\varphi} \neq \varphi$ such that $\mathit{associated\_formulas}(\widetilde{\varphi}) = \mathit{associated\_formulas}(\varphi)$. Such cases for φ are as follows:

- If φ is *false*, then Lemma 1.3 implies that no state-sketch $s_{\mathtt{t}}$ contains φ, so the desired condition here holds.

- If φ is $\neg$*false*, then the desired condition holds by basic semantics: for any $\mathsf{t} \in \boldsymbol{T}$, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \neg \mathit{false}$.
- If φ is a literal, then for each $\mathsf{t} \in \boldsymbol{T}$, if $\varphi \in s_{\mathsf{t}}$ then $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \varphi$ by the construction of $\boldsymbol{\mathcal{M}}@\mathsf{t}$.
- Suppose formula φ has the form $\left(\bigcirc^{\langle\eta+1\rangle}\psi\right)$. Then because processing has concluded, each edge $\langle s_{\mathsf{t}}, s_{\mathsf{t}+1}\rangle$ (for $\mathsf{t} \in \boldsymbol{T}$) is accurate in the direction `future`. Then if $\varphi = \left(\bigcirc^{\langle\eta+1\rangle}\psi\right) \in s_{\mathsf{t}}$, $(\bigcirc^{\eta}\psi) \in s_{\mathsf{t}+1}$. Now, $\mathit{associated\_formulas}(\bigcirc^{\eta}\psi) \subseteq \mathit{associated\_formulas}\left(\bigcirc^{\langle\eta+1\rangle}\psi\right) = \mathit{associated\_formulas}(\varphi)$. With $\mathit{associated\_formulas}(\bigcirc^{\eta}\psi) \neq \mathit{associated\_formulas}(\varphi)$ assumed here, $(\bigcirc^{\eta}\psi) \prec \varphi$. Then by inductive hypothesis, $(\bigcirc^{\eta}\psi) \in s_{\mathsf{t}+1}$ implies that $\boldsymbol{\mathcal{M}}@(\mathsf{t}+1) \models (\bigcirc^{\eta}\psi)$. Then by semantics, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \left(\bigcirc^{\langle\eta+1\rangle}\psi\right) = \varphi$.
- The case when φ has the form $\left(\ominus^{\langle\eta+1\rangle}\psi\right)$ resembles the preceding case when φ has the form $\left(\bigcirc^{\langle\eta+1\rangle}\psi\right)$. A necessary precondition is that $\mathsf{t} > 0$ so $\mathsf{t} - 1 \in \boldsymbol{T}$ when $\varphi \in s_{\mathsf{t}}$. This precondition holds because immediate constraints are satisfied by Lemma 1.3, the immediate constraint of the formula $\left(\ominus^{\langle\eta+1\rangle}\psi\right) \in s_{\mathsf{t}}$ requires $\neg \mathit{first} \in s_{\mathsf{t}}$, and as discussed above in this proof (in the case when $\varphi = \neg \mathit{first}$), $\neg \mathit{first} \notin s_0$.
- In any other case of formula φ here, suppose $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$, and let $o \in$ `options`(φ) be a set of formulas that are included in s_{t} to satisfy the immediate constraint for φ, by Lemma 1.3. Certainly, by definition of *associated_formulas*(), $\mathit{associated\_formulas}(\psi) \subset \mathit{associated\_formulas}(\varphi)$ for each formula $\psi \in o$. By supposition here, each $\mathit{associated\_formulas}(\psi) \neq \mathit{associated\_formulas}(\varphi)$. Then for each $\psi \in o$, $\psi \prec \varphi$. Then by inductive hypothesis, $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \psi$ for each $\psi \in o$. Then by semantics and analysis of Table I.1 (which is the source of `options()`), $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \varphi$.

Next, suppose that there is at least one formula $\upsilon \prec \varphi$ and that there is a formula $\widetilde{\varphi} \neq \varphi$ such that $\mathit{associated\_formulas}(\widetilde{\varphi}) = \mathit{associated\_formulas}(\varphi)$.

- If φ has the form $\Box\psi$ and $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$, then with φ's immediate constraint satisfied by Lemma 1.3, $\{\psi, \bigcirc\Box\psi\} \subset s_{\mathsf{t}}$. Then with the edge $\langle s_{\mathsf{t}}, s_{\mathsf{t}+1}\rangle$ accurate in the direction `future`, $\bigcirc\Box\psi \in s_{\mathsf{t}}$ implies that $\Box\psi \in s_{\mathsf{t}+1}$. Then $\{\psi, \bigcirc\Box\psi\} \subset s_{\mathsf{t}+1}$. Then $\Box\psi \in s_{\mathsf{t}+2}$, so $\{\psi, \bigcirc\Box\psi\} \subset s_{\mathsf{t}+2}$. Continuing, $\psi \in s_{\mathsf{t}'}$ for every $\mathsf{t}' \geq \mathsf{t}$. Now, $\psi \prec \Box\psi = \varphi$. Then by inductive hypothesis, for every $\mathsf{t}' \geq \mathsf{t}$, $\boldsymbol{\mathcal{M}}@\mathsf{t}' \models \psi$. Then by the semantics of "$\Box$", $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \Box\psi = \varphi$.
- If φ has the form $\bigcirc\Box\psi$ and $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$, then $\Box\psi \in s_{\mathsf{t}+1}$, and then analysis as in the preceding case shows that $\boldsymbol{\mathcal{M}}@(\mathsf{t}+1) \models \Box\psi$. Then, by the semantics of "$\bigcirc$", $\boldsymbol{\mathcal{M}}@\mathsf{t} \models \bigcirc\Box\psi = \varphi$.
- Suppose φ has the form $\Diamond\psi$ and $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$. Then φ's immediate constraint

entails that either $\psi \in s_{\mathsf{t}}$ or $\bigcirc\Diamond\psi \in s_{\mathsf{t}}$. In the former case, with $\psi \prec \Diamond\psi$, inductive hypothesis yields $\mathcal{M}@\mathsf{t} \models \psi$; then, $\mathcal{M}@\mathsf{t} \models \Diamond\psi$ by the semantics of "$\Diamond$". In the latter case (when $\bigcirc\Diamond\psi \in s_{\mathsf{t}}$), $\Diamond\psi \in s_{\mathsf{t+1}}$, hence either $\psi \in s_{\mathsf{t+1}}$ or $\bigcirc\Diamond\psi \in s_{\mathsf{t+1}}$. If $\psi \in s_{\mathsf{t+1}}$, then $\mathcal{M}@\mathsf{t} \models \Diamond\psi$; otherwise, $\Diamond\psi \in s_{\mathsf{t+2}}$. This progression can continue with $\Diamond\psi \in s_{\mathsf{t+3}}$, $s_{\mathsf{t+4}}$, etc. But recall that the sequence of state-sketches $s_0, s_1, s_2, s_3, \ldots$ comprises a finite initial part followed by infinite repetition of a cycle throughout an *mscc* which has passed the tests of procedure `search_graph()`. Suppose the progression here continues to a state-sketch $s_{\tilde{\mathsf{t}}} \in$ *mscc*. (The progression can otherwise terminate only if it finds ψ in a state-sketch, which yields $\mathcal{M}@\mathsf{t} \models \Diamond\psi = \varphi$, which is what is desired anyway.) With $\Diamond\psi \in s_{\tilde{\mathsf{t}}} \in$ *mscc* and *mscc* having passed the tests of procedure `search_graph()`, the formula ψ appears somewhere in *mscc*, i.e. it is present in some state-sketch $s_{\mathrm{f}} \in$ *mscc*. Then as the cycle throughout *mscc* is supposed to be repeated infinitely in the sequence $s_0, s_1, s_2, s_3, \ldots$, let $\mathsf{t_f} \in \boldsymbol{T}$ be such that $\mathsf{t_f} \geq \tilde{\mathsf{t}}$ and $s_{\mathsf{t_f}} = s_{\mathrm{f}}$. Then $\psi \in s_{\mathsf{t_f}}$ with $\mathsf{t_f} \geq \mathsf{t}$. Then $\mathcal{M}@\mathsf{t} \models \Diamond\psi = \varphi$.

- Suppose φ has the form $(\Diamond_{\cong\eta_i}\psi)$ and $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$. This case resembles the preceding case for $\Diamond\psi$, with the added condition that $\mathsf{t_f} - \mathsf{t} \cong \eta_i$. Using the notation of the presentation of function `fulfilling_congruences()` (which starts on page 34), let $\mathsf{t}_{\bar{0}} \in \boldsymbol{T}$ be such that $\mathsf{t}_{\bar{0}} \geq t$ and $s_{\mathsf{t}_{\bar{0}}} = s_{\bar{0}}$. If for each $\mathsf{t}' \in \{\mathsf{t}..(\mathsf{t}_{\bar{0}} - 1)\}$, $\psi \notin s_{\mathsf{t}'}$ or $\mathsf{t}' - \mathsf{t} \not\cong \eta_i$, then immediate constraints plus accuracy of edges entails that $\varphi \in s_{\tilde{\mathsf{t}}}$ for some $\tilde{\mathsf{t}} \geq \mathsf{t}_{\bar{0}}$ with $(\tilde{\mathsf{t}} - \mathsf{t})\ \mathit{MOD}\ \eta_i = 0$. If $(\tilde{\mathsf{t}} - \mathsf{t}_{\bar{0}})\ \mathit{MOD}\ \eta_i = m$, then $m \in \mathit{needing}[i]$. Then in generating a cycle through *mscc*, `delineate_model()` ensured that $\psi \in s_{\mathsf{t_f}}$ for some $\mathsf{t_f} \geq \mathsf{t}_{\bar{0}}$ such that $(\mathsf{t_f} - \mathsf{t}_{\bar{0}})\ \mathit{MOD}\ \eta_i = m$. Then $(\mathsf{t_f} - \mathsf{t})\ \mathit{MOD}\ \eta_i = ((\mathsf{t_f} - \mathsf{t}) + (\mathsf{t}_{\bar{0}} - \mathsf{t}_{\bar{0}}) - (\tilde{\mathsf{t}} - \mathsf{t}))\ \mathit{MOD}\ \eta_i = ((\mathsf{t_f} - \mathsf{t}_{\bar{0}}) + (\tilde{\mathsf{t}} - \mathsf{t}_{\bar{0}}) - (\mathsf{t} - \mathsf{t}))\ \mathit{MOD}\ \eta_i = (m - m + 0)\ \mathit{MOD}\ \eta_i = 0$, i.e. $\mathsf{t_f} - \mathsf{t} \cong \eta_i$. Thus, the added condition is satisfied.
- Suppose φ has the form $\diamondminus\psi$ and $\varphi \in s_{\mathsf{t}}$ for some $\mathsf{t} \in \boldsymbol{T}$. Then φ's immediate constraint entails that either $\psi \in s_{\mathsf{t}}$ or $\ominus\diamondminus\psi \in s_{\mathsf{t}}$. The former possibility $\psi \in s_{\mathsf{t}}$ straightforwardly implies that $\mathcal{M}@\mathsf{t} \models \diamondminus\psi = \varphi$. In the other case, the immediate constraint for the formula $\ominus\diamondminus\psi \in s_{\mathsf{t}}$ entails that $\neg\mathit{first} \in s_{\mathsf{t}}$. Then as above, $\mathsf{t} > 0$. Then further, with the edge $\langle s_{\mathsf{t-1}}, s_{\mathsf{t}}\rangle$ accurate in the direction `past`, $\diamondminus\psi \in s_{\mathsf{t-1}}$. Then again, either $\psi \in s_{\mathsf{t-1}}$, in which case straightforwardly $\mathcal{M}@\mathsf{t} \models \diamondminus\psi = \varphi$, or $\ominus\diamondminus\psi \in s_{\mathsf{t-1}}$. This progression can continue, but certainly not past s_0, and it stops only when it finds a state-sketch containing ψ, at which point straightforwardly $\mathcal{M}@\mathsf{t} \models \diamondminus\psi$.[7]
- All other cases resemble the preceding ones.

[7] It may be interesting to consider what the algorithm does in the 'pathological' version of the situation here. For example, suppose the formula $\varphi_{\mathrm{checking}}$ is $(\Box\neg p \wedge \bigcirc\bigcirc\bigcirc\diamondminus p)$. Then the algorithm constructs a graph whose most

(Lemma) ■

The proof of the theorem continues as follows, showing that the originally given formula $\varphi_{\text{checking}}$ occurs appropriately in a state-sketch so the preceding Lemma can be applied to it.

According to the presentation of procedure `delineate_model()`, the state-sketch s_0 contains the formula *first*. Considering procedure `init_graph()` and function `add_options*()`, processing begins with the originally given formula $\varphi_{\text{checking}}$ present in every state-sketch other than *blank*. Subsequently, each new state-sketch $\widehat{n}$ contains a relevant prior state-sketch n. Then every state-sketch contains one of the initial state-sketches. Now, any state-sketch that contains *blank* contains $\neg$*first* and hence does not contain *first*. Then with state-sketch s_0 containing *first*, it contains some initial state-sketch other than *blank*. Since each such other initial state-sketch contains $\varphi_{\text{checking}}$, so does s_0. Then by the preceding Lemma, $\boldsymbol{\mathcal{M}}@0 \models \varphi_{\text{checking}}$, i.e. model $\boldsymbol{\mathcal{M}}$ satisfies the formula $\varphi_{\text{checking}}$.

(Theorem 2: Sufficiency) ■

---

significant state-sketches comprise the following sequence:

$$\left\{\begin{array}{l} \mathit{first},\ \varphi_{\text{checking}}, \\ \Box\neg p,\ \neg p,\ \bigcirc\Box\neg p, \\ \bigcirc\bigcirc\bigcirc\diamondminus p \end{array}\right\} \qquad \left\{\begin{array}{l} \neg\mathit{first},\ \Box\neg p, \\ \neg p,\ \bigcirc\Box\neg p, \\ \bigcirc\bigcirc\diamondminus p, \\ \diamondminus p,\ \ominus\diamondminus p \end{array}\right\} \longrightarrow \left\{\begin{array}{l} \neg\mathit{first},\ \Box\neg p, \\ \neg p,\ \bigcirc\Box\neg p, \\ \bigcirc\diamondminus p, \\ \diamondminus p,\ \ominus\diamondminus p \end{array}\right\} \longrightarrow \left\{\begin{array}{l} \neg\mathit{first},\ \Box\neg p, \\ \neg p,\ \bigcirc\Box\neg p, \\ \diamondminus p,\ \ominus\diamondminus p \end{array}\right\} \longrightarrow \left\{\begin{array}{l} \neg\mathit{first},\ \Box\neg p, \\ \neg p,\ \bigcirc\Box\neg p \end{array}\right\}$$

— with the edge between the first two state-sketches removed. Then, when procedure `search_graph()` searches the graph for strongly connected components *mscc* — with the search starting from state-sketches containing *first* — it can't reach any (appropriate) *mscc*. Consequently, the algorithm reports that this formula $\varphi_{\text{checking}}$ is unsatisfiable — as it is.

Chapter I.3

Complexity of the Algorithm

Chapter I.2's Theorem 0 (Termination) shows that this algorithm requires only a finite amount of work. This chapter estimates the amount.

The algorithm here is designed to maximize efficiency, but in the worst cases, it does as much as (and no more than) the amount of work of classical algorithms such as [LicPZ85]. As has been mentioned above, the complexity here for metric temporal logic differs from the complexity specified in [He91] because of differences in the semantics.

I.3.1 Sizes of Elements

The graph comprising vertices V_{final} and edges E_{final} plus $\overline{E}_{\text{final}}$ which is constructed by this algorithm naturally has $|E_{\text{final}} \cup \overline{E}_{\text{final}}| \leq |V_{\text{final}}|^2$. Then, since the vertices V_{final} are sets of formulas collected in $\mathit{associated\_formulas}(\varphi_{\text{checking}})$:

$$|V_{\text{final}}| \leq \left|\wp\big(\mathit{associated\_formulas}(\varphi_{\text{checking}})\big)\right| = 2^{|\mathit{associated\_formulas}(\varphi_{\text{checking}})|}.$$

Then, the crucial question is: what is the cardinality $|\mathit{associated\_formulas}(\varphi_{\text{checking}})|$, say as a function of some measure of formula $\varphi_{\text{checking}}$? With the function $\mathit{size}()$: formulas $\rightarrow \boldsymbol{N}$ denoting this measure, the following shows that $|\mathit{associated\_formulas}(\varphi)| < c \cdot \mathit{size}(\varphi)$, for a constant c.

Unfortunately, the operators "$\Leftrightarrow$" and "$\otimes$" present difficulties. For example, consider a formula $(\psi \Leftrightarrow \xi)$ with, say, $\mathit{size}(\psi) = i$ and $\mathit{size}(\xi) = j$, and suppose $\mathit{size}(\psi \Leftrightarrow \xi) = i + 1 + j$ — which seems natural. The set $\mathit{associated\_formulas}(\psi \Leftrightarrow \xi)$ involves the formulas ψ, ξ, $\neg\psi$, and $\neg\xi$, and hence their $\mathit{associated\_formulas}()$. Straightforwardly applying the size-formula, these sub-$\mathit{associated\_formulas}()$ altogether would number approximately $c{\cdot}i + c{\cdot}j + c{\cdot}(1 + i) + c{\cdot}(1 + j) = 2{\cdot}c{\cdot}(i + 1 + j) = 2{\cdot}c{\cdot}\mathit{size}(\psi \Leftrightarrow \xi)$ — double the purported value "$|\mathit{associated\_formulas}(\psi \Leftrightarrow \xi)| <$

$c \cdot size(\psi \Leftrightarrow \xi)$"! The situation with "$\otimes$" is similar. Consequently, in the following, the $size()$ of a "$\Leftrightarrow$"- or "$\otimes$"-formula is considered to be double its apparent size.[1]

Aside from the preceding special consideration for the operators "$\Leftrightarrow$" and "$\otimes$", the $size()$ of a formula φ is the sum of the number of propositions and operators in φ (including operators' repetitions due to large arities) plus the numbers that occur as numerals in superscripts and subscripts in φ. For example, $size\big(\mathit{first} \wedge \neg\Diamond p \wedge (\bigcirc^3 q)\big) = 11$.

The following claim specifies how the desired amount $|associated\_formulas(\varphi)|$ may be estimated from $size(\varphi)$:

Claim: For any formula φ, $|associated\_formulas(\varphi)| \leq 5 \cdot size(\varphi) - 2$ — with strict inequality ("$<$"), actually, for every formula except *false* and the propositions.

Proof: This proof proceeds by induction on the sizes of formulas.

- This claim clearly holds if φ is *false*, $\neg$*false*, a literal, *first*, or $\neg$*first*. For example, $associated\_formulas(p) = \{p, \neg \mathit{first}, \mathit{first}\}$ and $size(p) = 1$.
- If φ is $(\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)$, then $size(\varphi) = k-1 + size(\psi_1) + size(\psi_2) + \ldots + size(\psi_k)$. Also:

$$\begin{aligned} associated\_formulas(\varphi) \;=\; & \{\varphi, \neg \mathit{first}, \mathit{first}\} \\ & \cup\; associated\_formulas(\psi_1) \\ & \cup\; associated\_formulas(\psi_2) \\ & \vdots \\ & \cup\; associated\_formulas(\psi_k)\,, \end{aligned}$$

so:

$$\begin{aligned} |associated\_formulas(\varphi)| \;\leq\; & 3 \\ & +\; |associated\_formulas(\psi_1)| \\ & +\; |associated\_formulas(\psi_2)| \\ & \vdots \\ & +\; |associated\_formulas(\psi_k)|\,. \end{aligned}$$

Then by inductive hypothesis:

$$\begin{aligned} |associated\_formulas(\varphi)| \;\leq\; & 3 \\ & +\; 5 \cdot size(\psi_1) - 2 \\ & +\; 5 \cdot size(\psi_2) - 2 \\ & \vdots \\ & +\; 5 \cdot size(\psi_k) - 2\,. \end{aligned}$$

[1] An alternative resolution of this difficulty is to simply refuse to handle formulas that contain "$\Leftrightarrow$" or "$\otimes$". This alternative approach is plausible because any "$\Leftrightarrow$"- or "$\otimes$"-formula can be rewritten as an equivalent formula not using "$\Leftrightarrow$" or "$\otimes$" (as specified in Table II.9, on page 101).

Then with the arity $k \geq 2$:

$$\begin{aligned} |associated\_formulas(\varphi)| &< 5{\cdot}(k-1) \\ &\quad + 5{\cdot}size(\psi_1) \\ &\quad + 5{\cdot}size(\psi_2) \\ &\quad \vdots \\ &\quad + 5{\cdot}size(\psi_k) \\ &\quad - 2 \\ &= 5 \cdot size(\varphi) - 2\,, \end{aligned}$$

so the claim holds in this case.

– If φ is $(\psi \Leftrightarrow \xi)$, then $size(\varphi) = 2{\cdot}[size(\psi) + 1 + size(\xi)]$ and:

$$\begin{aligned} associated\_formulas(\varphi) &= \{\varphi, \neg first, first\} \\ &\quad \cup associated\_formulas(\psi) \\ &\quad \cup associated\_formulas(\xi) \\ &\quad \cup associated\_formulas(\neg\psi) \\ &\quad \cup associated\_formulas(\neg\xi)\,. \end{aligned}$$

Then by inductive hypothesis:

$$\begin{aligned} |associated\_formulas(\varphi)| &\leq 3 \\ &\quad + 5{\cdot}size(\psi) - 2 \\ &\quad + 5{\cdot}size(\xi) - 2 \\ &\quad + 5{\cdot}size(\neg\psi) - 2 \\ &\quad + 5{\cdot}size(\neg\xi) - 2\,. \end{aligned}$$

If the formula ψ is not a negation, then naturally $size(\neg\psi) = 1 + size(\psi)$. If ψ is a negation $\neg\tilde{\psi}$ (with $\tilde{\psi}$ not a negation), then $\neg\psi$ is $\tilde{\psi}$ and $size(\neg\psi) = size(\tilde{\psi}) = size(\neg\tilde{\psi}) - 1 = size(\psi) - 1 < 1 + size(\psi)$. Thus, regardless, $size(\neg\psi) \leq 1 + size(\psi)$. Similarly, $size(\neg\xi) \leq 1 + size(\xi)$. Then:

$$\begin{aligned} |associated\_formulas(\varphi)| &< 5{\cdot}size(\psi) \\ &\quad + 5{\cdot}size(\xi) \\ &\quad + 5{\cdot}[1 + size(\psi)] \\ &\quad + 5{\cdot}[1 + size(\xi)] \\ &\quad - 2 \\ &= 5{\cdot}\big(2{\cdot}[size(\psi) + 1 + size(\xi)]\big) - 2 \\ &= 5{\cdot}size(\varphi) - 2\,. \end{aligned}$$

Thus, the claim holds in this case.

– If φ is $\neg(\xi \mathsf{S} \psi)$, then $size(\varphi) = 1 + \big(size(\xi) + 1 + size(\psi)\big)$ and:

$$\begin{aligned} associated\_formulas(\varphi) = \ &\big\{ \varphi, \neg first, first, \\ &\ [\neg\psi \mathsf{B} (\neg\xi \wedge \neg\psi)], \\ &\ (\neg\xi \wedge \neg\psi), \neg\ominus\neg[\neg\psi \mathsf{B} (\neg\xi \wedge \neg\psi)], \\ &\ \ominus[\neg\psi \mathsf{B} (\neg\xi \wedge \neg\psi)] \big\} \\ &\cup associated\_formulas(\neg\xi) \\ &\cup associated\_formulas(\neg\psi)\,. \end{aligned}$$

Note that both $associated\_formulas(\neg\xi)$ and $associated\_formulas(\neg\psi)$ contain the formulas $\neg first$ and $first$. Then:

$$\begin{array}{rcl} associated\_formulas(\varphi) & = & \{\ \varphi, \\ & & \quad [\neg\psi\ \mathsf{B}\ (\neg\xi \wedge \neg\psi)], \\ & & \quad (\neg\xi \wedge \neg\psi),\ \neg\ominus\neg[\neg\psi\ \mathsf{B}\ (\neg\xi \wedge \neg\psi)], \\ & & \quad \ominus[\neg\psi\ \mathsf{B}\ (\neg\xi \wedge \neg\psi)]\ \} \\ & & \cup\ \big[associated\_formulas(\neg\xi)\ -\ \{\neg first,\ first\}\big] \\ & & \cup\ associated\_formulas(\neg\psi)\,. \end{array}$$

If the formula ξ is either $\neg false$ or the negation of a proposition, then $\neg\xi$ is $false$ or a proposition, respectively. Then, $size(\neg\xi) = 1$ and $|associated\_formulas(\neg\xi)| = 3$, so $|associated\_formulas(\neg\xi)| \leq 5{\cdot}[1 + size(\xi)] - 3$. In all other cases for ξ, inductive hypothesis gives $|associated\_formulas(\neg\xi)| < 5{\cdot}size(\neg\xi) - 2 \leq 5{\cdot}[1 + size(\xi)] - 2$, i.e. again $|associated\_formulas(\neg\xi)| \leq 5{\cdot}[1 + size(\xi)] - 3$. The situation for ψ is similar. Then:

$$\begin{array}{rcl} |associated\_formulas(\varphi)| & \leq & 5 \\ & & +\ \big(5{\cdot}[1 + size(\xi)] - 3\big) - 2 \\ & & +\ 5{\cdot}[1 + size(\psi)] - 3 \\ & = & 5{\cdot}[1 + size(\xi) + 1 + size(\psi)] - 3 \\ & = & 5{\cdot}size(\varphi) - 3\,, \end{array}$$

i.e. $|associated\_formulas(\varphi)| < 5 \cdot size(\varphi) - 2$, as desired.

– If φ is $(\xi\ \mathsf{B}_{\cong\eta}\ \psi)$, then $size(\varphi) = 1 + \big(size(\xi) + 1 + \eta + size(\psi)\big)$ and:

$$\begin{array}{rcl} associated\_formulas(\varphi) & = & \{\ \varphi,\ \neg first,\ first,\ (\boxminus_{\leq\langle\eta-1\rangle}\,\xi), \\ & & \neg\ominus\neg(\boxminus_{\leq\langle\eta-2\rangle}\,\xi),\ \ominus(\boxminus_{\leq\langle\eta-2\rangle}\,\xi),\ (\boxminus_{\leq\langle\eta-2\rangle}\,\xi), \\ & & \neg\ominus\neg(\boxminus_{\leq\langle\eta-3\rangle}\,\xi),\ \ominus(\boxminus_{\leq\langle\eta-3\rangle}\,\xi),\ (\boxminus_{\leq\langle\eta-3\rangle}\,\xi), \\ & & \quad\vdots \\ & & \neg\ominus\neg(\boxminus_{\leq 0}\,\xi),\ \ominus(\boxminus_{\leq 0}\,\xi),\ (\boxminus_{\leq 0}\,\xi), \\ & & \neg\Big(\ominus^{\eta}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)\Big),\ \ominus\neg(\ominus^{\langle\eta-1\rangle}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)), \\ & & \neg(\ominus^{\langle\eta-1\rangle}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)),\ \ominus\neg(\ominus^{\langle\eta-2\rangle}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)), \\ & & \quad\vdots \\ & & \neg(\ominus^{2}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)),\ \ominus\neg(\ominus\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)), \\ & & \neg\ominus(\xi\ \mathsf{B}_{\cong\eta}\ \psi),\ \ominus(\xi\ \mathsf{B}_{\cong\eta}\ \psi)\ \} \\ & & \cup\ associated\_formulas(\xi) \\ & & \cup\ associated\_formulas(\psi)\,, \end{array}$$

so, by inductive hypothesis:

$$\begin{array}{rcl} |associated\_formulas(\varphi)| & \leq & 4 \\ & & +\ 3(\eta - 1) \\ & & +\ 2\eta \\ & & +\ 5{\cdot}size(\xi) - 2 \\ & & +\ 5{\cdot}size(\psi) - 2 \\ & < & 5{\cdot}[size(\xi) + 1 + \eta + size(\psi)] - 2 \\ & = & 5{\cdot}size(\varphi) - 3\,. \end{array}$$

Thus, the claim holds in this case.

– All other cases resemble the preceding ones.

(Claim) ■

Clearly, this Claim's factor 5 is an overestimate. One way to provably achieve a smaller number is to analytically reduce options-formulas, e.g. having the options for $\neg(\xi \mathsf{S} \psi)$ be $\{\neg\psi, \neg\xi\}$ and $\{\neg\psi, \neg\ominus(\xi \mathsf{S} \psi)\}$ instead of $\{\neg\psi \mathsf{B} (\neg\xi \wedge \neg\psi)\}$. An alternative perspective would involve determining a factor for each operator and then combining these factors as appropriate for each formula φ's combination of operators. A final consideration is the sharing (which enables savings) of *associated_formulas*() that is typical of formulas which people care to check.

The context of consideration of $|associated\_formulas()|$ is the bound $|V_{\text{final}}| \leq 2^{|associated\_formulas(\varphi_{\text{checking}})|}$. But this bound can be reduced.

The most important reduction in the bound involves procedure `add_options*()`'s reduction of most formulas to equivalent options. The formulas that remain after this reduction are *first*, $\neg$*first*, the literals, "$\bigcirc$"-formulas, and "$\ominus$"-formulas; all other reducible formulas can be ignored as specifiers of elements of $|V|$. For example, if the state-sketch $\{\neg first, p, \bigcirc\Box p, \Box p\}$ exists, then a state-sketch $\{\neg first, p, \bigcirc\Box p\}$ would not be added; the former state-sketch would be used wherever the latter might be desired.[2] Typically, these formulas to which all others reduce comprise approximately half of the total number of formulas; then, $|V_{\text{final}}|$ diminishes accordingly (e.g. if indeed the formulas are precisely halved, then $|V_{\text{final}}|$ is the square root of what it would be otherwise).

A further consideration is that the algorithm does not construct state-sketches containing obviously contradictory pairs of formulas $\{\psi, \neg\psi\}$. Let the set *ir_a** comprise all the irreducible formulas in $associated\_formulas(\varphi_{\text{checking}})$. Then, let:

$$\begin{aligned}
n &= \{\text{all the formulas } \psi \in ir_a* \text{ such that } \neg\psi \in ir_a*, \text{ with } \neg\psi \text{ really being a negation}\} \\
&\quad\ (\text{negated formulas}), \\
\tilde{n} &= \{\text{all the formulas } \neg\psi \in ir_a* \text{ such that } \psi \in n\}, \quad \text{and:} \\
u &= ir_a* - (n \cup \tilde{n}) \qquad (\text{unnegated formulas}).
\end{aligned}$$

Notice that $|n| = |\tilde{n}|$. Then the bound $|V_{\text{final}}| \leq 2^{|ir\_a*|} = 2^{|n|+|\tilde{n}|+|u|} = 2^{|n|+|n|+|u|} = 4^{|n|} \cdot 2^{|u|}$ can be reduced to $|V_{\text{final}}| \leq 3^{|n|} \cdot 2^{|u|}$ since, for any element of n, either it or its negation or neither — never both — is or are present in any element of $|V|$.

The approximate bounds here on sizes of features are indeed reached. For example, if the

[2] Keeping reducible formulas present is useful to facilitate checking of fulfillment. For example, in checking whether the formula $\Diamond\Box p$ is fulfilled, it's easier to check for $\Box p$ rather than for the pair $\{p, \bigcirc\Box p\}$.

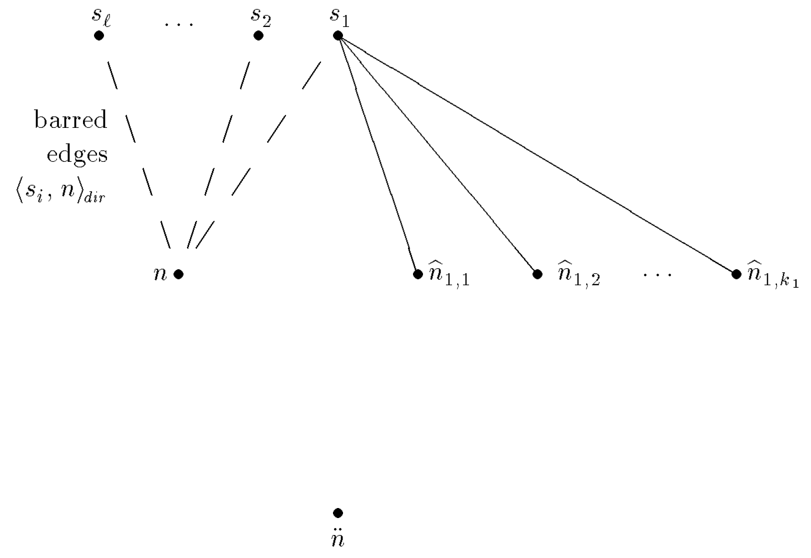
formula $\varphi_{\text{checking}}$ is $\Box(\Diamond p_1 \wedge \Diamond p_2 \wedge \ldots \wedge \Diamond p_k)$, then $|V_{\text{final}}| = 2^{k+1} + 1 \approx \sqrt{2^{[size(\varphi_{\text{checking}})]}}$ and $|E_{\text{final}} \cup \overline{E}_{\text{final}}| = 2^{k+1} \cdot (2^k + 1) + 1 \approx |V_{\text{final}}|^2$.

I.3.2 Complexity of Operations

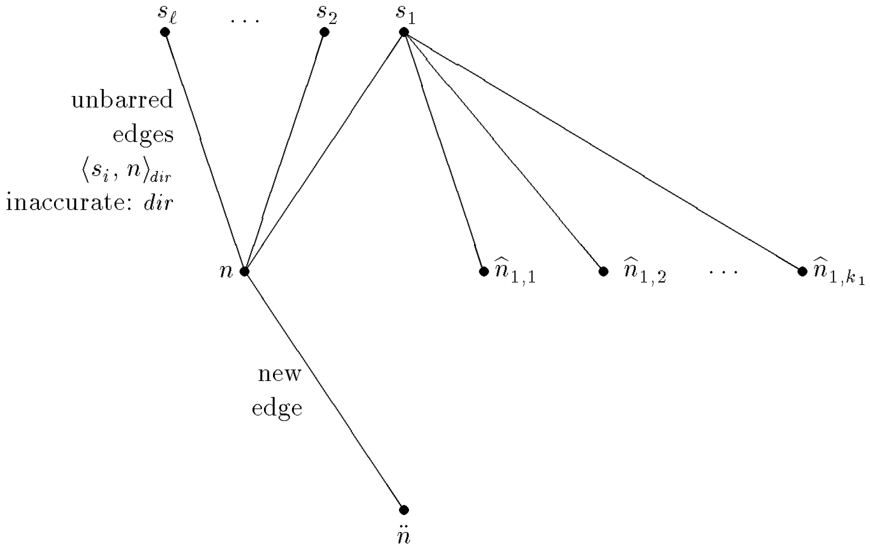
The algorithm's first operations, in procedure `init_graph()`, clearly all are negligible except for the execution of function `add_options*()`. And this execution may be lumped with the executions of `add_options*()` in procedure `elaborate_graph()`, as if that procedure were doing one more iteration of its main loop with a pseudo-edge "$\langle\{\bigcirc\varphi_{\text{checking}}\},\ \{\mathit{first}\}\rangle$", i.e. as if $|E_{\text{final}} \cup \overline{E}_{\text{final}}|$ were larger by one.

As for procedure `elaborate_graph()`, clearly its main loop "for each edge ..." must address each edge if only to determine whether the edge is accurate. Then *prima facie*, the loop's number of iterations is $|E_{\text{final}} \cup \overline{E}_{\text{final}}|$. (As mentioned above (in the proof of Claim 1.1.2), each edge is accurate in at least one direction, so the amount of processing here is not multiplied by the number of directions.)

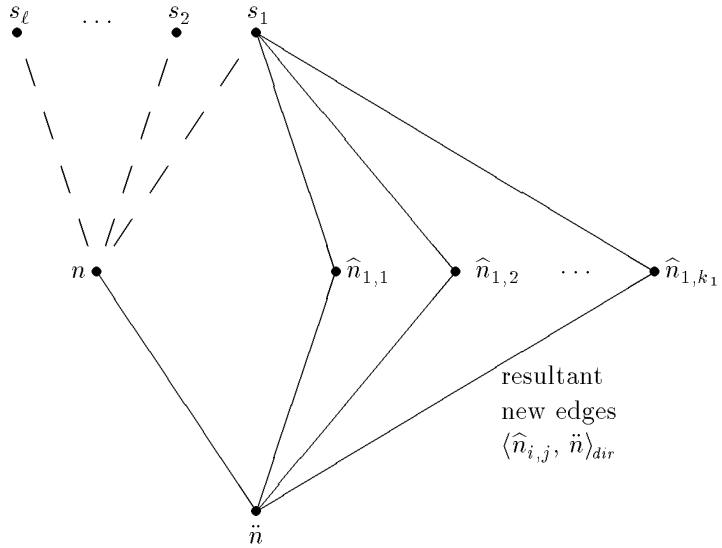
But procedure `unbar_edges()` causes reprocessing of edges. To understand how much, consider a situation such as the following: the edges $\langle s_i, n\rangle_{dir}$ are inaccurate in the direction dir, and they have been processed, engendering vertices $\widehat{n}_{i,j}$ (shown here only for $i = 1$); the direction dir is downward:



Now, suppose a new edge $\langle n, \ddot{n}\rangle_{dir}$ is created, at which occasion `unbar_edges(`n, dir`)` is executed. Then each barred edge $\langle s_i, n\rangle_{dir}$ is unbarred:



Then, each edge $\langle s_i, n\rangle_{dir}$ is reprocessed. As is the intent of unbarring edges, that first new edge $\langle n, \ddot{n}\rangle_{dir}$ engenders further new edges $\langle \widehat{n}_{i,j}, \ddot{n}\rangle_{dir}$ (shown here only for $i = 1$):



Reprocessing does not produce new edges if they already exist in $E \cup \overline{E}$. In one extreme case, reprocessing of the edge $\langle s_i, n\rangle_{dir}$ produces zero new edges; in the other extreme case, all of the k_i edges $\langle \widehat{n}_{i,1}, \ddot{n}\rangle_{dir}$, $\langle \widehat{n}_{i,2}, \ddot{n}\rangle_{dir}$, $\ldots$, and $\langle \widehat{n}_{i,k_i}, \ddot{n}\rangle_{dir}$ are new. A conservative estimate which seems reasonable is that reprocessing $\langle s_i, n\rangle_{dir}$ produces one new edge $\langle \widehat{n}_i, \ddot{n}\rangle_{dir}$. Then subsequently, this new edge will be processed, requiring some amount of work. With the amount of work required to reprocess the edge $\langle s_i, n\rangle_{dir}$ considered similar to the amount required to process the edge $\langle \widehat{n}_i, \ddot{n}\rangle_{dir}$, it's possible to count the reprocessing work as simply doubling the normal processing work. This, then, is the effect that procedure `unbar_edges()` has on the amount of all processing: at most doubling it.[3]

The next question is: How much work is required during each iteration of procedure `elaborate_graph()`'s main loop?

Well, first the edge to be processed must be found. Naturally, the scheme used for storing the graph's elements (V, E) determines the amount of work here. With vertices indexed via hashing and edges simply listed with vertices, the amount of work here is a constant.

Next, determining whether an edge $\langle s, n\rangle_{dir}$ is accurate in the direction dir involves collecting all of s's formulas that have the form $\big([neighbor\_op(dir)]^{\langle\eta+1\rangle}\,\psi\big)$ into `neighbor_formulas(`s`, `dir`)` and then checking whether `neighbor_formulas(`s`, `dir`)` $\in n$. These operations are linear in the sizes of the sets of formulas, which are state-sketches, and state-sketches are subsets of $associated\_formulas(\varphi_{\text{checking}})$, whose size is linear in the size of $\varphi_{\text{checking}}$ according to the preceding Section; so these operations require an amount of work which is linear in the size of the originally given formula $\varphi_{\text{checking}}$.

The next operation is execution of function `add_options*()`. That function combines divers `options()`-sets, producing a multiplicity of state-sketches. Then clearly, the amount of work done here can be expressed as the amount of work per result times the number of results. Well, as each result's relevant chain of `add_options*()`'s recursive sub-calls proceeds, the result is gradually constructed (in *bases*), with each formula $\varphi \in associated\_formulas(\varphi_{\text{checking}})$ being considered (among the options) for inclusion at most once. Then this amount of work per result is linear in the size of $associated\_formulas(\varphi_{\text{checking}})$[4] and hence linear in the size of the originally given formula $\varphi_{\text{checking}}$. Then, as these results are state-sketches included in V, it's reasonable to count the work done in `add_options*()` for these (new) state-sketches rather than for the edge that was involved in their creation.[5] Thus, `add_options*()` overall contributes an amount of work which is $O\big(size(\varphi_{\text{checking}})\big)$ for each element of V_{final}.

Similarly to the preceding treatment of the work of `add_options*()`, the work of establishing the multiplicity of edges may be counted with the newly established edges. This work is clearly constant per edge.

The other operations of procedure `elaborate_graph()`'s main loop require constant amounts of work.

Summarizing, procedure `elaborate_graph()` (with procedure `init_graph()`) requires an amount of work which is $O\Big(\big(|E_{\text{final}} \cup \overline{E}_{\text{final}}| + |V_{\text{final}}|\big) \cdot size(\varphi_{\text{checking}})\Big)$.

[3] Running the algorithm with procedure `unbar_edges()` activated and then deactivated (just observing processing-time — ignoring results as they are not necessarily correct) confirms this estimate.

[4] Conceivably, the depth of the chain of recursive calls of `add_options*()` would be a factor, but proper management of *additions* makes the recursive processing basically a linear progression through formulas.

[5] Results can be repeated, but experimentation with an implementation of the algorithm confirms that such repetition is rare, multiplying the amount of work by only a small factor.

Procedure `search_graph()` needs to find the maximal strongly connected components (*msccs*) of the graph. [AhHU74] specifies the complexity of this work as $O\big(MAX(|V_{\text{final}}|, |E_{\text{final}}|)\big)$. Then, with each *mscc*, first fulfillment of unsubscripted $\Diamond$- and U-formulas is checked; clearly, this operation requires an amount of work which is linear in the quantity of such formulas, which is linear in the size of the originally given formula $\varphi_{\text{checking}}$. (The size of the *mscc* is another factor, but it can be totaled for all *msccs* as the size of the entire graph.) When formulas φ_i having either the form $(\Diamond_{\cong\eta_i}\ \psi_i)$ or the form $(\xi\ \mathsf{U}_{\cong\eta_i}\ \psi_i)$ are present, the function `fulfilling_congruences()` searches *mscc* more thoroughly. Since the state of any branch of this search is determined by a vertex in *mscc* and the contents of the array *branch_reached*[] — whose element *branch_reached*$[i]$ ranges from 0 to η_i−1 — the amount of work required for the search is limited by the product of the (distinct) numbers η_i (and the size of *mscc*, but again this factor can be expressed in the total). Lastly, if it is executed, procedure `delineate_model()` searches the satisfactory *mscc* using a search-algorithm which resembles breadth-first search, requiring an amount of work which is linear in the quantity of the *mscc*'s edges if no congruences are involved or like the amount of work of `fulfilling_congruences()` if congruences are involved. Summarizing, procedure `search_graph()` requires an amount of work which is $O\big(MAX(|V_{\text{final}}|, |E_{\text{final}}|) \cdot size(\varphi_{\text{checking}})\big)$ generally, with additional factors being the numbers η that appear in formulas $(\Diamond_{\cong\eta}\ \psi)$ or $(\xi\ \mathsf{U}_{\cong\eta}\ \psi)$, if any.

Summarizing overall for procedure `satisfy()`, the complexity is $O\Big(\big(|V_{\text{final}}| + |E_{\text{final}} \cup \overline{E}_{\text{final}}|\big) \cdot size(\varphi_{\text{checking}})\Big)$, with additional factors being the numbers η that appear in formulas $(\Diamond_{\cong\eta}\ \psi)$ or $(\xi\ \mathsf{U}_{\cong\eta}\ \psi)$, if any. Considering that $|V_{\text{final}}| \leq 2^{[5 \cdot size(\varphi_{\text{checking}})]}$ so $|E_{\text{final}} \cup \overline{E}_{\text{final}}| \leq |V_{\text{final}}|^2 \leq 2^{[c \cdot size(\varphi_{\text{checking}})]}$, for a constant c, the complexity may alternatively be expressed as $O\Big(size(\varphi_{\text{checking}}) \cdot \tilde{c}^{[size(\varphi_{\text{checking}})]}\Big)$, for a constant $\tilde{c}$.[6]

In comparing the complexity here to that of other algorithms, one should be aware of the base-number upon which linearity versus exponentiality is determined. With $size(\varphi_{\text{checking}})$ as the base-number, the complexity here is exponential as discussed by Sistla and Clarke [SiC85]. With the size of the graph admitted as a base-number, the complexity here is linear as for the system of Clarke, Emerson, and Sistla [ClES86]. (See also the work of Clarke, Grumberg, and Hamaguchi [ClGH94] re this issue.)

[6] An implementation of the algorithm has indeed obeyed these bounds.

Chapter I.4

Examples of Application of the Algorithm

This algorithm has been implemented;[1] here are some examples of its use.

I.4.1 An Introductory Example

As in Section I.1.2, suppose the formula $\varphi_{\text{checking}}$ is $(\bigcirc\boxminus p \vee \Diamond q)$. This formula can be expressed in ASCII characters as "`(O[-]p \/ <>q)`". Given this input, the implementation produces the following output:

```
SATISFIABLE:

Leading states:
0.  { p    }
1.  { p    }

Repeat:
2.  {}
```

This output delineates a model of the given formula as on page 28 in Section I.1.2, specifying that such a model should have its states satisfying proposition p at time 0, satisfying p again at time 1, and otherwise arbitrary (i.e. assignments to p at times 2ff. are arbitrary, and assignments to q at all times are arbitrary); such a model does indeed satisfy the originally given formula $\varphi_{\text{checking}} = (\bigcirc\boxminus p \vee \Diamond q)$.

[1]This implementation is freely available, for any UNIX system. To get the implementation, send electronic mail to `mcguire_hugh@cs.stanford.edu`. (Even after I leave Stanford, mail sent to this address should reach me.)

I.4.2 Correction of a Specification of a Circuit

[WiP89] used an early version of this implementation [SheP89] to detect an error in a specification of a circuit presented by Gordon [Gor86]. The circuit's desired behavior was that of an edge-triggered D-type flip-flop; abstractly, such a circuit's behavior is as follows: The circuit's boolean input is d, and its boolean output is q; a boolean clock-input ck also impinges upon the circuit. Under certain conditions, the value of the input d should be transferred to the output q:

[Gor86]'s conditions for q getting the value of d are as follows. Some terminology here is that "rising edges" of the clock's sequence of values correspond to changes of the clock's value from `false` to `true`; the occurrence of such a rise is denoted by the expression "$ck\uparrow$". If [1] a clock-rise $ck\uparrow$ occurs, [2] the value of d has been stable — say being "$\_val$", some specific value which is either `true` or `false` — for $\mathtt{c}_1$ units of time before this clock-rise $ck\uparrow$, and [3] the next clock-rise $ck\uparrow$ does not follow that first one for at least $\mathtt{c}_2$ units, then the value of q will be that value $\_val$ (i.e. the value of d which was held stable) from $\mathtt{c}_3$ units of time after that first clock-rise to $\mathtt{c}_4$ units of time after the second clock-rise.

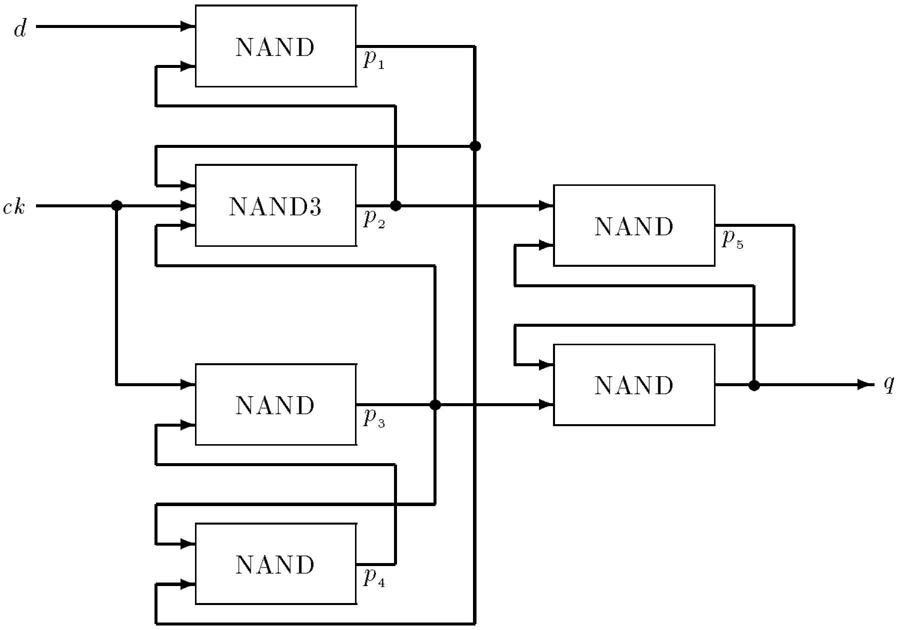
These specifications can be expressed in temporal logic. But this expression involves requiring d to be stable for $\mathtt{c}_1$ units of time before the first clock-rise. The subscripted operator "$\boxminus_{\leq \mathtt{c}_1}$" would seem appropriate for this circumstance, but it is not since it does not enforce the condition at times near $\mathtt{0}$. Consequently, it's convenient to introduce a new temporal operator "$\boxleft$", pronounced "retroactively", which always has a subscript "$_{\leq\eta}$" or "$_{<\eta}$", and whose semantics is the same as that of "$\boxminus$" with the added condition that for a state $\mathcal{M}$@t to satisfy a formula $(\boxleft_{\leq\eta}\,\psi)$ or $(\boxleft_{<\eta}\,\psi)$, it must be true that $t \geq \eta$ or $t \geq \eta - 1$ (and $\eta > 0$), respectively. The algorithm here processes this new operator "$\boxleft$" as follows: Any formula $(\boxleft_{<0}\,\psi)$ is simplified to *false*, and any formula $(\boxleft_{<\langle\eta+1\rangle}\,\psi)$ is changed to $(\boxleft_{\leq\eta}\,\psi)$. Then, Table I.1 (i.e. `options()`) is extended to include the following entries:

Formula	Option #1	Further Options (if any)
$\boxleft_{\leq 0}\,\psi$	$\{\psi\}$	
$\boxleft_{\leq\langle\eta+1\rangle}\,\psi$	$\{\psi,\ \ominus(\boxleft_{\leq\eta}\,\psi)\}$	
$\neg\boxleft_{\leq 0}\,\psi$	$\{\neg\psi\}$	
$\neg\boxleft_{\leq\langle\eta+1\rangle}\,\psi$	$\{\neg\psi\}$	$\{\neg\ominus(\boxleft_{\leq\eta}\,\psi)\}$

With this new operator $\boxleft$, the specification above can be expressed as follows: a clock-rise $ck\uparrow$ can be expressed as $(ck \wedge \ominus\neg ck)$, and then the conditions (above) for q := d can be expressed as the following formula φ_{s}:

$$\Big(ck{\uparrow} \wedge [\boxed{\blacktriangleleft}_{\leq \mathsf{c}_1} (d \Leftrightarrow \_val)] \wedge \bigcirc[\Box_{<\mathsf{c}_2} \neg(ck{\uparrow})]\Big) \Rightarrow \Big(\bigcirc^{\mathsf{c}_3} \big[(q \Leftrightarrow \_val) \ \mathsf{A} \ \big(ck{\uparrow} \wedge [\Box_{<\mathsf{c}_4} (q \Leftrightarrow \_val)]\big)\big]\Big)$$

As is the intent of a specification, it can be applied to a proposal (for a circuit) to ensure that the proposal is correct. Such a proposal for a D-type flipflop is as follows, using NAND-gates; given some inputs at one instant of time, a NAND-gate's output at the next instant is the negation of the conjunction of the given inputs.



(The p_is are useful when referring to the outputs of the NAND-gates.) [Gor86] claimed that this circuit satisfies the specification above with $\mathsf{c}_1 = 2$, $\mathsf{c}_2 = 3$, $\mathsf{c}_3 = 4$, and $\mathsf{c}_4 = 1$.

[WiP89] checked [Gor86]'s claim using the aforementioned implementation, encoding the operation of the circuit as a formula φ_{c} and then verifying the super-formula $(\Box\varphi_{\mathsf{c}} \Rightarrow \Box\varphi_{\mathsf{s}})$ — i.e. checking the satisfiability of the formula $\neg(\Box\varphi_{\mathsf{c}} \Rightarrow \Box\varphi_{\mathsf{s}})$. The implementation here allows the following expression of this material: (The character "%" begins comments.)

```
% "gor86.tl.c"

% Choose a value, "TRUE" or "FALSE" --- here, "TRUE":
#define _val                    TRUE

#define _rise(_x)               ((_x) /\ (-)[~(_x)])

% Here, "i" indicates input and "o" indicates output.
#define _nand(_i1,_i2,_o)       (O(_o)  <==>  ~[(_i1) /\ (_i2)])
#define _nand3(_i1,_i2,_i3,_o)  (O(_o)  <==>  ~[(_i1) /\ (_i2) /\ (_i3)])

#define _circuit_operation      [_nand(d,p_2,p_1)                     \
                                    /\ _nand3(p_1,ck,p_3,p_2)          \
```

```
                                  /\ _nand(ck,p_4,p_3)              \
                                  /\ _nand(p_3,p_1,p_4)             \
                                  /\ _nand(p_2,q,p_5)               \
                                  /\ _nand(p_5,p_3,q)               \
                                  ]


#define _specification \
    [(_rise(ck)  /\  [<-]_{LEQ 2}:(d <==> _val)  /\  O:[]_{< 2}:~_rise(ck)) \
        ==>                                                                  \
        (O^4[(q <==> _val)   A   (_rise(ck)  /\  []_{< 1}:(q <==> _val))])  \
        ]


% To verify:
[]:_circuit_operation  ==>  []:_specification
```

The C preprocessor enables use of the macros here;[2] after such preprocessing, the actual formula $(\Box\varphi_c \Rightarrow \Box\varphi_s)$ is as follows:

```
[]((Op_1  <==>  ~(d /\ p_2))
   /\
   (Op_2  <==>  ~(p_1 /\ ck /\ p_3))
   /\
   (Op_3  <==>  ~(ck /\ p_4))
   /\
   (Op_4  <==>  ~(p_3 /\ p_1))
   /\
   (Op_5  <==>  ~(p_2 /\ q))
   /\
   (Oq  <==>  ~(p_5 /\ p_3))
   )
==>
[](ck /\ (-)~ck /\ ([<-]_{LEQ 2} (d  <==>  T)) /\ O([]_{LEQ 1} ~(ck /\ (-)~ck))
   ==>
   (O^4 ((q  <==>  T)  A  (ck /\ (-)~ck /\ ([]_{LEQ 0} (q  <==>  T)))))
   )
```

As [WiP89] discovered, this formula is not valid. When given the preceding as input,[3] the implementation here produces the following output:

[2]Edward Chang suggested this use of the C preprocessor (in 1993).

[3]As the actual algorithm checks satisfiability, not validity, the implementation actually applies the algorithm to the negation of the given input. If such application determines unsatisfiability, then validity of the originally given input is reported; satisfiability reported by the sub-application is interpreted as falsifiability of the originally given input.

```
FALSIFIABLE:

Leading states:
0.  { ck,  d,  p_1, ~p_2,  p_3, ~p_4, ~p_5, ~q}
1.  {~ck,  d,  p_1, ~p_2,  p_3, ~p_4,  p_5,  q}
2.  { ck,  d,  p_1,  p_2,  p_3, ~p_4,  p_5, ~q}
3.  { ck,     ~p_1, ~p_2,  p_3, ~p_4,  p_5, ~q}
4.  {~ck, ~d,  p_1,  p_2,  p_3,  p_4,  p_5, ~q}
5.  {~ck, ~d,  p_1,  p_2,  p_3, ~p_4,  p_5, ~q}

Repeat:
6.  {~ck, ~d,  p_1,  p_2,  p_3, ~p_4,  p_5, ~q}
```

Thus, one has a trace of the incorrect behavior and can debug it. With such information, [WiP89] determined that too rapid fluctuation of the clock causes such incorrect behavior here, and they discuss better specifications.

I.4.3 Compositional Verification of Timing-Specifications

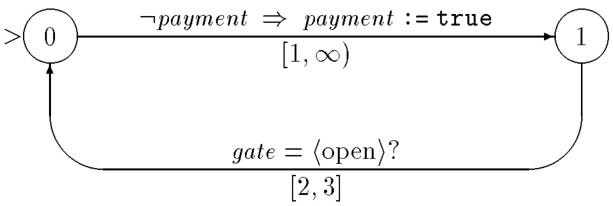
The work of Henzinger, Manna, and Pnueli [HeMP94] presents "timed transition-diagrams" for expressing systems with timing-specifications. An example derived from [HeMP94]'s traffic-light example is as follows.

Here, a system is represented by a collection of automata-like diagrams — plus some variables, for which there may be initial conditions. Each of the diagrams corresponds to a component of the overall system being represented. In each diagram, locations correspond to 'states' of the system-component being represented, and arcs between locations correspond to transitions between these states; the initial location for a component is specified via the annotation "$>$" at the location. Preconditions for a transition are specified via a label (on the relevant arc) comprising a boolean expression delimited at the end by "?" or "$\Rightarrow$"; a transition from a state is 'enabled' when the current location is the state's location and the transition's preconditions' current value is `true`, and otherwise the transition is not enabled. Effects of a transition are specified via assignment-statements following the label-delimiter "$\Rightarrow$": in the state following occurrence of a transition, for each assignment-statement "⟨variable⟩ `:=` ⟨value⟩" labeling the transition's arc, the value of ⟨variable⟩ is ⟨value⟩. A transition's timing-specifications comprise a lower bound l and an upper bound u, which label the transition's arc as "$[l, u]$" (or "$[l, \infty)$" if u is ∞); a transition with lower bound l and upper bound u must be enabled for l units of time before occurring, and it must never be enabled for more than u units of time (without occurring). An upper bound of "!" specifies that the associated transition is required to occur as soon as its lower bound is satisfied. At each location is an idling transition which is not shown with an arc; this transition does not change location, it has no preconditions, it changes no variables, its lower bound is 1, and its upper bound is ∞.

The sample system considered here simulates activity involving traffic at an automated toll-booth. Each time a vehicle's driver pays the toll, the toll-booth's gate (which in reality is typically merely a bar) should be opened (i.e. raised) for an amount of time sufficient to allow passage of the vehicle through the gate, and then the gate should be closed. Here, the traffic is considered to comprise a single process P_{t}; the gate-controller also comprises a single process, P_{c}. Thus, two diagrams are used to represent the overall system. Variables used comprise *gate* and *payment*; initially, $gate = \langle \text{closed} \rangle$.

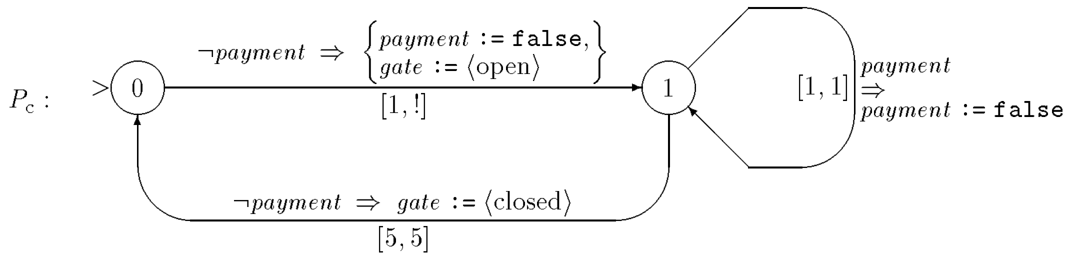
The diagram representing the traffic is as follows:

P_{t} :



The transition from location 0 to 1 represents the arrival of a driver who pays the toll. The transition back represents the passage of the driver through the gate; this transition back is possible only if the gate is open, and between two and three units of time may pass before completion of this transition.

The diagram representing the gate-controller is as follows:



The transition from location 0 to 1 here represents the acceptance of a payment and the concomitant opening of the gate. The transition from location 1 back to location 0 represents the closing of the gate five units of time after the payment was accepted. The transition from location 1 to itself represents acceptance of payments while the gate is already open; after each such acceptance, the gate will be kept open for five units of time.

Translating those diagrams into temporal formulas is somewhat automatic. One issue that arises involves 'frame-axioms', as discussed by McCarthy and Hayes [McCH69] and Kowalski [Kow79]: if a transition does not modify a variable, then its value is preserved. Expressing a transition's preservation of a variable which is private to a module is straightforward: the transition's formula can simply specify that the variable's next value is the same as the current value ("$\bigcirc\langle\text{variable}\rangle \Leftrightarrow \langle\text{variable}\rangle$", for a boolean $\langle$variable$\rangle$). The following macro-file is used to express such preservation:

```
% "pres_macro.tl.h"

#ifndef PRES_MACRO_TL_H
#define PRES_MACRO_TL_H

#define _preserving(_v)         (O[_v] <==> [_v])

#endif
```

Preservation of variables which are modifiable by multiple modules happens only when each module does not modify the variable; special propositions for each such variable and module carry this information, e.g. `pt_preserving_payment` and `pc_preserving_payment` in the following.

Here is the translation of the traffic-process P_t into temporal forms:

```
% "traffic.tl.h"

% Identifier for elements of this module: "pt" (*p*rocess: *t*raffic).

#include "pres_macro.tl.h"
% Publicly writable variable: "payment".
% So public frame-axioms involving this module use "pt_preserving_payment".
```

```
% Variable read but not written here: "gate_is_open".

% Locations: 0 and 1.
% So the location-variables here comprise "pt_at_l0" and "pt_at_l1".


#define _pt_init                        pt_at_l0


% Idling ("I") transition ("tr") --- at any location:
#define _enabled_pt_tr_I                TRUE
#define occurring_pt_tr_I \
    (_enabled_pt_tr_I \
        /\ [_preserving(pt_at_l0) /\ _preserving(pt_at_l1)] \
        /\ pt_preserving_payment)
% lower bound is simply 1:
#define _lower_bound_pt_tr_I            TRUE
% upper bound is simply infinity:
#define _upper_bound_pt_tr_I            TRUE

% transition from location 0 to 1:
#define _enabled_pt_tr_0_1              (pt_at_l0 /\ ~payment)
#define occurring_pt_tr_0_1 \
    (_enabled_pt_tr_0_1 /\ Opt_at_l1 /\ Opayment)
% lower bound is simply 1, upper bound is simply infinity:
#define _lower_bound_pt_tr_0_1          TRUE
#define _upper_bound_pt_tr_0_1          TRUE

% transition from location 1 to 0:
#define _enabled_pt_tr_1_0              (pt_at_l1 /\ gate_is_open)
#define occurring_pt_tr_1_0 \
    (_enabled_pt_tr_1_0 /\ Opt_at_l0 /\ pt_preserving_payment)
#define _lower_bound_pt_tr_1_0 \
    (occurring_pt_tr_1_0  ==>  [<-]_{< 2} _enabled_pt_tr_1_0)
#define _upper_bound_pt_tr_1_0          ~[]_{LEQ 3}:_enabled_pt_tr_1_0


#define _pt_distinct_locations          (pt_at_l0 XOR pt_at_l1)
#define _pt_some_trans \
    (occurring_pt_tr_I \/ occurring_pt_tr_0_1 \/ occurring_pt_tr_1_0)
#define _pt_lower_bounds \
    (_lower_bound_pt_tr_0_1 /\ _lower_bound_pt_tr_1_0)
#define _pt_upper_bounds \
    (_upper_bound_pt_tr_0_1 /\ _upper_bound_pt_tr_1_0)

#define _pt_operation \
    (_pt_distinct_locations /\ _pt_some_trans /\ _pt_lower_bounds \
        /\ _pt_upper_bounds)
```

Next, here is the temporal translation for the gate-controlling process, P_c:

```
% "controller.tl.h"

% Identifier for elements of this module: "pc" (*p*rocess: *c*ontroller).


#include "pres_macro.tl.h"


% Publicly writable variable: "payment".
% So public frame-axioms involving this module use "pc_preserving_payment".

% Privately writable variable: "gate_is_open".

% Locations: 0 and 1.
% So the location-variables here comprise "pc_at_l0" and "pc_at_l1".


#define _gate_is_closed                 ~gate_is_open


#define _pc_init                        (pc_at_l0 /\ _gate_is_closed)


% Idling transition (at any location):
#define _enabled_pc_tr_I                TRUE
#define _occurring_pc_tr_I \
    (_enabled_pc_tr_I \
        /\ [_preserving(pc_at_l0) /\ _preserving(pc_at_l1)]     \
        /\ _preserving(gate_is_open) /\ pc_preserving_payment   \
        )
% lower bound is simply 1:
#define _lower_bound_pc_tr_I            TRUE
% upper bound is simply infinity:
#define _upper_bound_pc_tr_I            TRUE

% transition from location 0 to 1:
#define _enabled_pc_tr_0_1              (pc_at_l0 /\ payment)
#define _occurring_pc_tr_0_1 \
    (_enabled_pc_tr_0_1 /\ Opc_at_l1 /\ O~payment /\ Ogate_is_open)
#define _lower_bound_pc_tr_0_1          TRUE
% upper bound is "!" --- `forcing' action at lower bound (which is 1):
#define _upper_bound_pc_tr_0_1 \
    ([<-]_{< 1}:_enabled_pc_tr_0_1  ==>  _occurring_pc_tr_0_1)

% transition from location 1 to 1:
#define _enabled_pc_tr_1_1              (pc_at_l1 /\ payment)
#define _occurring_pc_tr_1_1 \
    (_enabled_pc_tr_1_1 /\ Opc_at_l1 /\ O~payment /\ _preserving(gate_is_open))
#define _lower_bound_pc_tr_1_1          TRUE
#define _upper_bound_pc_tr_1_1          ~[]_{LEQ 1}:_enabled_pc_tr_1_1
```

```
% transition from location 1 to 0:
#define _enabled_pc_tr_1_0              (pc_at_l1 /\ ~payment)
#define _occurring_pc_tr_1_0 \
    (_enabled_pc_tr_1_0 /\ Opc_at_l0 /\ O:_gate_is_closed \
        /\ pc_preserving_payment)
#define _lower_bound_pc_tr_1_0 \
    (_occurring_pc_tr_1_0  ==>  [<-]_{< 5}:_enabled_pc_tr_1_0)
#define _upper_bound_pc_tr_1_0          ~[]_{LEQ 5}:_enabled_pc_tr_1_0


#define _pc_distinct_locations          (pc_at_l0 XOR pc_at_l1)
#define _pc_some_trans \
    (_occurring_pc_tr_I \/ _occurring_pc_tr_0_1 \/ _occurring_pc_tr_1_1 \
        \/ _occurring_pc_tr_1_0)
#define _pc_lower_bounds \
    (_lower_bound_pc_tr_0_1 /\ _lower_bound_pc_tr_1_1 \
        /\ _lower_bound_pc_tr_1_0)
#define _pc_upper_bounds \
    (_upper_bound_pc_tr_0_1 /\ _upper_bound_pc_tr_1_1 \
        /\ _upper_bound_pc_tr_1_0)


#define _pc_operation \
    (_pc_distinct_locations /\ _pc_some_trans /\ _pc_lower_bounds \
        /\ _pc_upper_bounds)
```

And here is a file expressing composition of P_{t} and P_{c}:

```
% "composition.tl.h"

#include "traffic.tl.h"
#include "controller.tl.h"

% frame-axiom for the one publicly writable variable:
#define _frame_public_var \
    (pt_preserving_payment /\ pc_preserving_payment  ==> _preserving(payment))

#define _system_operation \
    (_pt_operation /\ _pc_operation /\ _frame_public_var)

#define _system_init                    (_pt_init /\ _pc_init)
```

The implementation here can be used to check various properties of the overall system expressed by the preceding. For example, it's naturally desirable that a driver who pays actually progresses through the gate, say within four units of time after paying. This property can be expressed as follows:

```
% "progression.tl.c"

#include "composition.tl.h"

[]:_system_operation
==>
_system_init  ==>  [](pt_at_l1  ==>  <>_{LEQ 4}:pt_at_l0)
```

When given this material as input, the implementation verifies it; the output is simply “VALID”.

Other desirable properties derive from the original source for this example, which was the traffic-light example of [HeMP94]:

$$
\begin{array}{ll}
\text{(A)} & \Box\Big(payment \;\Rightarrow\; \big(\Diamond_{<2}\,[\Box_{\leq 4}\,(gate = \text{open})]\big)\Big)\\
\text{(B)} & \Box\,(payment \Rightarrow [\Diamond_{\leq 5}\,(\neg payment)])\\
\text{(C)} & \Box\big([\boxminus_{\leq 5}(\neg payment)] \Rightarrow (gate = \text{closed})\big)
\end{array}
$$

These properties actually concern only the gate-controlling module; the traffic-module is irrelevant to them. Hence they may be expressed as follows:

```
% "a.tl.c"

#include "controller.tl.h"

[]:_pc_operation
==>
_pc_init  ==>  [](payment  ==>  <>_{< 2}:[]_{LEQ 4}:gate_is_open)


% "b.tl.c"

#include "controller.tl.h"

[]:_pc_operation
==>
_pc_init  ==>  [](payment  ==>  <>_{LEQ 5}:~payment)


% "c.tl.c"

#include "controller.tl.h"

[]:_pc_operation
==>
_pc_init  ==>  []([-]_{LEQ 5}~payment  ==>  _gate_is_closed)
```

Again, the implementation here verifies the (resultant) formulas, producing the output “VALID”.

Verifying such properties of one module is useful because such can then be used to facilitate verification of other modules or the overall system. For example, property (A) here can replace details of the gate-controlling module in verification of the progression-property above:

```
% "a_progression.tl.c"

#include "traffic.tl.h"

[]:_pt_operation
    /\
    [](payment  ==>  <>_{< 2}:[]_{LEQ 4}:gate_is_open)
==>
_pt_init  ==>  [](pt_at_l1  ==>  <>_{LEQ 4}:pt_at_l0)
```

Again, given this input, the implementation here produces the output "`VALID`".

It is perhaps further interesting to purposefully 'munge' the material here and observe the results. For example, changing the "`4`" to "`6`" in `a.tl.c` engenders the following output:

```
FALSIFIABLE:

Leading states:
0.  {~gate_is_open,  payment,  pc_at_l0, ~pc_at_l1, ~pc_preserving_payment}
1.  { gate_is_open, ~payment, ~pc_at_l0,  pc_at_l1,  pc_preserving_payment}
2.  { gate_is_open, ~payment, ~pc_at_l0,  pc_at_l1,  pc_preserving_payment}
3.  { gate_is_open, ~payment, ~pc_at_l0,  pc_at_l1,  pc_preserving_payment}
4.  { gate_is_open, ~payment, ~pc_at_l0,  pc_at_l1,  pc_preserving_payment}
5.  { gate_is_open, ~payment, ~pc_at_l0,  pc_at_l1,  pc_preserving_payment}

Repeat:
6.  {~gate_is_open, ~payment,  pc_at_l0, ~pc_at_l1,  pc_preserving_payment}
```

This output shows processing of the system where *payment* is `true` at some moment yet the gate is not then open for six units of time.

I.4.4 Interpolation-Properties of This Temporal Logic

Maksimova [Mak91] shows that basic temporal logics fail to provide all interpolants. Specifically, [Mak91] exhibits a formula $A\langle P, X\rangle$ (where "P" and "X" are placeholders for expressions) such that the super-formula $[A\langle p,x\rangle \wedge A\langle p,y\rangle \;\Rightarrow\; (x \;\Leftrightarrow\; y)]$ (where "x" and "y" are propositions) is valid, but any basic temporal logic contains no formula $B\langle P\rangle$ such that the super-formula $\big(A\langle p,x\rangle \Rightarrow (x \Leftrightarrow B\langle p\rangle)\big)$ is valid. The formula $A\langle p,x\rangle$ here is as follows:

$$\Box(\neg x \wedge \neg\Box p \;\Rightarrow\; \neg\bigcirc\neg x) \;\wedge\; \Box(\neg\bigcirc\neg x \Rightarrow \bigcirc x) \;\wedge\; \Box(\bigcirc x \;\Rightarrow\; \neg x \wedge \neg\Box p) \;\wedge\; \Box(x \Rightarrow \neg\Box p) \;\wedge\; \Box\Diamond\Box p$$

Interpolation of this formula is difficult because this formula indicates alternation of x through time, and as Wolper [Wo83] discusses (re "$EVEN\langle X\rangle$"), basic temporal logics cannot express such.

But in the temporal logic here, an interpolant $B\langle p\rangle$ for $A\langle p,x\rangle$ is as follows:

$$\big(\Diamond_{\cong 2}(\neg p \;\wedge\; \bigcirc\Box p)\big)$$

Giving the following input to the implementation here confirms this interpolation:

```
% "interp.tl.c"

#define A(_P,_X) \
    ([](~_X /\ ~[]:_P  ==>  ~O~_X)  /\  [](~O~_X ==> O:_X)                    \
        /\  [](O:_X  ==>  ~_X /\ ~[]:_P)  /\  [](_X ==> ~[]:_P)  /\  []<>[]:_P \
        )

A(p,x)   ==>   [x  <==>  <>_{CONG 2}(~p /\ O[]p)]
```

Given this input, the implementation produces the output "**VALID**". (Again, it may be more interesting to see the implementation find (falsifying) models, e.g. if $\breve{B}\langle p\rangle = \big(\neg\Box p \;\wedge\; \bigcirc(\Diamond_{\cong 2}\Box p)\big)$ is used instead of the correct $B\langle p\rangle$.) The implementation can also verify the formula $\big(A\langle p,x\rangle \Rightarrow \Box[x \Leftrightarrow B\langle p\rangle]\big)$.

Chapter I.5

Concluding Remarks for Part I

I.5.1 Advantages of This Method

- The past-operators and metric operators which are handled — efficiently — by the algorithm here facilitate formal verification (and/or debugging) of concurrent and reactive systems by enabling one to express both the workings and the desired properties of systems concisely, conveniently, and clearly.
- With a specification (of, say, a program) and a desired property of it being expressed together as one formula, the state-space here is restricted by the property being checked, i.e. inconsequential state-sketches are elided. As a contrasting example, the program `Mur`φ of Dill, Drexler, Hu, and Yang [DDHY92] does not employ such restriction, so it explores more states.
- Instead of having completely specific states, state-sketches are used here, avoiding specificity by neither satisfying nor falsifying formulas unless required to commit to one or the other. In some sense, state-sketches represent sets of states, as with binary decision-diagrams [Mcm93].
- Formulas are generally maintained as the user specified them[1] — rather than, say, all being translated to some internal forms as in the work of Fisher [Fish92] — so the user can more easily appreciate or use the algorithm's auxiliary information about a satisfying model: While the algorithm's basic output is simply a list of sets of literals delineating a model, associated state-sketches' formulas are available so the user can see the details of how the original formula is satisfied, if desired.

[1] If additional forms such as in Section 0.3 are desired, one need merely determine their `options()`. This easy extensibility is demonstrated with the operator "⧏" on page 75 above.

– This algorithm is obviously parallelizable since sufficiently distant parts of a graph are independent, i.e. different processors could work on different portions of the graph. A particularly useful scheme would be to have one processor not modifying the graph but merely searching portions of it for fulfilling strongly connected components. By contrast, algorithms such as that of Burch, Clarke, McMillan, Dill, and Hwang [BuCMDH90] seem not straightforwardly parallelizable because their main data-structure, a binary decision-diagram, is basically monolithic, not easily treatable as independent components. (But see also the work of Kimura and Clarke [KimC90] — which actually involves converting binary decision-diagrams to graphs(!) — and the work of Zhang, Sokolosky, and Smolka [ZSS94].)

I.5.2 Contributions of This Work

The specifically new contributions of mine in this part of this dissertation comprise:

– efficiently handling all formulas with past-operators.

– the same, for metric temporal operators.

– a constructive proof of correctness of the algorithm here, employing the idea of embedding of a model.

Part II

A Method for Checking Predicate Temporal Formulas

Chapter II.0

Introduction to Part II

"Let us calculate." — Gottfried Leibniz*

This part of this dissertation presents a deduction-calculus for temporal logic.[1]

While model-checking or -exploring ('semantic tableau') methods such as that presented in Part I (or related methods such as that of Plaisted [Pl86]) are effective for temporal formulas which are essentially propositional, such methods have been inapplicable to general predicate temporal formulas. The deduction-calculus of Abadi and Manna [AbM90] extends nonclausal resolution for predicate logic (as presented by Manna and Waldinger [ManWa80]) to handle temporal operators directly, but that deduction-calculus requires a Cut rule which gratuitously introduces new formulas into proofs, and it imposes complicated temporal restrictions on its rules.

An alternative to reasoning directly with temporal operators involves translating temporal formulas into nontemporal first-order predicate logic, using quantifications and comparisons of explicit, "reified" (as discussed by Shoham [Shoh87]) time-parameters[2] to express the temporal operators. Then, one can apply the accumulated technology for reasoning in first-order logic. Such an approach is used in the work of Ohlbach [O88] and Wallen [Wa89] for modal logics other than temporal logic, and obliquely in the work of Frühwirth [Frü94] for a temporal logic whose scope is limited (so its deduction can be automated via logic-programming). But indiscriminately performing that translation can obfuscate otherwise simple modal-logic proofs, in which the modal operators are intuitively

*In Gerhardt·C. (editor): *Die philosophischen Schriften von Gottfried Wilhelm Leibniz*, Volume VII, first part: "Scientia Generalis; Characteristica", 'Borarbeiten zur allgemeinen Charakteristik', Section XIV (written no earlier than 1684), page 200. Wiedmann, Berlin, 1875–1890.

[1]This part of this dissertation is an extended version of the joint work of McGuire, Manna, and Waldinger [McgMW94].

[2]The term "reify" is used elsewhere — e.g. in the work of Aitken, Reichgelt, and Shadbolt [AiRS94] — more broadly, referring to schemes involving handling modal-logic formulas and possible worlds as 'objects' in a first-order logic. Here, the only reified 'objects' are time-parameters (which correspond to possible worlds).

meaningful; proofs requiring the already cumbersome operation of modal induction are particularly susceptible to such disservice. (See also the work of Ohlbach [O93] re efficiency.)

This part of this dissertation presents a deduction-calculus which employs translation like the preceding, but with some finesse. Temporal operators are translated selectively and gradually, and the resulting expressions representing time are simple: since the domain of time here is the natural numbers, temporal operators can be represented by addition (and subtraction).[3] Then, reasoning about terms which represent time is done automatically via unification (as with "t" in the following simple example) plus a decision-procedure for Presburger arithmetic.

An example of such a proof, of the validity of the formula $(\Box p \Rightarrow p)$, is as follows:

	Assertions	Goals	Explanations
0		$0 : (\Box p \;\underline{\Rightarrow}\; p)$	Theorem
1	$0 : \Box^{-} p$		0, split "$\Rightarrow$"
2		$0 : \boxed{p}^{+}$	0, split "$\Rightarrow$"
3	$t : \boxed{p}^{-}$		1, reify "$\Box^{-}$"
4		*true*	2&3, resolution, unifier $\{t := 0\}$

This structure formalizes the goal-directed style of proofs that humans prefer to construct and find easiest to understand.[4]

The deduction-calculus here also provides equality-application, induction for temporal operators, and rules such as rewriting, for convenience. This deduction-calculus subsumes prior ones such as that of [AbM90]; i.e. any proof using that deduction-calculus can also be done here.

[3] Use of addition (and subtraction) certainly was possible previously, but apparently it wasn't done; comparisons (such as "$\leq$") were used.

[4] Re this goal-directed paradigm for (interactive) theorem-proving, see also the work of Constable et al. [Con86].

Chapter II.1

A Deduction-Calculus

II.1.0 Framework: Deductive Tableaux

The framework here, using deductive tableaux, is that of [ManWa93].

Notation and Procedure

The deduction-calculus presented here is designed for establishing the validity of any given formula $\varphi_{\text{checking}}$ which has no free parameters[1]. (It's customary for such a formula $\varphi_{\text{checking}}$ to actually have the form $[0 : \psi_{\text{checking}}]$.)

The form of a proof in the deduction-calculus here is a **deductive tableau**, which is a table with four columns and any positive number of rows. The leftmost column simply contains row-numbers; the middle two columns, titled "Assertions" and "Goals", contain formulas — only one per row; the rightmost column, titled "Explanations", contains texts which explain how rows were derived. The intuitive interpretation of a deductive tableau is that the column of Goals contains the formula being proved valid, followed by reductions of it to more tractable Goals, and the column of Assertions contains formulas which can be assumed true, to be used in the proof. Construction of a proof of the validity of a formula $\varphi_{\text{checking}}$ begins with the following deductive tableau:

	Assertions	Goals	Explanations
0		$\varphi_{\text{checking}}$	Theorem

[1] This restriction is necessary for soundness; without this restriction it would be possible to prove invalid formulas. For example, without this restriction the formula $[0 : p(a) \Rightarrow p(x)]$ would be provable via one application of the Splitting rule and then one application of the Resolution rule. But this formula is not valid because the semantics of the logic here specifies nothing such as some sort of implicit quantification for the free parameter-symbol x; a model can interpret x as one specific object while interpreting a as a different object.

Then, rows are added via rules. The proof succeeds when *true* is generated as a Goal formula. (The deductive tableau on the preceding page presents a complete, successful proof.)

Associated with every subformula in a deductive tableau is a characteristic called **polarity**, as in the work of Murray [Mu78, Mu82], which is determined by the parity of the number of explicit or implicit negations within whose scope the subformula lies. An implication ("$\Rightarrow$") implicitly negates its first argument. Any strict subformula of an equivalence ("$\Leftrightarrow$") is considered to have both even and odd parity of negations. And placement of a formula in the Assertions column implicitly negates the formula.[2] Even parity of negations occasions **positive** polarity, and odd parity occasions **negative** polarity. A superscript of either "$^+$" or "$^-$" (or both) on a formula indicates the formula's polarity (or polarities).

Implicit Automatic Operations

During the process of constructing a proof, quasi-deductive primitive operations which act within rows rather than adding new ones include renaming free or quantified parameter-symbols as desired and automatically simplifying subexpressions.[3] Simplifications are listed in Tables II.1, II.2, II.3, II.4, II.5, and II.6 (on the following four pages).

[2] This situation reflects the 'semantics' of a deductive tableau (see Section II.3.1), involving an implication whose first argument contains the Assertions (while the second argument contains the Goals).

[3] Such an automatic operation enforces compliance with the syntactic restriction that the operator $\otimes$ does not accept $\otimes$-formulas as arguments by rewriting potential argument $\otimes$-formulas as specified in Table II.9 (on page 101).

Table II.1: Simplifications for Boolean Operators

$$
\begin{array}{rcl}
\neg \mathit{true} & \longrightarrow & \mathit{false} \\
\neg \mathit{false} & \longrightarrow & \mathit{true} \\
\neg\neg\psi & \longrightarrow & \psi \\
\neg(\neg\psi_1 \wedge \neg\psi_2 \wedge \ldots \wedge \neg\psi_k) & \longrightarrow & \psi_1 \vee \psi_2 \vee \cdots \vee \psi_k \\
\neg(\neg\psi_1 \vee \neg\psi_2 \vee \cdots \vee \neg\psi_k) & \longrightarrow & \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k \\[4pt]
\ldots \wedge \mathit{true} \wedge \ldots & \longrightarrow & \ldots \wedge \ldots \\
\ldots \wedge \mathit{false} \wedge \ldots & \longrightarrow & \mathit{false} \\
\ldots \wedge \psi \wedge \ldots \wedge \psi \wedge \ldots & \longrightarrow & \ldots \wedge \psi \wedge \ldots \wedge \ldots \\
\ldots \wedge \psi \wedge \ldots \wedge \neg\psi \wedge \ldots & \longrightarrow & \mathit{false} \\
\ldots \wedge \neg\psi \wedge \ldots \wedge \psi \wedge \ldots & \longrightarrow & \mathit{false} \\
\ldots \wedge (\ldots \wedge \ldots) \wedge \ldots & \longrightarrow & \ldots \wedge \ldots \wedge \ldots \wedge \ldots \\[4pt]
\cdots \vee \mathit{true} \vee \cdots & \longrightarrow & \mathit{true} \\
\cdots \vee \mathit{false} \vee \cdots & \longrightarrow & \cdots \vee \cdots \\
\cdots \vee \psi \vee \cdots \vee \psi \vee \cdots & \longrightarrow & \cdots \vee \psi \vee \cdots \vee \cdots \\
\cdots \vee \psi \vee \cdots \vee \neg\psi \vee \cdots & \longrightarrow & \mathit{true} \\
\cdots \vee \neg\psi \vee \cdots \vee \psi \vee \cdots & \longrightarrow & \mathit{true} \\
\cdots \vee (\cdots \vee \cdots) \vee \cdots & \longrightarrow & \cdots \vee \cdots \vee \cdots \vee \cdots \\[4pt]
\psi \Rightarrow \mathit{true} & \longrightarrow & \mathit{true} \\
\psi \Rightarrow \mathit{false} & \longrightarrow & \neg\psi \\
\mathit{true} \Rightarrow \xi & \longrightarrow & \xi \\
\mathit{false} \Rightarrow \xi & \longrightarrow & \mathit{true} \\
\psi \Rightarrow \psi & \longrightarrow & \mathit{true} \\
\psi \Rightarrow \neg\psi & \longrightarrow & \neg\psi \\
\neg\psi \Rightarrow \psi & \longrightarrow & \psi \\
\neg\psi \Rightarrow \neg\xi & \longrightarrow & \xi \Rightarrow \psi \\
\neg(\psi \Rightarrow \neg\xi) & \longrightarrow & \psi \wedge \xi \\[4pt]
\psi \Leftrightarrow \mathit{true} & \longrightarrow & \psi \\
\psi \Leftrightarrow \mathit{false} & \longrightarrow & \neg\psi \\
\psi \Leftrightarrow \psi & \longrightarrow & \mathit{true} \\
\psi \Leftrightarrow \neg\psi & \longrightarrow & \mathit{false} \\
\neg\psi \Leftrightarrow \neg\xi & \longrightarrow & \psi \Leftrightarrow \xi \\
\neg(\psi \Leftrightarrow \neg\xi) & \longrightarrow & \psi \Leftrightarrow \xi \\[4pt]
\cdots \otimes \cdots \otimes \mathit{true} \otimes \cdots \otimes \cdots & \longrightarrow & \neg(\cdots \vee \cdots \vee \cdots \vee \cdots) \\
\cdots \otimes \mathit{false} \otimes \cdots & \longrightarrow & \cdots \otimes \cdots \\
\cdots \otimes \psi \otimes \cdots \otimes \psi \otimes \cdots & \longrightarrow & \neg\psi \wedge (\cdots \otimes \cdots \otimes \cdots) \\
\cdots \otimes \psi \otimes \cdots \otimes \neg\psi \otimes \cdots & \longrightarrow & \neg(\cdots \vee \cdots \vee \cdots)
\end{array}
$$

(Some simplifications with elements simply reversed from the preceding are elided here.)

As such might arise, an application of "$\wedge$", "$\vee$", or "$\otimes$" to one argument reduces to the argument, an application of "$\wedge$" to zero arguments reduces to *true*, and an application of "$\vee$" or "$\otimes$" to zero arguments reduces to *false*.

For each simplification which specifies multiple occurrences of a formula ψ in the expression being simplified, quantified parameter-symbols can be renamed to enable matching. Thus, the formula $\big((\forall x)\,p(x) \wedge (\forall y)\,p(y)\big)$ can be simplified to $(\forall x)\,p(x)$.

Table II.2: Simplification for Quantifiers

$$(\mathcal{Q}\ldots,\chi,\ldots)\psi \quad\longrightarrow\quad (\mathcal{Q}\ldots,\ldots)\psi \qquad \text{when parameter } \chi \text{ is not free in } \psi.$$

(If the quantifier $(\mathcal{Q}\ldots,\chi,\ldots)$ actually has no parameter-symbols other than χ, it is simply deleted.)

Table II.3: Simplifications for Plain Temporal Operators

$$\begin{array}{rcl}
\bigcirc \mathit{true} & \longrightarrow & \mathit{true} \\
\bigcirc \mathit{false} & \longrightarrow & \mathit{false} \\
\Box \mathit{true} & \longrightarrow & \mathit{true} \\
\Box \mathit{false} & \longrightarrow & \mathit{false} \\
\Box\Box \psi & \longrightarrow & \Box \psi \\
\Diamond \mathit{true} & \longrightarrow & \mathit{true} \\
\Diamond \mathit{false} & \longrightarrow & \mathit{false} \\
\Diamond\Diamond \psi & \longrightarrow & \Diamond \psi \\
\mathit{true}\ \mathsf{U}\ \psi & \longrightarrow & \Diamond \psi \\
\mathit{false}\ \mathsf{U}\ \psi & \longrightarrow & \psi \\
\xi\ \mathsf{U}\ \mathit{true} & \longrightarrow & \mathit{true} \\
\xi\ \mathsf{U}\ \mathit{false} & \longrightarrow & \mathit{false} \\
\psi\ \mathsf{U}\ \psi & \longrightarrow & \psi \\
\mathit{true}\ \mathsf{A}\ \psi & \longrightarrow & \mathit{true} \\
\mathit{false}\ \mathsf{A}\ \psi & \longrightarrow & \psi \\
\xi\ \mathsf{A}\ \mathit{true} & \longrightarrow & \mathit{true} \\
\xi\ \mathsf{A}\ \mathit{false} & \longrightarrow & \Box \xi \\
\psi\ \mathsf{A}\ \psi & \longrightarrow & \psi \\
\ominus \mathit{true} & \longrightarrow & \neg \mathit{first} \\
\ominus \mathit{false} & \longrightarrow & \mathit{first} \\
\boxminus \mathit{true} & \longrightarrow & \mathit{true} \\
\boxminus \mathit{false} & \longrightarrow & \mathit{false} \\
\boxminus\boxminus \psi & \longrightarrow & \boxminus \psi \\
\diamondminus \mathit{true} & \longrightarrow & \mathit{true} \\
\diamondminus \mathit{false} & \longrightarrow & \mathit{false} \\
\diamondminus\diamondminus \psi & \longrightarrow & \diamondminus \psi \\
\mathit{true}\ \mathsf{S}\ \psi & \longrightarrow & \diamondminus \psi \\
\mathit{false}\ \mathsf{S}\ \psi & \longrightarrow & \psi \\
\xi\ \mathsf{S}\ \mathit{true} & \longrightarrow & \mathit{true} \\
\xi\ \mathsf{S}\ \mathit{false} & \longrightarrow & \mathit{false} \\
\psi\ \mathsf{S}\ \psi & \longrightarrow & \psi \\
\mathit{true}\ \mathsf{B}\ \psi & \longrightarrow & \mathit{true} \\
\mathit{false}\ \mathsf{B}\ \psi & \longrightarrow & \psi \\
\xi\ \mathsf{B}\ \mathit{true} & \longrightarrow & \mathit{true} \\
\xi\ \mathsf{B}\ \mathit{false} & \longrightarrow & \boxminus \xi \\
\psi\ \mathsf{B}\ \psi & \longrightarrow & \psi
\end{array}$$

Table II.4: Simplifications for Metric Temporal Operators

$\bigcirc^0 \psi$	$\longrightarrow$	ψ
$\ominus^0 \psi$	$\longrightarrow$	ψ
$\odot^0 \tau$	$\longrightarrow$	τ
$\Box_{<0}\, \psi$	$\longrightarrow$	*true*
$\Diamond_{<0}\, \psi$	$\longrightarrow$	*false*
$\xi\ \mathsf{U}_{<0}\ \psi$	$\longrightarrow$	*false*
$\xi\ \mathsf{A}_{<0}\ \psi$	$\longrightarrow$	*true*
$\boxminus_{<0}\, \psi$	$\longrightarrow$	*true*
$\Diamond\hspace{-0.6em}-_{<0}\, \psi$	$\longrightarrow$	*false*
$\xi\ \mathsf{S}_{<0}\ \psi$	$\longrightarrow$	*false*
$\xi\ \mathsf{B}_{<0}\ \psi$	$\longrightarrow$	*true*
$\Box_{\leq 0}\, \psi$	$\longrightarrow$	ψ
$\Diamond_{\leq 0}\, \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{U}_{\leq 0}\ \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{A}_{\leq 0}\ \psi$	$\longrightarrow$	$\psi \vee \xi$
$\boxminus_{\leq 0}\, \psi$	$\longrightarrow$	ψ
$\Diamond\hspace{-0.6em}-_{\leq 0}\, \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{S}_{\leq 0}\ \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{B}_{\leq 0}\ \psi$	$\longrightarrow$	$\psi \vee \xi$
$\Box_{\cong 0}\, \psi$	$\longrightarrow$	ψ
$\Diamond_{\cong 0}\, \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{U}_{\cong 0}\ \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{A}_{\cong 0}\ \psi$	$\longrightarrow$	$\psi \vee \Box\xi$
$\boxminus_{\cong 0}\, \psi$	$\longrightarrow$	ψ
$\Diamond\hspace{-0.6em}-_{\cong 0}\, \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{S}_{\cong 0}\ \psi$	$\longrightarrow$	ψ
$\xi\ \mathsf{B}_{\cong 0}\ \psi$	$\longrightarrow$	$\psi \vee \boxminus\xi$

Any subscript "$_{\cong 1}$" is erased.

Table II.5: Simplifications for Time-Representing Terms

$$\begin{array}{rcl} \ldots + (\ldots + \ldots) + \ldots & \longrightarrow & \ldots + \ldots + \ldots + \ldots \\ \ldots + 0 + \ldots & \longrightarrow & \ldots + \ldots \\ pred(1) & \longrightarrow & 0 \\ pred(\ldots + 1 + \ldots) & \longrightarrow & \ldots + \ldots \end{array}$$

As such might arise, an application of "+" to one argument reduces to the argument, and an application of "+" to zero arguments reduces to 0.

Table II.6: Simplifications for Time-Annotations

$$\begin{array}{rcl} [\theta \,:\, true] & \longrightarrow & true \\ [\theta \,:\, false] & \longrightarrow & false \\ \big[\theta \,:\, [\tilde{\theta} \,:\, \psi]\big] & \longrightarrow & [\tilde{\theta} \,:\, \psi] \end{array}$$

Axioms and Lemmas

The reflexive axiom for equality is $(\forall x)[0 \,:\, \Box(x = x)]$, or simply $(\forall t)(t = t)$ as a time-formula. As desired, this axiom (either form) can be added to a deductive tableau as an Assertion. Similarly usable axioms involving time are as follows:

Table II.7: Axioms for Time-Representing Expressions

$$\begin{array}{l} \neg(1 = 0) \\ (\forall t, \tilde{t}\,)(\tilde{t} + 1 = t + 1 \;\Rightarrow\; \tilde{t} = t) \\ (\forall t)(t \leq 0 \;\Rightarrow\; t = 0) \\ (\forall t, \tilde{t}\,)\big(\tilde{t} \leq t \;\Leftrightarrow\; (\exists d)(\tilde{t} + d = t)\big) \\ (\forall t, \tilde{t}\,)\big(\tilde{t} < t \;\Leftrightarrow\; \tilde{t} \leq t \wedge \neg(\tilde{t} = t)\big) \\ (\forall t)(0 \cong t) \\ (\forall t, \tilde{t}\,)(\tilde{t} \cong t \;\Rightarrow\; \tilde{t} + t \cong t) \end{array}$$

(Other axioms for time which one might consider are subsumed by the simplifications of Table II.5 and associative-commutative unification for "+".)

Table II.8: Axioms for Temporally Invariable Function- and Relation-Symbols

- For each temporally invariable function-symbol γ used with arity k:

$$(\forall x_1, x_2, \ldots, x_k, y)\Big([0 \,:\, \Diamond(\gamma(x_1, \ldots, x_k) = y)] \;\Rightarrow\; [0 \,:\, \Box(\gamma(x_1, \ldots, x_k) = y)]\Big)\,.$$

- For each temporally invariable relation-symbol ρ used with arity k:

$$(\forall x_1, \ldots, x_k)([0 \,:\, \Diamond\rho(x_1, \ldots, x_k)] \Rightarrow [0 \,:\, \Box\rho(x_1, \ldots, x_k)])\,.$$

If the validity being proved is relative to a theory of a domain, then any domain-axiom of the theory can also be added to the deductive tableau as an Assertion.

Previously proved lemmas can be added similarly.

Typically, when an axiom or lemma is loaded into a deductive tableau, its essentially leading universal quantifiers and "$\Box$"s are unobtrusively eliminated and reified (see below), respectively. For example, the axiom $(\forall x)[0 \,:\, \Box(x = x)]$ can be loaded simply as $[t \,:\, x = x]$.

The Scheme for Deduction-Rules

A deduction-**rule** applies to one or two **antecedent** rows which can be anywhere in a deductive tableau; application of a deduction-rule adds one or more **generated** rows to the deductive tableau. A **schema** for such an operation may be presented like:

Assertions	Goals

with antecedent rows specified above the double line and generated rows below. A schema specifying simply substitution of a formula $\widetilde{\varphi}$ for a subformula φ of one antecedent may be presented simply as:

$$\varphi \longrightarrow \widetilde{\varphi}\,.$$

If there are two such schemas $\varphi \longrightarrow \widetilde{\varphi}$ and $\widetilde{\varphi} \longrightarrow \varphi$, they may be presented together as:

$$\varphi \longleftrightarrow \widetilde{\varphi}\,.$$

Each deduction-rule comprises a family of several related schemas.

As indicated in their presentations below, some deduction-rules can be combined with others. Such combination involves one rule's generated rows undergoing application of another rule, without the intermediate (generated) rows being added to the deductive tableau.

Highlighting

A box or underline highlights an expression (or symbol) in a row when the expression (or symbol) is particularly significant to a deduction-rule which is applied to that row; such a significant expression is called a **target** of the rule. For examples: for the deduction-rules of Quantifier-elimination, Reification of a temporal operator, and Equality-application, the target is a quantifier, a temporal operator, or an equation respectively.

II.1.1 Basic Deduction-Rules

The Rewriting Rule

This rule generates a new row from a prior one by replacing a target subformula with an equivalent formula. Rewritings are listed in Tables II.9, II.10, II.11, II.12, and II.13 (on the following four pages).[4]

[4] Compare these rewritings to the "options" of Table I.1 (on pages 22–23).

Table II.9: Rewritings for Boolean Operators and Quantifiers

$$
\begin{array}{rcl}
\ldots \land \varphi \land \psi \land \ldots & \longleftrightarrow & \ldots \land \psi \land \varphi \land \ldots \\
\cdots \lor \varphi \lor \psi \lor \cdots & \longleftrightarrow & \cdots \lor \psi \lor \varphi \lor \cdots \\
\psi \Rightarrow \xi & \longleftrightarrow & \neg\psi \lor \xi \\
\psi \Leftrightarrow \xi & \longleftrightarrow & \xi \Leftrightarrow \psi \\
\psi \Leftrightarrow \xi & \longleftrightarrow & (\psi \Rightarrow \xi) \land (\xi \Rightarrow \psi) \\
\psi \Leftrightarrow \xi & \longleftrightarrow & (\psi \land \xi) \lor (\neg\psi \land \neg\xi) \\
\neg(\psi_1 \land \psi_2 \land \ldots \land \psi_k) & \longleftrightarrow & \neg\psi_1 \lor \neg\psi_2 \lor \cdots \lor \neg\psi_k \\
\neg(\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k) & \longleftrightarrow & \neg\psi_1 \land \neg\psi_2 \land \ldots \land \neg\psi_k \\
\neg(\psi \Rightarrow \xi) & \longleftrightarrow & \psi \land \neg\xi \\
\neg(\psi \Leftrightarrow \xi) & \longleftrightarrow & (\psi \land \neg\xi) \lor (\xi \land \neg\psi) \\
(\forall\, \chi_1, \chi_2, \ldots, \chi_k)\psi & \longleftrightarrow & (\forall\chi_1)(\forall\chi_2)\cdots(\forall\chi_k)\psi \\
(\exists\, \chi_1, \chi_2, \ldots, \chi_k)\psi & \longleftrightarrow & (\exists\chi_1)(\exists\chi_2)\cdots(\exists\chi_k)\psi \\
(\forall\overline{\chi}_1)(\forall\overline{\chi}_2)\psi & \longleftrightarrow & (\forall\overline{\chi}_2)(\forall\overline{\chi}_1)\psi \\
(\exists\overline{\chi}_1)(\exists\overline{\chi}_2)\psi & \longleftrightarrow & (\exists\overline{\chi}_2)(\exists\overline{\chi}_1)\psi \\
\neg(\forall\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})\neg\psi \\
\neg(\exists\overline{\chi})\psi & \longleftrightarrow & (\forall\overline{\chi})\neg\psi \\
(\forall\overline{\chi})(\psi_1 \land \psi_2 \land \ldots \land \psi_k) & \longleftrightarrow & (\forall\overline{\chi})\psi_1 \land (\forall\overline{\chi})\psi_2 \land \ldots \land (\forall\overline{\chi})\psi_k \\
(\exists\overline{\chi})(\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k) & \longleftrightarrow & (\exists\overline{\chi})\psi_1 \lor (\exists\overline{\chi})\psi_2 \lor \cdots \lor (\exists\overline{\chi})\psi_k
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\psi_1 \land \psi_2 \land \ldots \land \psi_k \land (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_\ell)} \\
\longleftrightarrow & (\psi_1 \land \psi_2 \land \ldots \land \psi_k \land \psi_1) \lor (\psi_1 \land \psi_2 \land \ldots \land \psi_k \land \psi_2) \lor \cdots \lor (\psi_1 \land \psi_2 \land \ldots \land \psi_k \land \psi_\ell) \\
\multicolumn{2}{l}{\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k \lor (\psi_1 \land \psi_2 \land \ldots \land \psi_\ell)} \\
\longleftrightarrow & (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k \lor \psi_1) \land (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k \lor \psi_2) \land \ldots \land (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k \lor \psi_\ell)
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\psi_1 \otimes \psi_2 \otimes \psi_3 \otimes \cdots \otimes \psi_k} \\
\longleftrightarrow & (\psi_1 \land \neg\psi_2 \land \neg\psi_3 \land \ldots \land \neg\psi_k) \\
 & \lor\ (\neg\psi_1 \land \psi_2 \land \neg\psi_3 \land \neg\psi_4 \land \ldots \land \neg\psi_k) \\
 & \lor\ (\neg\psi_1 \land \neg\psi_2 \land \psi_3 \land \neg\psi_4 \land \neg\psi_5 \land \ldots \land \neg\psi_k) \\
 & \vdots \\
 & \lor\ (\neg\psi_1 \land \neg\psi_2 \land \ldots \land \neg\psi_{k-2} \land \neg\psi_{k-1} \land \psi_k) \\
\multicolumn{2}{l}{\neg(\psi_1 \otimes \psi_2 \otimes \psi_3 \otimes \cdots \otimes \psi_k)} \\
\longleftrightarrow & \neg(\psi_1 \lor \psi_2 \lor \cdots \lor \psi_k) \\
 & \lor\ \big(\psi_1 \land (\psi_2 \lor \psi_3 \lor \cdots \lor \psi_k)\big) \lor \big(\psi_2 \land (\psi_3 \lor \cdots \lor \psi_k)\big) \lor \cdots \lor (\psi_{k-1} \land \psi_k)
\end{array}
$$

Table II.10: Expansion-Rewritings for Plain Temporal Operators

$$
\begin{array}{rcl}
\Box\psi & \longleftrightarrow & \psi \land \bigcirc\Box\psi \\
\Diamond\psi & \longleftrightarrow & \psi \lor \bigcirc\Diamond\psi \\
(\xi\ \mathsf{U}\ \psi) & \longleftrightarrow & \psi \lor [\xi \land \bigcirc(\xi\ \mathsf{U}\ \psi)] \\
(\xi\ \mathsf{A}\ \psi) & \longleftrightarrow & \psi \lor [\xi \land \bigcirc(\xi\ \mathsf{A}\ \psi)] \\
\boxminus\psi & \longleftrightarrow & \psi \land \neg\ominus\neg\boxminus\psi \\
\diamondminus\psi & \longleftrightarrow & \psi \lor \ominus\diamondminus\psi \\
(\xi\ \mathsf{S}\ \psi) & \longleftrightarrow & \psi \lor [\xi \land \ominus(\xi\ \mathsf{S}\ \psi)] \\
(\xi\ \mathsf{B}\ \psi) & \longleftrightarrow & \psi \lor [\xi \land \neg\ominus\neg(\xi\ \mathsf{B}\ \psi)]
\end{array}
$$

When used from left to right, these Rewritings are called **expansion**s.

Table II.11: Rewritings for Plain Temporal Operators with Boolean Operators and Quantifiers

$$
\begin{array}{rcl}
\neg\bigcirc\psi & \longleftrightarrow & \bigcirc\neg\psi \\
\neg\Box\psi & \longleftrightarrow & \Diamond\neg\psi \\
\neg\Diamond\psi & \longleftrightarrow & \Box\neg\psi \\
\neg(\xi\ \mathsf{U}\ \psi) & \longleftrightarrow & (\neg\psi)\ \mathsf{A}\ (\neg\xi\ \wedge\ \neg\psi) \\
\neg(\xi\ \mathsf{A}\ \psi) & \longleftrightarrow & (\neg\psi)\ \mathsf{U}\ (\neg\xi\ \wedge\ \neg\psi) \\
\neg\ominus\psi & \longleftrightarrow & \ominus\neg\psi\ \vee\ \mathit{first} \\
\neg\boxminus\psi & \longleftrightarrow & \diamondminus\neg\psi \\
\neg\diamondminus\psi & \longleftrightarrow & \boxminus\neg\psi \\
\neg(\xi\ \mathsf{S}\ \psi) & \longleftrightarrow & (\neg\psi)\ \mathsf{B}\ (\neg\xi\ \wedge\ \neg\psi) \\
\neg(\xi\ \mathsf{B}\ \psi) & \longleftrightarrow & (\neg\psi)\ \mathsf{S}\ (\neg\xi\ \wedge\ \neg\psi) \\
\bigcirc(\psi_1\wedge\psi_2\wedge\ldots\wedge\psi_k) & \longleftrightarrow & \bigcirc\psi_1\ \wedge\ \bigcirc\psi_2\ \wedge\ \ldots\ \wedge\ \bigcirc\psi_k \\
\bigcirc(\psi_1\vee\psi_2\vee\cdots\vee\psi_k) & \longleftrightarrow & \bigcirc\psi_1\ \vee\ \bigcirc\psi_2\ \vee\ \cdots\ \vee\ \bigcirc\psi_k \\
\bigcirc(\psi\Rightarrow\xi) & \longleftrightarrow & \bigcirc\psi\ \Rightarrow\ \bigcirc\xi \\
\bigcirc(\psi\Leftrightarrow\xi) & \longleftrightarrow & \bigcirc\psi\ \Leftrightarrow\ \bigcirc\xi \\
\bigcirc(\forall\overline{\chi})\psi & \longleftrightarrow & (\forall\overline{\chi})\bigcirc\psi \\
\bigcirc(\exists\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})\bigcirc\psi \\
\Box(\psi_1\wedge\psi_2\wedge\ldots\wedge\psi_k) & \longleftrightarrow & \Box\psi_1\ \wedge\ \Box\psi_2\ \wedge\ \ldots\ \wedge\ \Box\psi_k \\
\Box(\forall\overline{\chi})\psi & \longleftrightarrow & (\forall\overline{\chi})\Box\psi \\
\Diamond(\psi_1\vee\psi_2\vee\cdots\vee\psi_k) & \longleftrightarrow & \Diamond\psi_1\ \vee\ \Diamond\psi_2\ \vee\ \cdots\ \vee\ \Diamond\psi_k \\
\Diamond(\exists\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})\Diamond\psi \\
\ominus(\psi_1\wedge\psi_2\wedge\ldots\wedge\psi_k) & \longleftrightarrow & \ominus\psi_1\ \wedge\ \ominus\psi_2\ \wedge\ \ldots\ \wedge\ \ominus\psi_k \\
\ominus(\psi_1\vee\psi_2\vee\cdots\vee\psi_k) & \longleftrightarrow & \ominus\psi_1\ \vee\ \ominus\psi_2\ \vee\ \cdots\ \vee\ \ominus\psi_k \\
\ominus(\forall\overline{\chi})\psi & \longleftrightarrow & (\forall\overline{\chi})\ominus\psi \\
\ominus(\exists\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})\ominus\psi \\
\boxminus(\psi_1\wedge\psi_2\wedge\ldots\wedge\psi_k) & \longleftrightarrow & \boxminus\psi_1\ \wedge\ \boxminus\psi_2\ \wedge\ \ldots\ \wedge\ \boxminus\psi_k \\
\boxminus(\forall\overline{\chi})\psi & \longleftrightarrow & (\forall\overline{\chi})\boxminus\psi \\
\diamondminus(\psi_1\vee\psi_2\vee\cdots\vee\psi_k) & \longleftrightarrow & \diamondminus\psi_1\ \vee\ \diamondminus\psi_2\ \vee\ \cdots\ \vee\ \diamondminus\psi_k \\
\diamondminus(\exists\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})\diamondminus\psi
\end{array}
$$

$$
\left.
\begin{array}{rcl}
(\xi_1\wedge\xi_2)\ \mathsf{U}\ \psi & \longleftrightarrow & (\xi_1\ \mathsf{U}\ \psi)\ \wedge\ (\xi_2\ \mathsf{U}\ \psi) \\
\xi\ \mathsf{U}\ (\psi_1\vee\psi_2) & \longleftrightarrow & (\xi\ \mathsf{U}\ \psi_1)\ \vee\ (\xi\ \mathsf{U}\ \psi_2) \\
(\forall\overline{\chi})\xi\ \mathsf{U}\ \varphi & \longleftrightarrow & (\forall\overline{\chi})(\xi\ \mathsf{U}\ \varphi) \\
\varphi\ \mathsf{U}\ (\exists\overline{\chi})\psi & \longleftrightarrow & (\exists\overline{\chi})(\varphi\ \mathsf{U}\ \psi)
\end{array}
\right\}
\begin{array}{l}
\text{same for} \\
\mathsf{A},\ \mathsf{S},\ \text{and}\ \mathsf{B}
\end{array}
$$

In each of the quantification-cases for U (etc.), the formula φ must not contain any free occurrence of any parameter-symbol in the quantified list $\overline{\chi}$.

Table II.12: Rewritings for Metric Temporal Operators

$$
\begin{array}{rcl}
\bigcirc^{\langle\eta+1\rangle}\psi & \longleftrightarrow & \bigcirc(\bigcirc^{\eta}\psi) \\
\bigcirc^{\langle\eta+1\rangle}\psi & \longleftrightarrow & \bigcirc(\bigcirc^{\eta}\psi) \\
\neg(\bigcirc^{\langle\eta+1\rangle}\psi) & \longleftrightarrow & \bigcirc^{\langle\eta+1\rangle}\neg\psi \\
\neg(\ominus^{\langle\eta+1\rangle}\psi) & \longleftrightarrow & \ominus\neg(\ominus^{\eta}\psi) \vee \mathit{first} \\
\odot^{\langle\eta+1\rangle}\tau & \longleftrightarrow & \odot(\odot^{\eta}\tau) \\
\Box_{<\langle\eta+1\rangle}\psi & \longleftrightarrow & \Box_{\leq\eta}\psi \quad \left\langle \begin{array}{l}\text{and similarly for every other}\\ \text{appropriate temporal operator}\end{array}\right. \\
\Box_{\leq\langle\eta+1\rangle}\psi & \longleftrightarrow & \psi \wedge \bigcirc(\Box_{\leq\eta}\psi) \\
\Diamond_{\leq\langle\eta+1\rangle}\psi & \longleftrightarrow & \psi \vee \bigcirc(\Diamond_{\leq\eta}\psi) \\
\xi\ \mathsf{U}_{\leq\langle\eta+1\rangle}\ \psi & \longleftrightarrow & \psi \vee [\xi \wedge \bigcirc(\xi\ \mathsf{U}_{\leq\eta}\ \psi)] \\
\xi\ \mathsf{A}_{\leq\langle\eta+1\rangle}\ \psi & \longleftrightarrow & \psi \vee [\xi \wedge \bigcirc(\xi\ \mathsf{A}_{\leq\eta}\ \psi)] \\
\boxminus_{\leq\langle\eta+1\rangle}\psi & \longleftrightarrow & \psi \wedge \neg\ominus\neg(\boxminus_{\leq\eta}\psi) \\
\diamondminus_{\leq\langle\eta+1\rangle}\psi & \longleftrightarrow & \psi \vee \ominus(\diamondminus_{\leq\eta}\psi) \\
\xi\ \mathsf{S}_{\leq\langle\eta+1\rangle}\ \psi & \longleftrightarrow & \psi \vee [\xi \wedge \ominus(\xi\ \mathsf{S}_{\leq\eta}\ \psi)] \\
\xi\ \mathsf{B}_{\leq\langle\eta+1\rangle}\ \psi & \longleftrightarrow & \psi \vee [\xi \wedge \neg\ominus\neg(\xi\ \mathsf{B}_{\leq\eta}\ \psi)] \\
\Box_{\cong\eta}\psi & \longleftrightarrow & \psi \wedge \big(\bigcirc^{\eta}(\Box_{\cong\eta}\psi)\big) \\
\Diamond_{\cong\eta}\psi & \longleftrightarrow & \psi \vee \big(\bigcirc^{\eta}(\Diamond_{\cong\eta}\psi)\big) \\
\xi\ \mathsf{U}_{\cong\eta}\ \psi & \longleftrightarrow & \psi \vee \big[(\Box_{<\eta}\xi) \wedge \big(\bigcirc^{\eta}(\xi\ \mathsf{U}_{\cong\eta}\ \psi)\big)\big] \\
\xi\ \mathsf{A}_{\cong\eta}\ \psi & \longleftrightarrow & \psi \vee \big[(\Box_{<\eta}\xi) \wedge \big(\bigcirc^{\eta}(\xi\ \mathsf{A}_{\cong\eta}\ \psi)\big)\big] \\
\boxminus_{\cong\eta}\psi & \longleftrightarrow & \psi \wedge \neg\big(\ominus^{\eta}\neg(\boxminus_{\cong\eta}\psi)\big) \\
\diamondminus_{\cong\eta}\psi & \longleftrightarrow & \psi \vee \big(\ominus^{\eta}(\diamondminus_{\cong\eta}\psi)\big) \\
\xi\ \mathsf{S}_{\cong\eta}\ \psi & \longleftrightarrow & \psi \vee \big[(\boxminus_{<\eta}\xi) \wedge \big(\ominus^{\eta}(\xi\ \mathsf{S}_{\cong\eta}\ \psi)\big)\big] \\
\xi\ \mathsf{B}_{\cong\eta}\ \psi & \longleftrightarrow & \psi \vee \big[(\boxminus_{<\eta}\xi) \wedge \neg\big(\ominus^{\eta}\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi)\big)\big] \\
\neg(\Box_{\leq\eta}\psi) & \longleftrightarrow & \Diamond_{\leq\eta}\neg\psi \\
\neg(\Diamond_{\leq\eta}\psi) & \longleftrightarrow & \Box_{\leq\eta}\neg\psi \\
\neg(\xi\ \mathsf{U}_{\leq\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{A}_{\leq\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\xi\ \mathsf{A}_{\leq\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{U}_{\leq\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\boxminus_{\leq\eta}\psi) & \longleftrightarrow & \diamondminus_{\leq\eta}\neg\psi \\
\neg(\diamondminus_{\leq\eta}\psi) & \longleftrightarrow & \boxminus_{\leq\eta}\neg\psi \\
\neg(\xi\ \mathsf{S}_{\leq\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{B}_{\leq\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\xi\ \mathsf{B}_{\leq\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{S}_{\leq\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\Box_{\cong\eta}\psi) & \longleftrightarrow & \Diamond_{\cong\eta}\neg\psi \\
\neg(\Diamond_{\cong\eta}\psi) & \longleftrightarrow & \Box_{\cong\eta}\neg\psi \\
\neg(\xi\ \mathsf{U}_{\cong\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{A}_{\cong\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\xi\ \mathsf{A}_{\cong\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{U}_{\cong\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\boxminus_{\cong\eta}\psi) & \longleftrightarrow & \diamondminus_{\cong\eta}\neg\psi \\
\neg(\diamondminus_{\cong\eta}\psi) & \longleftrightarrow & \boxminus_{\cong\eta}\neg\psi \\
\neg(\xi\ \mathsf{S}_{\cong\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{B}_{\cong\eta}\ (\neg\xi \wedge \neg\psi) \\
\neg(\xi\ \mathsf{B}_{\cong\eta}\ \psi) & \longleftrightarrow & \neg\psi\ \mathsf{S}_{\cong\eta}\ (\neg\xi \wedge \neg\psi)
\end{array}
$$

Table II.13: Rewritings for Time-Annotations

$$
\begin{array}{rcl}
[\theta : (\neg\psi)] & \longleftrightarrow & \neg[\theta : \psi] \\
[\theta : (\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k)] & \longleftrightarrow & [\theta : \psi_1] \wedge [\theta : \psi_2] \wedge \ldots \wedge [\theta : \psi_k] \\
[\theta : (\psi_1 \vee \psi_2 \vee \cdots \vee \psi_k)] & \longleftrightarrow & [\theta : \psi_1] \vee [\theta : \psi_2] \vee \cdots \vee [\theta : \psi_k] \\
[\theta : (\psi \Rightarrow \xi)] & \longleftrightarrow & [\theta : \psi] \Rightarrow [\theta : \xi] \\
[\theta : (\psi \Leftrightarrow \xi)] & \longleftrightarrow & [\theta : \psi] \Leftrightarrow [\theta : \xi] \\
[\theta : (\psi_1 \otimes \psi_2 \otimes \cdots \otimes \psi_k)] & \longleftrightarrow & [\theta : \psi_1] \otimes [\theta : \psi_2] \otimes \cdots \otimes [\theta : \psi_k] \\
[\theta : (\forall \overline{\chi})\psi] & \longleftrightarrow & (\forall \overline{\chi})[\theta : \psi] \\
[\theta : (\exists \overline{\chi})\psi] & \longleftrightarrow & (\exists \overline{\chi})[\theta : \psi]
\end{array}
$$

(In each of the quantification-cases, the time-annotation θ must not contain any free occurrence of any parameter-symbol in the quantified list $\overline{\chi}$.) When used from left to right, these Rewritings are called **time-distribution**s.

Time-distribution can be unobtrusively combined with the normal operations of other deduction-rules, as desired. (For clear examples of such combination, see applications of the Splitting rule such as in the introductory deductive tableau on page 92.)

The Splitting Rule

This rule splits a formula into its component pieces so processing of them will be clearer:

Assertions	Goals
	$\psi \Rightarrow \xi$
ψ	
	ξ

Assertions	Goals
$\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_k$	
ψ_1	
ψ_2	
$\vdots$	
ψ_k	

Assertions	Goals
	$\xi_1 \vee \xi_2 \vee \cdots \vee \xi_k$
	ξ_1
	ξ_2
	$\vdots$
	ξ_k

When an implication $(\psi \Rightarrow \xi)$ is split as shown, if formula ψ is a conjunction $(\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_k)$, then instead of generating ψ as an Assertion, it's possible to generate simply the results of splitting $\psi = (\psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_k)$; and similarly if formula ξ is a disjunction $(\xi_1 \vee \xi_2 \vee \cdots \vee \xi_k)$ or an implication $(\tilde{\psi} \Rightarrow \tilde{\xi})$.

The Duality Rule

Applied to a negated or simply *false* formula which is in one column (Assertions or Goals), this rule adds the formula's inverse to the other column:

Assertions	Goals
	$\neg\varphi$
φ	

Assertions	Goals
$\neg\varphi$	
	φ

Assertions	Goals
false	
	true

(A *false* Goal is inconsequential.)

The Quantifier-Elimination Rule (Skolemization)

A caveat here is that "skolemization" in other deduction-calculi (such as classical Resolution, for example as presented in the work of Genesereth and Nilsson [GeN87]) has somewhat reversed operations since the goal there is refutation while the goal here is validation.

The schema of the quantifier-elimination rule is:

$$(\mathcal{Q}\chi)\varphi \longrightarrow \tilde{\varphi}\,,$$

where the target subformula $(\mathcal{Q}\chi)\varphi$ must not occur within the scope of any "$\Leftrightarrow$", and formula $\tilde{\varphi}$ is constructed from formula φ as specified below. Some terminology here is as follows: A quantifier $(\mathcal{Q}\chi)$ has **universal character**[5] if either its quantifier-symbol $\mathcal{Q}$ is "$\forall$" and its polarity is positive or if its quantifier-symbol is "$\exists$" and its polarity is negative; otherwise, $(\mathcal{Q}\chi)$ has **existential character**. Then:

- If the quantifier $(\mathcal{Q}\chi)$ that is being eliminated has universal character, then it must not lie within the scope of any $\Diamond$, U, A, $\diamondminus$, S, or B.[6] Let $\overline{\chi}$ be a list of the following parameter-symbols: every parameter-symbol which is free in the row-formula to which this rule is being applied, and every parameter-symbol which is bound by a quantifier whose character is existential and whose scope contains the target subformula $(\mathcal{Q}\chi)\varphi$. Also, let "f_{new}" be a function-symbol which does not already appear in the deductive tableau. Then the new formula $\tilde{\varphi}$ is constructed from the original subformula φ by substituting the term "$f_{\text{new}}(\overline{\chi})$" for each occurrence of parameter-symbol χ which was bound by the target quantifier $(\mathcal{Q}\chi)$. For each of the resulting occurrences of the term "$f_{\text{new}}(\overline{\chi})$", each parameter-symbol $\tilde{\chi} \in \overline{\chi}$ must be bound (or free) as it was before the substitution; otherwise, this attempted application of the Quantifier-elimination rule is rejected. Renaming of parameter-symbols enables

[5] [ManWa93] uses the word "force" instead of "character".

[6] Any such temporal operator specifies implicit quantification of time on which the quantifier $(\mathcal{Q}\chi)$ here depends.

satisfaction of this condition. For example, if Quantifier-elimination is applied to the quantifier $(\exists x)$ in the formula $\Big[0 \,:\, (\forall y)\,(\exists x)^{-}\,\big(p(x,y,v) \,\wedge\, (\forall y)\,q(x,y,\tilde{v})\big)\Big]^{-}$, then rather than obtaining $\Big[0 \,:\, (\forall y)\Big(p\big(f(y),y,v\big) \,\wedge\, (\forall y)\,q\big(f(y),y,\tilde{v}\big)\Big)\Big]$ as the result, beforehand renaming of the parameter-symbol y to $\tilde{y}$ should occur (at proper places), yielding $\big[0 \,:\, (\forall y)(\exists x)\big(p(x,y,v) \,\wedge\, (\forall \tilde{y})\,q(x,\tilde{y},\tilde{v})\big)\big]$; then, eliminating $(\exists x)$ yields $\Big[0 \,:\, (\forall y)\Big(p\big(f(y),y,v\big) \,\wedge\, (\forall \tilde{y})\,q\big(f(y),\tilde{y},\tilde{v}\big)\Big)\Big]$.

- If the quantifier $(\mathcal{Q}\chi)$ that is being eliminated has existential character, then the parameter-symbol χ must not be free in the row-formula to which this rule is being applied, and $(\mathcal{Q}\chi)$ must not lie within the scope of any other quantifier $(\widetilde{\mathcal{Q}}\ldots,\chi,\ldots)$ with the same parameter-symbol χ, nor within the scope of any quantifier having universal character, nor inside the scope and first argument of any $\Box$, $\boxminus$, U, A, S, or B. If these conditions are satisfied, then elimination of $(\mathcal{Q}\chi)$ proceeds in this case by simply removing it; i.e. the 'new' formula $\tilde{\varphi}$ is simply identical to the original subformula φ.

II.1.2 The Time-Reification Rule

Time-reification removes a temporal operator from a formula via a process which is equivalent to the two-step operation of [1] translating the temporal operator to 'equivalent' nontemporal (but first-order) forms, and [2] eliminating the quantifiers introduced by step [1]. Subexpressions are not translated or otherwise modified. (The process is slightly different for the operator "$\odot$".)

The Underlying Translation-Scheme

The underlying scheme for translation is crucial. Examples for previous translation-schemes [AbM90, O93] are as follows:

$$\begin{array}{lll}
[0 \,:\, \bigcirc\bigcirc\bigcirc\Box p] & \longrightarrow & (\forall t)[t \geq 3 \;\Rightarrow\; p(t)] \\
[0 \,:\, \bigcirc\bigcirc\bigcirc\Diamond\Box p] & \longrightarrow & (\exists t_1)\big(t_1 \geq 3 \;\wedge\; (\forall t_2)[t_2 \geq t_1 \;\Rightarrow\; p(t_2)]\big) \\
[0 \,:\, \bigcirc\bigcirc\bigcirc(p \,\mathsf{U}\, q)] & \longrightarrow & (\exists t_{\mathrm{f}})\big(t_{\mathrm{f}} \geq 3 \;\wedge\; q(t_{\mathrm{f}}) \;\wedge\; (\forall t_{\mathrm{i}})[(3 \leq t_{\mathrm{i}}) \wedge (t_{\mathrm{i}} < t_{\mathrm{f}}) \;\Rightarrow\; p(t_{\mathrm{i}})]\big)
\end{array}$$

The scheme used here translates differently. For those examples:

$$\begin{array}{lll}
[0 \,:\, \bigcirc\bigcirc\bigcirc\Box p] & \longrightarrow & (\forall d)\,p(3+d) \\
[0 \,:\, \bigcirc\bigcirc\bigcirc\Diamond\Box p] & \longrightarrow & (\exists d_1)(\forall d_2)\,p(3+d_1+d_2) \\
[0 \,:\, \bigcirc\bigcirc\bigcirc(p \,\mathsf{U}\, q)] & \longrightarrow & (\exists d_{\mathrm{f}})\big(q(3+d_{\mathrm{f}}) \;\wedge\; (\forall d_{\mathrm{i}})[d_{\mathrm{i}} < d_{\mathrm{f}} \;\Rightarrow\; p(3+d_{\mathrm{i}})]\big)
\end{array}$$

This translation-scheme used here is specified via two binary meta-functions $TL_{\mathrm{t}}(\!(\,)\!)$ and $TL(\!(\,)\!)$; the first argument of $TL_{\mathrm{t}}(\!(\,)\!)$ or $TL(\!(\,)\!)$ is a term or a formula, respectively, and the second argument of each is a time-representing term. Translation of a given formula $[\theta \,:\, \varphi]$ naturally begins as $TL(\!(\varphi,\theta)\!)$. The translation-schemas are:

- $TL_{\mathrm{t}}(\!(\nu, \theta)\!)$ is $\nu(\theta)$ for a temporal variable ν.
- $TL_{\mathrm{t}}(\!(\chi, \theta)\!)$ is χ for a parameter-symbol χ.
- $TL_{\mathrm{t}}\big(\!\big(\gamma(\tau_1, \ldots \tau_k),\, \theta\big)\!\big)$ is $\gamma\big(TL_{\mathrm{t}}(\!(\tau_1, \theta)\!), \ldots, TL_{\mathrm{t}}(\!(\tau_k, \theta)\!)\big)$ for a function-application $\gamma(\tau_1, \ldots \tau_k)$.
- $TL_{\mathrm{t}}(\!(\odot\tau, \theta)\!)$ is $TL_{\mathrm{t}}(\!(\tau,\, \theta + 1)\!)$ for the application of the operator "$\odot$" to a term τ.
- $TL(\!(\pi, \theta)\!)$ is $\pi(\theta)$ for a temporally variable proposition π, i.e. a relation-symbol used with zero arity.
- $TL(\!(\rho(\tau_1, \ldots, \tau_k),\, \theta)\!)$ is $\rho\big(TL_{\mathrm{t}}(\!(\tau_1, \theta)\!), \ldots, TL_{\mathrm{t}}(\!(\tau_k, \theta)\!)\big)$ for the not temporally variable application of a relation-symbol ρ to a nonzero number (k) of argument-terms.
- $TL(\!(\tau = \sigma,\, \theta)\!)$ is $\big(TL_{\mathrm{t}}(\!(\tau, \theta)\!) = TL_{\mathrm{t}}(\!(\sigma, \theta)\!)\big)$ for an equation $(\tau = \sigma)$.
- $TL(\!(\star(\psi_1, \ldots, \psi_k),\, \theta)\!)$ is $\star\big(TL(\!(\psi_1, \theta)\!), \ldots, TL(\!(\psi_k, \theta)\!)\big)$ for the application of a boolean operator $\star$ to argument-formulas ψ_1, …, and ψ_k.
- $TL\big(\!\big((\mathcal{Q}\overline{\chi})\psi,\, \theta\big)\!\big)$ is $(\mathcal{Q}\tilde{\overline{\chi}})\big(TL(\!(\tilde{\psi}, \theta)\!)\big)$ for a quantification $(\mathcal{Q}\overline{\chi})\psi$, where $\tilde{\overline{\chi}}$ and $\tilde{\psi}$ are obtained from $(\mathcal{Q}\overline{\chi})\psi$ by renaming parameter-symbols so that no parameter-symbol $\tilde{\chi} \in \tilde{\overline{\chi}}$ occurs in θ.
- $TL(\!(\mathit{first},\, \theta)\!)$ is $(\theta = 0)$.
- $TL(\!(\bigcirc\psi,\, \theta)\!)$ is $TL(\!(\psi,\, \theta + 1)\!)$.
- $TL(\!(\Box\psi,\, \theta)\!)$ is $(\forall d)[TL(\!(\psi,\, \theta + d)\!)]$.
- $TL(\!(\Diamond\psi,\, \theta)\!)$ is $(\exists d)[TL(\!(\psi,\, \theta + d)\!)]$.
- $TL\big(\!\big((\xi \,\mathsf{U}\, \psi),\, \theta\big)\!\big)$ is $(\exists d_{\mathrm{f}})\big(TL(\!(\xi,\, \theta + d_{\mathrm{f}})\!) \,\wedge\, (\forall d_{\mathrm{i}})[d_{\mathrm{i}} < d_{\mathrm{f}} \,\Rightarrow\, TL(\!(\psi,\, \theta + d_{\mathrm{i}})\!)]\big)$.
- $TL\big(\!\big((\xi \,\mathsf{A}\, \psi),\, \theta\big)\!\big)$ is
$$\begin{pmatrix} (\exists d_{\mathrm{f}})\big(TL(\!(\xi,\, \theta + d_{\mathrm{f}})\!) \,\wedge\, (\forall d_{\mathrm{i}})[d_{\mathrm{i}} < d_{\mathrm{f}} \,\Rightarrow\, TL(\!(\psi,\, \theta + d_{\mathrm{i}})\!)]\big) \\ \vee \\ (\forall d)[TL(\!(\xi,\, \theta + d)\!)] \end{pmatrix}.$$
- $TL(\!(\ominus\psi,\, \theta)\!)$ is $\big((0 < \theta) \Rightarrow TL(\!(\psi, \mathit{pred}(\theta))\!)\big)$.
- $TL(\!(\boxminus\psi,\, \theta)\!)$ is $(\forall t)\big[(t \le \theta) \,\Rightarrow\, TL(\!(\psi, t)\!)\big]$.
- $TL(\!(\diamondminus\psi,\, \theta)\!)$ is $(\exists t)\big[(t \le \theta) \,\Rightarrow\, TL(\!(\psi, t)\!)\big]$.
- $TL\big(\!\big((\xi \,\mathsf{S}\, \psi),\, \theta\big)\!\big)$ is
$$(\exists t)\Big((t \le \theta) \,\wedge\, TL(\!(\xi, t)\!) \,\wedge\, (\forall t_{\mathrm{i}})\big[(t < t_{\mathrm{i}}) \wedge (t_{\mathrm{i}} \le \theta) \,\Rightarrow\, TL(\!(\psi, t_{\mathrm{i}})\!)\big]\Big).$$
- $TL\big(\!\big((\xi \,\mathsf{B}\, \psi),\, \theta\big)\!\big)$ is
$$\begin{pmatrix} (\exists t)\Big((t \le \theta) \,\wedge\, TL(\!(\xi, t)\!) \,\wedge\, (\forall t_{\mathrm{i}})\big[(t < t_{\mathrm{i}}) \wedge (t_{\mathrm{i}} \le \theta) \,\Rightarrow\, TL(\!(\psi, t_{\mathrm{i}})\!)\big]\Big) \\ \vee \\ (\forall t)\big[(t \le \theta) \,\Rightarrow\, TL(\!(\xi, t)\!)\big] \end{pmatrix}.$$
- $TL(\!([\tilde{\theta} \,:\, \psi], \theta)\!)$ is $TL(\!(\psi, \tilde{\theta})\!)$.

Translation of metric temporal operators resembles the above with appropriate subformulas added at appropriate places; e.g. $(d \le \eta)$ would be added as a conjunct for $\Diamond_{\le\eta}$.

Reification uses variants of those translation-schemas which specify only one step of translation by changing any 'sub-call' "$TL(\!(\varphi, \theta')\!)$" to "$[\theta' : \varphi]$". Thus, for example, the variant $TL_1(\!(\Box\psi,\ \theta)\!)$ is $(\forall d)[\theta + d\ :\ \psi]$.

The Reification-Schemas

Reification of *first*

The schema for reification of the operator *first* is obvious:

$$[\theta\ :\ \mathit{first}] \quad \longrightarrow \quad (\theta = 0)\ .$$

Reification of $\bigcirc$ and $\ominus$

As an illustrative example, translating the operator "$\bigcirc$" in the formula $[t\ :\ \bigcirc p]$ yields $[t+1\ :\ p]$. Generalizing (facilely), the schema for reification of the operator "$\bigcirc$" is:

$$[\theta\ :\ \bigcirc\psi] \quad \longrightarrow \quad [\theta + 1\ :\ \psi]\ .$$

The translation-scheme TL similarly facilely gives the reification-schema for the operator "$\ominus$":

$$[\theta\ :\ \ominus\psi] \quad \longrightarrow \quad (0 < \theta) \wedge [\mathit{pred}(\theta)\ :\ \psi]\ .$$

Reification of $\Box$ and $\Diamond$

As an illustrative example, suppose the following is a row in a deductive tableau:

	Assertions	Goals	Explanations
	$3\ :\ \Box p$		

Step [1] for reification translates the formula $[3 : \Box p]$ to $(\forall d)[3 + d\ :\ p]$. Then, step [2] eliminates the just-introduced quantifier $(\forall d)$. With the given formula being an assertion, it has negative polarity, so the quantifier $(\forall d)$ has existential character. Then applying quantifier-elimination to $(\forall d)[3 + d\ :\ p]$ yields $[3 + d\ :\ p]$. Thus, applying the Reification rule to the given row yields the new row:

	Assertions	Goals	Explanations
	$3 + d\ :\ p$		reified $\Box^-$

Another example, for "$\Box^+$", is as follows:

	Assertions	Goals	Explanations
		$\big[t\ :\ \neg(v = f(x))\big]\ \wedge\ (\exists y)\Big[c\ :\ \boxed{\Box(v = y)}^{+}\Big]$	
		$\big[t\ :\ \neg(v = f(x))\big]\ \wedge\ (\exists y)[c + h(t, x, y)\ :\ v = y]$	reified $\Box^+$; "h" is new

The operator $\Diamond$ is treated analogously, using the quantifier-symbol "$\exists$" instead of "$\forall$".

Considering these examples and the translation-scheme *TL*, the schemas for reification of the operators $\Box$ and $\Diamond$ are:

$$\Big[\theta : (\Box\psi)^{+}\Big] \longrightarrow [\theta + h_{\text{new}}(\overline{x}) : \psi] \qquad \Big[\theta : (\Box\psi)^{-}\Big] \longrightarrow [\theta + d_{\text{new}} : \psi]$$
$$\Big[\theta : (\Diamond\psi)^{-}\Big] \longrightarrow [\theta + h_{\text{new}}(\overline{x}) : \psi] \qquad \Big[\theta : (\Diamond\psi)^{+}\Big] \longrightarrow [\theta + d_{\text{new}} : \psi]$$

Restrictions and other details are due to the quantifier-elimination step: In each case, the target formula must not occur within the scope of an equivalence ("$\Leftrightarrow$"). In the two cases on the left, for "$\Box^{+}$" and "$\Diamond^{-}$", the expression "$\overline{x}$" denotes a list of parameter-symbols as in skolemization; and in the two cases on the right, for "$\Box^{-}$" and "$\Diamond^{+}$", reification is disallowed for any formula $[\theta : (\Box\psi)^{-}]$ or $[\theta : (\Diamond\psi)^{+}]$ that occurs within the scope of any quantifier which has universal character.[7]

Reification of $\boxminus$ and $\diamondminus$

Clearly:

$$\text{both } \Big[\theta : (\boxminus\psi)^{+}\Big] \text{ and } \Big[\theta : (\diamondminus\psi)^{-}\Big] \longrightarrow (h_{\text{new}}(\overline{x}) \le \theta) \wedge \Big[h_{\text{new}}(\overline{x}) : \psi\Big], \quad \text{and}$$
$$\text{both } \Big[\theta : (\boxminus\psi)^{-}\Big] \text{ and } \Big[\theta : (\diamondminus\psi)^{+}\Big] \longrightarrow (t_{\text{new}} \le \theta) \wedge [t_{\text{new}} : \psi]\ ,$$

with details as for $\Box$ and $\Diamond$.

Reification of U, S, A, and B

Again, clearly:

$$\Big[\theta : (\xi \;\mathsf{U}\; \psi)^{-}\Big] \longrightarrow [\theta + h_{\text{new}}(\overline{x}) : \psi] \wedge \Big(\big(d_{\text{new}} < h_{\text{new}}(\overline{x})\big) \Rightarrow [\theta + d_{\text{new}} : \xi]\Big)$$

$$\Big[\theta : (\xi \;\mathsf{U}\; \psi)^{+}\Big] \longrightarrow$$
$$[\theta + d_{\text{new}} : \psi] \wedge \Big(\big(h_{\text{new}}(\overline{x}, d_{\text{new}}) < d_{\text{new}}\big) \Rightarrow [\theta + h_{\text{new}}(\overline{x}, d_{\text{new}}) : \xi]\Big)$$

$$\Big[\theta : (\xi \;\mathsf{S}\; \psi)^{-}\Big] \longrightarrow$$
$$(h_{\text{new}}(\overline{x}) \le \theta) \wedge \big[h_{\text{new}}(\overline{x}) : \psi\big] \wedge \Big(\Big[\big(h_{\text{new}}(\overline{x}) < t_{\text{new}}\big) \wedge \big(t_{\text{new}} \le \theta\big)\Big] \Rightarrow \big[t_{\text{new}} : \xi\big]\Big)$$

$$\Big[\theta : (\xi \;\mathsf{S}\; \psi)^{+}\Big] \longrightarrow$$
$$(t_{\text{n}} \le \theta) \wedge [t_{\text{n}} : \psi] \wedge \Big(\big[\big(t_{\text{n}} < h_{\text{n}}(\overline{x}, t_{\text{n}})\big) \wedge \big(h_{\text{n}}(\overline{x}, t_{\text{n}}) \le \theta\big)\big] \Rightarrow [h_{\text{n}}(\overline{x}, t_{\text{n}}) : \xi]\Big)$$

Details are as usual, with all cases for these operators disallowing targets that occur within the scope of quantifiers which have universal character.

The reification-schemas for A and B should be obvious.

[7] The scopes of 'external' temporal operators are significant for regular instances of quantifier-elimination, but not here because the salient time-annotation here shadows any external time-frame.

Reification of $\odot$

Reifying the operator "$\odot$" simply involving a single step of translation is not possible: the language here provides no way to express $TL_t(\!(\odot\tau,\, \theta)\!)$'s result involving τ and $\theta{+}1$. So some analysis is applied to the situation.

An example of a formula containing the operator "$\odot$" is $[t\,:\,\odot v = a]$. (Assume that the symbol v here is a temporal variable.) Using the reification-schema of "$\bigcirc$" as a model, it seems appropriate for the result of reification here to be $[t+1\,:\,v = a]$. Generally, this result is correct, for it is generally equivalent to the original formula $[t\,:\,\odot v = a]$. But unfortunately, these two formulas are not equivalent if the symbol a is a temporal variable: in this case, the original formula $[t\,:\,\odot v = a]$ may be interpreted as the nontemporal formula $\big(v(t+1) = a(t)\big)$ while the other formula $[t+1\,:\,v = a]$ may be interpreted as the nontemporal formula $\big(v(t+1) = a(t+1)\big)$. In this case, reification is not possible.

Considering the preceding analysis, the reification-schema for the operator "$\odot$" is:

$$\big[\theta\,:\,\varphi\langle\odot\tau_1,\ldots,\odot\tau_n\rangle\big] \quad\longrightarrow\quad \big[\theta+1\,:\,\varphi\langle\tau_1,\ldots,\tau_n\rangle\big]\ ,$$

where the target subformula $\varphi\langle\ldots\rangle$ must not contain any temporally variable symbols outside of the specified terms $\odot\tau_i$ (and no $\odot\tau_i$ here contains any other $\odot\tau_j$ here).

Reification of Metric Temporal Operators

If desired, metric temporal operators can be reified. Schemas for such would naturally resemble the preceding ones, typically with added subformulas such as $(d_{\text{new}} < \eta)$, $(d_{\text{new}} \leq \eta)$, or $(d_{\text{new}} \cong \eta)$, as appropriate.

II.1.3 The Resolution Rule

The schemas for the Resolution rule are:

Assertions	Goals
$\varphi\Big\langle\big[\theta\,:\,\delta^+\big]\Big\rangle$	
$\psi\Big\langle\big[\tilde\theta\,:\,\tilde\delta^-\big]\Big\rangle$	
$\big(\varphi\langle \mathit{true}\rangle\ \vee\ \psi\langle \mathit{false}\rangle\big)\circ\mu$	

Assertions	Goals
	$\varphi\Big\langle\big[\theta\,:\,\delta^+\big]\Big\rangle$
	$\psi\Big\langle\big[\tilde\theta\,:\,\tilde\delta^-\big]\Big\rangle$
	$\big(\varphi\langle \mathit{true}\rangle\ \wedge\ \psi\langle \mathit{false}\rangle\big)\circ\mu$

Assertions	Goals
$\varphi\left\langle[\theta : \delta^{+}]\right\rangle$	
	$\psi\left\langle[\tilde{\theta} : \tilde{\delta}^{-}]\right\rangle$
	$\begin{pmatrix}\neg\varphi\langle \mathit{true}\rangle \\ \wedge \\ \psi\langle \mathit{false}\rangle\end{pmatrix} \circ \mu$

Assertions	Goals
	$\varphi\left\langle[\theta : \delta^{+}]\right\rangle$
$\psi\left\langle[\tilde{\theta} : \tilde{\delta}^{-}]\right\rangle$	
	$\begin{pmatrix}\varphi\langle \mathit{true}\rangle \\ \wedge \\ \neg\psi\langle \mathit{false}\rangle\end{pmatrix} \circ \mu$

In each case: [1] the formulas φ and ψ must not share any free variables; [2] each target subformula $[\theta : \delta]$ or $[\tilde{\theta} : \tilde{\delta}]$ must not contain any occurrence of any variable which is bound by a quantifier outside of the target;[8] and [3] these target subformulas must unify via a most general unifier μ.

The main target expressions here, δ and $\tilde{\delta}$, are not required to be predicate-applications: resolution here is nonclausal, as in [ManWa80]. Also, the specifications for polarities are not strict: a target subformula which has both polarities certainly has either polarity as desired. Additionally, in each case here, more target subformulas can participate in the resolution; if so, the unification must include them. When either antecedent formula φ or ψ contains such multiple targets, only one target needs to satisfy the polarity-requirement ("$^{+}$" in φ, "$^{-}$" in ψ); but regardless, all targets in φ are replaced by *true* and all targets in ψ are replaced by *false*. (For an example of the use of multiple targets, see Section II.2.2's sample deductive tableau's application of resolution to rows 18 and 19, generating row 20 (on page 127); row 18 has multiple targets.) A final particular re the targets here is that when they are time-formulas, they do not have time-annotations.

The target $[\theta : \delta]$ here is highlighted "$\left[\theta : \boxed{\delta}\right]$" rather than "$\boxed{[\theta : \delta]}$", and similarly for each other target. This choice of highlighting reflects [1] the paramountcy of δ in the search for a matching target with $\tilde{\delta}$ (or vice-versa), since the outermost symbol of formula δ must match the outermost symbol of $\tilde{\delta}$ (see the next paragraph re laxness in matching θ and $\tilde{\theta}$); [2] the importance of $\delta/\tilde{\delta}$'s being true or false, which enables resolution to work; and [3] the time-annotation θ's relatively insignificant status as merely a (time-)parameter. For example, in resolving $[t : p]$ with $[0 : p]$, highlighting "$\left[t : \boxed{p}\right]$" and "$\left[0 : \boxed{p}\right]$" clarifies [1] the matching and [2] the basic element here that is either true or false; [3] re "t" and "0" as mere parameters, note that translation here to nontemporal logic would yield $p(t)$ and $p(0)$ (respectively).

As one may desire, associative-commutative unification for "+" — as in the work of Lincoln and Christian [LinC88] and Stickel [St75] — may be employed.

A further option, when resolution fails only because the targets' time-annotations θ and $\tilde{\theta}$ fail

[8] Without this requirement, one could prove the invalid formula $\big((\forall x)(\exists y)[x < y] \Rightarrow (\exists z)(\forall x)[x < z]\big)$ by eliminating "$\exists$"s and resolving "$<$"s.

to unify, is that the equation $(\theta = \tilde{\theta})$ can be added to the desired result of resolution as something additional which needs to be proven: if the desired result would be a Goal formula $\tilde{G}$, the actual result is $\big(\tilde{G} \wedge (\theta = \tilde{\theta})\big)$; for an Assertion formula $\tilde{A}$, the actual result is $\big((\theta = \tilde{\theta}) \Rightarrow \tilde{A}\big)$. For an example of this situation, see the bottom of page 121.

However, additional restrictions apply if the unifier μ involves a replacement $\chi := \tau$ with term τ containing a temporal variable. For then:

1. Each free occurrence of parameter-symbol χ in the formulas φ and ψ must not lie within the scope of any temporal operator or inside any time-annotation or time-formula.
2. All the time-annotations within whose scopes free occurrences of χ lie must be unified (by μ).

These two restrictions ensure consistency of the time at which a temporal variable may be interpreted. Otherwise, such a symbol — contained in term τ, placed in various contexts by the replacement $\chi := \tau$ — could refer to different objects, violating soundness. The following segment of a deductive tableau illustrates obeying these restrictions: (The symbol v is a temporal variable.)

	Assertions	Goals	Explanations
		$0 \,:\, \boxed{q(v)}^{+}$	
	$[\tilde{t} \,:\, p(x)] \Rightarrow \Big[t \,:\, \boxed{q(x)}^{-}\Big]$		
		$0 \,:\, p(v)$	resolution, unifier $\{x := v,\ t := 0,\ \tilde{t} := 0\}$

To see the necessity of these restrictions, consider the formula:

$$(\forall x)\big(p(x) \Rightarrow \bigcirc p(x)\big) \Rightarrow \big(p(v) \Rightarrow \bigcirc p(v)\big)$$

This formula is not valid, being false in a model $\mathcal{M}$ such that the universe $|\mathcal{M}|$ is {OBJECT0, OBJECT1}, the interpretation $\mathcal{M}[\![p,1]\!]$ [9] is {OBJECT0}, the interpretation $(\mathcal{M}@0)[\![v]\!]$ is OBJECT0, and $(\mathcal{M}@1)[\![v]\!]$ is OBJECT1. Nonetheless, one might begin an attempt to prove validity of this

[9] (Recall that the "1" here indicates p's arity.)

formula as follows:

	Assertions	Goals	Explanations
0		$0 : \begin{pmatrix} (\forall x)[p(x) \Rightarrow \bigcirc p(x)] \\ \Rightarrow \\ [p(v) \Rightarrow \bigcirc p(v)] \end{pmatrix}$	Theorem
1	$0 : \boxed{(\forall x)\left[p(x) \Rightarrow \bigcirc p(x)\right]}^{-}$		0, splitting
2	$0 : \boxed{p(v)}^{-}$		0, splitting
3		$0 : \boxed{\bigcirc p(v)}^{+}$	0, splitting
4	$0 : [p(x) \Rightarrow \bigcirc p(x)]$		1, eliminate $\forall^{-}$
5	$[0 : p(x)] \Rightarrow \left[0 : \boxed{\bigcirc p(x)}^{-}\right]$		4, time-distribution

One might attempt to continue as follows, violating restriction 1 here since the parameter-symbol x lies within the scope of the temporal operator $\bigcirc$ in row 5:

6		$0 : \boxed{p(v)}^{+}$	5&3, resolution, unifier $\{x := v\}$
7		*true*	6&2, resolution

Alternatively, one might attempt to continue as follows:

6		$1 : \boxed{p(v)}^{+}$	3, reify $\bigcirc$
7	$[0 : p(x)] \Rightarrow \left[1 : \boxed{p(x)}^{-}\right]$		5, reify $\bigcirc$

The next step violates restriction 2 since the parameter-symbol x lies within the scope of two ununifiable time-annotations, "0" and "1", in row 7:

8		$0 : \boxed{p(v)}^{+}$	7&6, resolution, unifier $\{x := v\}$
9		*true*	8&2, resolution

For intuition about this situation, note that the temporal variable v would be translated into nontemporal predicate logic as a function whose argument is time. The improper resolutions here each involve unifying x with v at time 1 — i.e. $v(1)$ — but somehow in each case obtaining the result $[0 : p(v)]$ — i.e. $p\big(v(0)\big)$, with an occurrence of x yielding $v(0)$ instead of $v(1)$.

II.1.4 The Equality-Application Rule

This rule involves resolution-based reasoning. Hence, like the Resolution rule itself, this rule comprises a multiplicity of schemas because of the alternatives for columns — plus the alternatives for sides of the equation. Only one such schema need be shown (others may be inferred):

Assertions	Goals
	$\varphi\Big\langle[\theta\,:\,\xi\langle\tau\rangle]\Big\rangle$
$\psi\Big\langle\Big[\tilde{\theta}\,:\,(\tilde{\tau}=\sigma)^{-}\Big]\Big\rangle$	
	$\Big(\varphi\Big\langle[\theta\,:\,\xi\langle\sigma\rangle]\Big\rangle \;\wedge\; \neg\psi\langle \mathit{false}\rangle\Big)\circ\mu$

Here, μ must unify term τ with term $\tilde{\tau}$ as well as time-annotation θ with time-annotation $\tilde{\theta}$; further details are as for resolution. The principal replacement "$\tau := \sigma$" is subject to (the applicable parts of) the restrictions concerning temporal variables (broadened to address the fact that τ is potentially more than a parameter-symbol): if τ contains a temporal variable, then it must not lie within the scope of any temporal operator; and if σ contains a temporal variable, then τ must not lie within the scope of any temporal operator or inside any time-annotation. An example of (correct) equality-application is as follows:

	Assertions	Goals	Explanations
		$0\,:\,p\Big(\boxed{f(a+2)}\Big)$	
	$t\,:\,\boxed{f(x+2)=x}^{-}$		
		$0\,:\,p(a)$	equality-application, unifier $\{x := a,\; t := 0\}$

II.1.5 Induction Rules

Induction for a Domain

The logic may refer to a domain, such as the natural numbers or lists, for which principles of induction hold. Then, rules which implement these principles are available. An example, for the natural numbers, is:

Assertions	Goals
	$\theta \,:\, (\forall \chi)\varphi\langle\chi\rangle$
	$\theta \,:\, \Big(\varphi\langle 0\rangle \,\wedge\, \big[\varphi\langle n\rangle \,\Rightarrow\, \varphi\langle n+1\rangle\big]\Big)$

where the initial row must not contain any free parameter-symbols and the symbol n must be a new zero-arity function-symbol. Thus, a goal can be reduced to a base case plus an inductive step.

Induction for Temporal Operators

Induction also applies to temporal operators, as follows:

Assertions	Goals
	$\theta \,:\, \Box\varphi$
	$[\theta \,:\, \varphi] \,\wedge\, \Big(\big[\theta + c_{\text{new}} \,:\, \varphi\big] \,\Rightarrow\, \big[\theta + c_{\text{new}} + 1 \,:\, \varphi\big]\Big)$

Assertions	Goals
$\theta \,:\, \Diamond\varphi$	
$[\theta \,:\, \varphi] \,\vee\, \Big(\big[(\theta + c_{\text{new}}) \,:\, \neg\varphi\big] \,\wedge\, \big[(\theta + c_{\text{new}} + 1) \,:\, \varphi\big]\Big)$	

In each case, the antecedent row must not contain any free parameter-symbols and the symbol c_{new} must be new.

Induction for Time-Formulas

Induction also applies to time-formulas, using the domain-induction schema(s) for the natural numbers but without time-annotations.

II.1.6 Invocation of a Presburger Decision-Procedure for Time-Formulas

If a row contains only a time-formula φ, then a Presburger-like decision-procedure[10] can be applied to φ as follows. Let the sequence of parameter-symbols $\overline{\chi}$ comprise the free parameter-symbols of φ. If φ is in the Goals column, then the actual formula submitted to the decision-procedure is $(\exists\overline{\chi})\varphi$; if φ is in the Assertions column, then the actual formula submitted to the decision-procedure is $(\forall\overline{\chi})\varphi$. The decision-procedure should return *true* if its input is valid, and *false* if its input is not valid.

If the deductive tableau has multiple rows containing time-formulas, they can be submitted to the decision-procedure together as follows. Let n_{tfA} and n_{tfG} be the numbers of Assertions or Goals — respectively — which are time-formulas, let the formulas A_{i_1}, A_{i_2}, $\ldots$, $A_{i_{n_{\text{tfA}}}}$ be the Assertions that are time-formulas, let the formulas G_{j_1}, G_{j_2}, $\ldots$, $G_{j_{n_{\text{tfG}}}}$ be the Goals that are time-formulas, and let $\overline{\chi}_i$ or $\overline{\chi}_j$ (for $i \in \{i_1, i_2, \ldots, i_{n_{\text{tfA}}}\}$ or $j \in \{j_1, j_2, \ldots, j_{n_{\text{tfG}}}\}$) comprise the free variables of A_i or G_j, respectively. Then the formula $\varphi_{\text{tf}*}$ submitted to the decision-procedure is:

$$[(\forall\overline{\chi}_{i_1})A_{i_1} \wedge (\forall\overline{\chi}_{i_2})A_{i_2} \wedge \ldots \wedge (\forall\overline{\chi}_{i_{n_{\text{tfA}}}})A_{i_{n_{\text{tfA}}}}] \;\Rightarrow\; [(\exists\overline{\chi}_{j_1})G_{j_1} \vee (\exists\overline{\chi}_{j_2})G_{j_2} \vee \cdots \vee (\exists\overline{\chi}_{j_{n_{\text{tfG}}}})G_{j_{n_{\text{tfG}}}}]$$

(As such might arise here, a quantifier with no parameter-symbols is elided, an application of "$\wedge$" or "$\vee$" to one argument reduces to the argument, an application of "$\wedge$" to zero arguments reduces to *true*, and an application of "$\vee$" to zero arguments reduces to *false*.) The result of such an application of the decision-procedure is a Goal formula. For an example of this situation, see Section II.2.1 (from the bottom of page 121 to the top of page 122).

The complexity of the formula $\varphi_{\text{tf}*}$ naturally depends on the actual formulas A_i and G_j that are involved. If $\varphi_{\text{tf}*}$ is too complex, the automatic Presburger decision-procedure that one has may not be applicable. Then, further 'manual' work within the deduction-calculus here would be necessary. Page 122 provides an example of such return to manual labor.

[10] Cooper [Coo72] and Shostak [Shos79] provide examples of such Presburger-based decision-algorithms.

Chapter II.2

Examples of Proofs

II.2.1 A Tutorial Example

The formula $\Big(\Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big) \;\Rightarrow\; \Box\big(p \Rightarrow \Diamond q\big)\Big)$ is valid.[1] Proving the validity is practically automatic with the strategy: apply the Splitting and Reification rules whenever possible, then apply the Resolution rule, and invoke the decision-procedure for (arithmetic) time-formulas. For illustrative purposes, the details of the process of constructing the proof are shown here.

[1] This formula is derived from [MₐₙP91]'s Exercise 3.2.p (on page 259 of that work); proving the validity of the formula here comprises essentially one quarter of that exercise.

The proof begins as follows:

	Assertions	Goals	Explanations
0		$0 : \begin{pmatrix} \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big) \\ \Rightarrow \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem

Splitting clarifies matters; here, it conveniently demarcates the given formula's premise and desired conclusion. Implicitly, the Time-distribution rule is combined with the splitting. These operations add (generated) rows to the proof (the deductive tableau), yielding the following: (Marking rows used, as with row 0 here, facilitates later finding unused ones, to use them.)

	Assertions	Goals	Explanations
0$\surd$		$0 : \begin{pmatrix} \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
1	$0 : \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
2		$0 : \Box(p \Rightarrow \Diamond q)$	0, split

The most straightforward rule applicable to the current Goal, in row 2, is Reification. The result is:

	Assertions	Goals	Explanations
0$\surd$		$0 : \begin{pmatrix} \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
1	$0 : \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
2$\surd$		$0 : \Box^{+}(p \Rightarrow \Diamond q)$	0, split
3		$c_1 : p \Rightarrow \Diamond q$	2, reify $\Box^{+}$

Again, for the current Goal (in row 3), splitting clarifies the situation:

	Assertions	Goals	Explanations
$0\surd$		$0 \,:\, \begin{pmatrix} \Box\Diamond((\neg p)\ \mathsf{S}\ q) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
1	$0 \,:\, \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
$2\surd$		$0 \,:\, \Box^{+}(p \Rightarrow \Diamond q)$	0, split
$3\surd$		$c_1 \,:\, p \underline{\Rightarrow} \Diamond q$	2, reify $\Box^{+}$
4	$c_1 \,:\, p$		3, split
5		$c_1 \,:\, \Diamond^{+} q$	3, split

And again, Reification is the first choice among rules applicable to the current Goal:

	Assertions	Goals	Explanations
$0\surd$		$0 \,:\, \begin{pmatrix} \Box\Diamond((\neg p)\ \mathsf{S}\ q) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
1	$0 \,:\, \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
$2\surd$		$0 \,:\, \Box^{+}(p \Rightarrow \Diamond q)$	0, split
$3\surd$		$c_1 \,:\, p \underline{\Rightarrow} \Diamond q$	2, reify $\Box^{+}$
4	$c_1 \,:\, p$		3, split
$5\surd$		$c_1 \,:\, \Diamond^{+} q$	3, split
6		$c_1 + d_2 \,:\, q$	5, reify $\Diamond^{+}$

There are no obviously useful operations for the current Goal, #6. Similarly for Assertion #4. So one would focus attention on the other as-yet unused Assertion, #1. Then, again, reification appears appropriate — three times (for the $\Box$, the $\Diamond$, and the S):

	Assertions	Goals	Explanations
$0\surd$		$0 : \begin{pmatrix} \Box\Diamond\big((\neg p)\ \mathsf{S}\ q\big) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
$1\surd$	$0 : \Box^{-}\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
$2\surd$		$0 : \Box^{+}(p \Rightarrow \Diamond q)$	0, split
$3\surd$		$c_1 : p \underline{\Rightarrow} \Diamond q$	2, reify $\Box^{+}$
4	$c_1 : \boxed{p}^{-}$		3, split
$5\surd$		$c_1 : \Diamond^{+} q$	3, split
6		$c_1 + d_2 : \boxed{q}^{+}$	5, reify $\Diamond^{+}$
$7\surd$	$t_3 : \Diamond^{-}\big((\neg p)\ \mathsf{S}\ q\big)$		1, reify $\Box^{-}$
$8\surd$	$t_3 + h_4(t_3) : (\neg p)\ \mathsf{S}^{-}\ q$		7, reify $\Diamond^{-}$
9	$h_5(t_3) \leq t_3 + h_4(t_3)$		8, reify S^{-}
10	$h_5(t_3) : \boxed{q}^{-}$		8, reify S^{-}
11	$\big(h_5(t_3) < t_6\big) \wedge \big(t_6 \leq t_3 + h_4(t_3)\big)$ $\Rightarrow$ $\Big[t_6 : \neg\, \boxed{p}^{+}\Big]$		8, reify S^{-}

At this point, it seems plausible to resolve either p or q. Closer inspection indicates that resolving p appears more feasible, since the time-annotations for p appear more unifiable than those for q; so:

	Assertions	Goals	Explanations
$0\surd$		$0 : \begin{pmatrix} \Box\Diamond((\neg p)\ \mathsf{S}\ q) \\ \underline{\Rightarrow} \\ \Box(p \Rightarrow \Diamond q) \end{pmatrix}$	Theorem
$1\surd$	$0 : \Box^{-}\Diamond\big((\neg p)\ \mathsf{S}\ q\big)$		0, split
$2\surd$		$0 : \Box^{+}(p \Rightarrow \Diamond q)$	0, split
$3\surd$		$c_1 : p \underline{\Rightarrow} \Diamond q$	2, reify $\Box^{+}$
$4\surd$	$c_1 : \boxed{p}^{-}$		3, split
$5\surd$		$c_1 : \Diamond^{+} q$	3, split
6		$\underline{c_1 + d_2} : q^{+}$	5, reify $\Diamond^{+}$
$7\surd$	$t_3 : \Diamond^{-}\big((\neg p)\ \mathsf{S}\ q\big)$		1, reify $\Box^{-}$
$8\surd$	$t_3 + h_4(t_3) : (\neg p)\ \mathsf{S}^{-}\ q$		7, reify $\Diamond^{-}$
9	$h_5(t_3) \leq t_3 + h_4(t_3)$		8, reify S^{-}
10	$h_5(t_3) : \boxed{q}^{-}$		8, reify S^{-}
$11\surd$	$\big(h_5(t_3) < t_6\big) \wedge \big(t_6 \leq t_3 + h_4(t_3)\big)$ $\Rightarrow$ $\Big[t_6 : \neg\, \boxed{p}^{+}\Big]$		8, reify S^{-}
$12\surd$	$\neg\Big(\big(h_5(t_3) < t_6\big) \wedge \big(t_6 \leq t_3 + h_4(t_3)\big)\Big)$		11&4, resolution, unifier $\{t_6 := c_1\}$
13		$h_5(t_3) < c_1$ $\wedge$ $\big(c_1 \leq t_3 + h_4(t_3)\big)^{+}$	12, duality

(After the resolution, the Duality-rule is applied — semi-automatically.) Next, one could resolve Goal #6, containing q^{+}, with Assertion #10, containing q^{-}; as the targets' time-annotations actually aren't unifiable, the result would be the Goal $\big(h_5(t_3) = c_1 + d_2\big)$, which would be passed with the deductive tableau's other time-formulas — Assertion #9 and Goal #13 — to a Presburger-like decision-procedure, altogether in the form '$\langle$assertions$\rangle \Rightarrow \langle$goals$\rangle$':

$$(\forall t_3)\Big(h_5(t_3) \leq t_3 + h_4(t_3)\Big) \;\Rightarrow\; \begin{pmatrix} (\exists t_3)\Big(\big(h_5(t_3) < c_1\big) \wedge \big(c_1 \leq t_3 + h_4(t_3)\big)\Big) \\ \bigvee \\ (\exists\, t_3, d_2)\big(h_5(t_3) = c_1 + d_2\big) \end{pmatrix}$$

(Quantifier-elimination is 'inverted', for the free variables.) This formula is indeed valid (for the natural numbers), so the decision-procedure should return *true* which would be added to the deductive tableau as a Goal, so these operations would satisfactorily finish the proof.

Alternatively, one may continue the proof 'manually' as follows:

14√	$(\forall t)^{-}\,(\forall d)^{-}\,(t \leq t+d)$		lemma (property of time)
15√	$\boxed{t \leq t+d}^{-}$		14, eliminate $\forall^{-}$ twice
16√		$\boxed{h_5(c_1) < c_1}^{+}$	15&13, resolution, unifier $\{t_3 := c_1,\ t := c_1,\ d := h_4(c_1)\}$
17√	$(\forall t)^{-}\,(\forall \tilde{t})^{-}\big(\tilde{t} < t \lor (\exists d)(t+d = \tilde{t})\big)$		lemma (property of time)
18√	$\boxed{\tilde{t} < t}^{-} \lor (\exists d)(t+d = \tilde{t})$		17, eliminate $\forall^{-}$ twice
19√	$(\exists d)^{-}\big(c_1 + d = h_5(c_1)\big)$		18&16, resolution, unifier $\{\tilde{t} := h_5(c_1),\ t := c_1\}$; duality
20√	$\boxed{\underline{c_1 + c_7} = h_5(c_1)}^{-}$		19, eliminate $\exists^{-}$
21√		$h_5(c_1) : \boxed{q}^{+}$	20&6, equality-application, unifier $\{d_2 := c_7\}$
22		*true*	21&10, resolution, unifier $\{t_3 := c_1\}$

(The provability of the lemmas used here from the axioms on page 98 should be obvious.)

II.2.2 A Demonstration of Relative Power

The work of Abadi [Ab89] indicates that proving the validity of the following formula φ_{p} is difficult, requiring a powerful logic:

$$\Big((\forall x)[\Diamond(v = x)] \;\wedge\; p(v) \;\wedge\; (\forall x)(\forall y)\big[\big(p(x) \;\wedge\; \Diamond[(v = x) \wedge \bigcirc(v = y)]\big) \;\Rightarrow\; p(y)\big]\Big) \;\Rightarrow\; (\forall x)p(x)$$

Here, in addition to v being a temporal variable, the relation-symbol p used with arity 1 is temporally variable. This formula φ_{p} is provable in the deduction-calculus here, as follows.[2]

Notice that the actual 'goal' to be proved valid, $(\forall x)p(x)$, involves no temporal operators. Indeed, throughout the overall formula φ_{p}, p appears within the scope of no temporal operators. Then, *prima facie*, one might consider it possible that purely nontemporal reasoning could achieve the goal; but such is not possible. Intuitively, considering the formula φ_{p}, the truth of $p(x)$ for each value of x depends on the value of the temporal variable v at some time. Consequently, temporal reasoning is required. Indeed, as indicated by the chaining of the premise $\big(\cdots\ (v = x) \wedge \bigcirc(v = y)\ \cdots \Rightarrow p(y)\big)$, induction — specifically, temporal induction[3] — is required: attempts at proofs using reification (instead of induction) fail. Unfortunately, the given formula's only temporal operator with appropriate form and polarity such that induction could apply to it is the first $\Diamond$, in the premise $(\forall x)[\Diamond(v = x)]$, and even if the Induction rule is amended to apply here, the resulting inductive hypothesis is too weak — again, attempts at proofs fail.

So, one might consider applying techniques as in the situation of 'inventor's paradox', when a theorem must be strengthened or generalized to be proven.[4] But the temporal variability of p hinders this generalization. Prefacing the theorem φ_{p} with the operator $\Box$ doesn't work because induction applied to that would require relating truth-values of p at different times, and there are no such preordained relationships here since p is temporally variable. Strengthening further by changing the theorem's main consequent $(\forall x)p(x)$ to $\Box(\forall x)p(x)$ doesn't work, similarly; indeed, this strengthening is 'excessive', for the formula with such a change actually isn't valid: p could be false for all arguments at all times other than $\mathsf{0}$. Changing $(\forall x)p(x)$ to $\Box p(v)$ also yields a formula which isn't valid, but it does express a key intuition: that for each value of v — at some time — p holds for that value; unfortunately, the time here at which v has this value varies, whereas the time at which p is supposed to hold for the value is $\mathsf{0}$. It's difficult to have a formula specify a value for

[2] [Ab89] gives only a sketch of a proof of the formula φ_{p}. That proof-sketch specifies invocation of a 'definition of an auxiliary predicate' q such that q is not temporally variable and q's 'definition' is: $\big[\mathsf{0}\ :\ (\forall x)\big(q(x) \Leftrightarrow p(x)\big)\big]$. With this definition, it's possible to derive the formula $\Box q(v)$, then $(\forall x)q(x)$, and then $(\forall x)p(x)$.

The proof here is (ultimately) more direct.

[3] The domain or type for the symbols v, x, and y is not specified as one for which a law of induction holds. But even if such were specified, the 'chaining' specifies only a temporal sequence, not necessarily one which is inductive for the domain. For example, even if the domain is the natural numbers, the sequence of values of v is not constrained to be an at all ordered progression of them.

[4] More precisely, one uses a lemma which is a strengthened or generalized version of the original theorem.

v at some arbitrary time and also specify the same value for an argument for p at time $\mathsf{0}$: the scope of the temporal operator for v is likely to cover the occurrence of p. Restraining the scope of the temporal operator in an attempt such as having $(\forall x)\big(\Diamond(v = x) \Rightarrow p(x)\big)$ as a substitute main goal is too weak.[5] The solution here is to explicitly constrain the occurrence of p to be at time $\mathsf{0}$ via a time-annotation which overrides or 'shadows' the scope of the temporal operator used for v. This scheme is employed in the following new main consequent: $\Box(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)$.

Changing the original theorem's main consequent $(\forall x)p(x)$ to this new one yields:

$$\begin{array}{l}\Big((\forall x)[\Diamond(v = x)] \bigwedge p(v) \bigwedge (\forall x)(\forall y)\big[\big(p(x) \wedge \Diamond[(v = x) \wedge \bigcirc(v = y)]\big) \Rightarrow p(y)\big]\Big) \\ \quad\Rightarrow \\ \Box(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)\end{array}$$

But the consequent here actually does not need the antecedent's first conjunct. So the lemma that is used here is actually:

$$\begin{array}{l}\Big(p(v) \bigwedge (\forall x)(\forall y)\big[\big(p(x) \wedge \Diamond[(v = x) \wedge \bigcirc(v = y)]\big) \Rightarrow p(y)\big]\Big) \\ \quad\Rightarrow \\ \Box(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)\end{array}$$

The lemma's proof is below; but first, here is how the lemma can be used to prove the originally desired validity of φ_{p}. In the following, let the formula $(\forall x)(\forall y)\big[\big(p(x) \wedge \Diamond[(v = x) \wedge \bigcirc(v = y)]\big) \Rightarrow p(y)\big]$ be abbreviated as ψ.

	Assertions	Goals	Explanations
0		$0 : \begin{pmatrix}\big((\forall x)[\Diamond(v = x)] \wedge p(v) \wedge \psi\big) \\ \underline{\Rightarrow} \\ (\forall x)p(x)\end{pmatrix}$	Theorem
1	$0 : (\forall x)[\Diamond(v = x)] \wedge p(v) \wedge \psi$		0, split
2		$0 : (\forall x)^{+} p(x)$	0, split
3		$0 : \boxed{p(a)}^{+}$	2, eliminate $\forall^{+}$

[5] This formula $(\forall x)\big(\Diamond(v = x) \Rightarrow p(x)\big)$ is actually directly derivable from the original theorem φ_{p}.

4	$0 \;:\; \big(p(v) \wedge \psi\big) \;\Rightarrow\; \Box(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)$		lemma
5	$[0 : p(v) \wedge \psi] \;\Rightarrow\; \big[0 \;:\; \Box^{-}(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)\big]$		4, time-distribution
6	$[0 : p(v) \wedge \psi] \;\Rightarrow\; \big[t \;:\; (\forall x)^{-}\big((v = x) \Rightarrow [0 : p(x)]\big)\big]$		5, reify $\Box^{-}$
7	$[0 : p(v) \wedge \psi] \;\Rightarrow\; \left[t \;:\; \left((v = x) \;\Rightarrow\; \left[0 : \boxed{p(x)}^{-}\right]\right)\right]$		6, eliminate $\forall^{-}$

Next, in this proof, resolution is applied to Goal #3 and Assertion #7. Note that the time-annotation t of the antecedent row 7 is not involved (in unification) because it is 'shadowed' by the inner time-annotation 0.[6]

8		$[0 : p(v) \wedge \psi] \;\wedge\; \left[t \;:\; \boxed{v = a}^{+}\right]$	7&3, resolution, unifier $\{x := a\}$
9	$0 \;:\; (\forall x)^{-} \Diamond (v = x)$		1, split
10	$\boxed{p(v)}^{-}$		1, split
11	$\boxed{\psi}^{-}$		1, split
12	$0 \;:\; \Diamond^{-}(v = x)$		9, eliminate $\forall^{-}$
13	$h(x) \;:\; \boxed{v = x}^{-}$		12, reify $\Diamond^{-}$
14		$0 : \boxed{p(v)}^{+} \wedge \psi$	13&8, resolution, unifier $\{x := a,\ t := h(a)\}$
15		$0 : \boxed{\psi}^{+}$	14&10, resolution
16		*true*	15&11, resolution

[6]Incidentally, the symbol a is not a temporal variable here.

Next, as promised, here is a proof of the validity of the lemma:

	Assertions	Goals	Explanations
0		$0 \,:\, \left(\begin{array}{l} \big(p(v) \wedge \psi\big) \\ \quad \underline{\Rightarrow} \\ \Box(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big) \end{array}\right)$	Theorem
1	$0 \,:\, \boxed{p(v)}^{-}$		0, split
2	$0 \,:\, \overbrace{(\forall x)^{-}(\forall y)^{-} \left[\begin{array}{l} \left(p(x) \;\wedge\; \Diamond\begin{bmatrix} (v = x) \\ \wedge \\ \bigcirc(v = y) \end{bmatrix}\right) \\ \quad \Rightarrow \\ p(y) \end{array}\right]}^{\psi}$		0, split
3		$0 \,:\, \underline{\Box}(\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)$	0, split
4		$0 \,:\, (\forall x)^{+}\big((v = x) \Rightarrow [0 : p(x)]\big) \;\bigwedge \left(\begin{array}{l} \big[c \,:\, (\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)\big] \\ \quad \Rightarrow \\ \big[c+1 \,:\, (\forall x)\big((v = x) \Rightarrow [0 : p(x)]\big)\big] \end{array}\right)$	3, temporal induction
5		$0 \,:\, \left(\boxed{v = a_0}^{-} \Rightarrow \left[0 \,:\, p\left(\boxed{a_0}\right)\right]\right) \;\bigwedge \big([c : \ldots] \Rightarrow [c+1 \,:\, \ldots]\big)$	4, eliminate $\forall^{+}$
6		$0 \,:\, \left((v = a_0) \Rightarrow \left[0 \,:\, \boxed{p(v)}^{+}\right]\right) \;\bigwedge \big([c : \ldots] \Rightarrow [c+1 \,:\, \ldots]\big)$	5&5, equality-application
7		$\big([c : \ldots] \Rightarrow [c+1 \,:\, \ldots]\big)$	6&1, resolution
8	$c \,:\, (\forall x)^{-}\big((v = x) \Rightarrow [0 : p(x)]\big)$		7, split
9		$c+1 \,:\, (\forall x)^{+}\big((v = x) \Rightarrow [0 : p(x)]\big)$	7, split
10		$c+1 \,\underline{:}\, (v = a_{c+1}) \Rightarrow [0 : p(a_{c+1})]$	9, eliminate $\forall^{+}$
11		$[c+1 \,:\, v = a_{c+1}] \;\underline{\Rightarrow}\; [0 : p(a_{c+1})]$	10, distribution; simplification (of ‘shadowed’ annotation)

12	$c+1\ :\ \boxed{v = a_{c+1}}^{-}$		11, split
13		$0\ :\ \boxed{p(a_{c+1})}^{+}$	11, split
14	$0\ :\ \begin{bmatrix} \left(p(x)\ \wedge\ \Diamond\begin{bmatrix}(v=x)\\ \wedge\\ \bigcirc(v=y)\end{bmatrix}\right) \\ \Rightarrow \\ \boxed{p(y)}^{-} \end{bmatrix}$		2, eliminate $\forall^{-}$ twice
15		$0\ :\ \boxed{p(x)}^{+}\ \wedge\ \Diamond\begin{bmatrix}(v=x)\\ \wedge\\ \bigcirc(v=a_{c+1})\end{bmatrix}$	14&13, resolution, unifier $\{y := a_{c+1}\}$
16	$c\ :\ (v=\tilde{x})\ \Rightarrow\ \left[0\ :\ \boxed{p(\tilde{x})}^{-}\right]$		8, eliminate $\forall^{-}$; rename parameter $(x \longrightarrow \tilde{x})$
17		$\left[0\ :\ \Diamond^{+}((v=\tilde{x})\ \wedge\ \bigcirc(v=a_{c+1}))\right]$ $\wedge$ $[c\ :\ v=\tilde{x}]$	16&15, resolution, unifier $\{x := \tilde{x}\}$
18		$\left[\tilde{t}\ :\ \boxed{v=\tilde{x}}^{+}\wedge\ \bigcirc(v=a_{c+1})\right]$ $\wedge$ $\left[c\ :\ \boxed{v=\tilde{x}}^{+}\right]$	17, reify $\Diamond^{+}$
19	$t\ :\ \boxed{x=x}^{-}$		axiom for equality
20		$c\ :\ \underline{\bigcirc}(v=a_{c+1})$	19&18, resolution, unifier $\{x := v,\ \tilde{x} := v$ $t := c,\ \tilde{t} := c\}$
21		$c+1\ :\ \boxed{v=a_{c+1}}^{+}$	20, reify $\bigcirc$
22		*true*	21&12, resolution

II.2.3 Verification of List-Processing

An assumption of list-processing code is that for any list l and at any time, if some temporal variable v is set equal to the list and then (as time passes) v repeatedly gets truncated (while doing so is possible), then eventually v will equal the empty list NIL. Such a statement can be formalized as follows:

$$(\forall l)\Box\Big(\big[(v = l) \;\wedge\; \Box\big(\neg(v = NIL) \;\Rightarrow\; [\odot v = tail(v)]\big)\big] \;\Rightarrow\; \Diamond(v = NIL)\Big).$$

The validity of this formula can be proven in the deduction-calculus here, using domain-induction for lists and reification of "$\odot$", in thirty-three steps. Abbreviating the formula as $(\forall l)\Box\varphi\langle l\rangle$, such a proof begins as follows:

	Assertions	Goals	Explanations
0		$0 \,:\, (\forall l)\Box\varphi\langle l\rangle$	Theorem
1		$0 \,:\, \Big(\underbrace{\Box\varphi\langle NIL\rangle}_{\lambda_1} \;\wedge\; \underbrace{\big[\Box\varphi\langle l_0\rangle \Rightarrow (\forall x)\Box\varphi\langle cons(x, l_0)\rangle\big]}_{\lambda_2}\Big)$	0, domain-induction for lists

With λ_1 and λ_2 as the indicated formulas here, a convenient way to proceed is to prove their validity separately and then load them as Assertions in this main deductive tableau, continuing as follows:

1		$0 \,:\, \Big(\boxed{\lambda_1}^{+} \;\wedge\; \lambda_2\Big)$	0, domain-induction for lists
2	$0 \,:\, \boxed{\lambda_1}^{-}$		lemma
3		$0 \,:\, \boxed{\lambda_2}^{+}$	2&1, resolution
4	$0 \,:\, \boxed{\lambda_2}^{-}$		lemma
5		$true$	4&3, resolution

The proofs of the validity of the lemmas are as follows.

Lemma 1

The first lemma, λ_1, is actually:

$$\Box\Big(\big[(v = NIL) \ \wedge \ \Box\underbrace{\big(\neg(v = NIL) \ \Rightarrow \ [\odot v = tail(v)]\big)}_{\psi}\,\big] \ \Rightarrow \ \Diamond(v = NIL)\Big)$$

Let the subformula $\big(\neg(v = NIL) \ \Rightarrow \ [\odot v = tail(v)]\big)$ be abbreviated as ψ. Then a proof of the validity of λ_1 proceeds as follows:

	Assertions	Goals	Explanations
0		$0 \ : \ \Box^{+}\big([(v = NIL) \ \wedge \ \Box\psi] \ \Rightarrow \ \Diamond(v = NIL)\big)$	Theorem
1		$c_1 \ : \ [(v = NIL) \ \wedge \ \Box\psi] \ \Rightarrow \ \Diamond(v = NIL)$	0, reify $\Box^{+}$
2	$c_1 \ : \ \boxed{(v = NIL)}^{-} \ \wedge \ \Box\psi$		1, split
3		$c_1 \ : \ \Diamond(v = NIL)$	1, split
4		$c_1 \ : \ \boxed{(v = NIL)}^{+} \ \vee \ \bigcirc\Diamond(v = NIL)$	3, expand $\Diamond$
5		*true*	4&2, resolution

Lemma 2

The second lemma, λ_2, is $\big(\Box\varphi\langle l_0\rangle \Rightarrow (\forall x)\Box\varphi\langle cons(x, l_0)\rangle\big)$. A proof of its validity begins as follows:

	Assertions	Goals	Explanations
0		$0 \ : \ \Box\varphi\langle l_0\rangle \Rightarrow (\forall x)\Box\varphi\langle cons(x, l_0)\rangle$	Theorem
1	$0 \ : \ \Box^{-}\varphi\langle l_0\rangle$		0, split
2		$0 \ : \ (\forall x)^{+}\Box\varphi\langle cons(x, l_0)\rangle$	0, split
3		$0 \ : \ \Box^{+}\varphi\langle cons(x, l_0)\rangle$	2, eliminate $\forall^{+}$
4		$c_1 \ : \ \varphi\langle cons(x, l_0)\rangle$	3, reify $\Box^{+}$

Recalling what "$\varphi\langle\ldots\rangle$" abbreviates, this last Goal #4 is actually:

(4) $\qquad c_1 \ : \ \big[[v = cons(a, l_0)] \ \wedge \ \Box\big(\neg(v = NIL) \ \Rightarrow \ [\odot v = tail(v)]\big)\big] \ \Rightarrow \ \Diamond(v = NIL)$

or, continuing to use the abbreviation ψ as in Lemma 1 above:

(4) $\qquad c_1 \ : \ \big([v = cons(a, l_0)] \ \wedge \ \Box\psi\big) \ \Rightarrow \ \Diamond(v = NIL)$

Then the proof may continue as follows:

5	$c_1 \; : \; \boxed{v = cons(a, l_0)}^{-}$		4, split
6	$c_1 \; : \; \underline{\Box}\psi$		4, split
7		$c_1 \; : \; \underline{\Diamond}(v = NIL)$	4, split
8		$[c_1 \; : \; (v = NIL)]$ $\vee \; [c_1 \; : \; \underline{\bigcirc}\Diamond(v = NIL)]$	7, expand $\Diamond$ (and distribution)
9		$[c_1 \; : \; (v = NIL)]$ $\vee \left[c_1 + 1 \; : \; \boxed{\Diamond(v = NIL)}^{+}\right]$	8, reify $\bigcirc$
10	$t_2 \; : \; \begin{pmatrix} [(v = l_0) \wedge \Box\psi] \\ \Rightarrow \\ \boxed{\Diamond(v = NIL)}^{-} \end{pmatrix}$		1, reify $\Box^{-}$ (and unabbreviate "$\varphi\langle\ldots\rangle$")
11		$c_1 + 1 \; : \; (v = l_0) \wedge \boxed{\Box\psi}^{+}$	10&9, resolution, $\{t_2 := (c_1 + 1)\}$
12	$c_1 \; : \; \psi \wedge \bigcirc\Box\psi$		6, expand $\Box$
13	$c_1 \; : \; \overbrace{\begin{pmatrix} \neg\left(\boxed{v} = NIL\right) \\ \Rightarrow \\ \left[\odot v = tail\left(\boxed{v}\right)\right] \end{pmatrix}}^{\psi}$		12, split
14	$c_1 \; : \; \underline{\bigcirc}\Box\psi$		12, split
15	$c_1 + 1 \; : \; \boxed{\Box\psi}^{-}$		14, reify $\bigcirc$
16		$c_1 + 1 \; : \; \boxed{v} = l_0$	15&11, resolution
17	$c_1 \; : \; \begin{pmatrix} \neg[cons(a, l_0) = NIL] \\ \Rightarrow \\ \left[\underline{\odot} v = tail\left(cons(a, l_0)\right)\right] \end{pmatrix}$		13&5, equality-application

18	$c_1+1 \ : \ \left(\begin{array}{l}\neg[cons(a,l_0) = NIL] \\ \Rightarrow \\ \boxed{v = tail\big(cons(a,l_0)\big)}^{-}\end{array}\right)$		17, reify $\odot$
19		$\big[c_1+1 \ : \ \neg\big(cons(a,l_0) = NIL\big)\big]$ $\wedge \ \Big[c_1+1 \ : \ \boxed{tail\big(cons(a,l_0)\big) = l_0}^{+}\Big]$	18&16, equality-application
20	$t \ : \ \boxed{tail\big(cons(x,l)\big) = l}^{-}$		axiom for lists
21		$c_1+1 \ : \ \boxed{\neg\big(cons(a,l_0) = NIL\big)}^{+}$	20&19, resolution, $\{t := (c_1+1),$ $x := a,$ $l := l_0\}$
22	$t \ : \ \boxed{\neg[cons(x,l) = NIL}^{-}$		axiom for lists
23		$true$	22&21, resolution, $\{t := (c_1+1),$ $x := a,$ $l := l_0\}$

Chapter II.3

Properties of the Deduction-Calculus

II.3.1 Soundness

Confirming the soundness of this deduction-calculus is actually trivial: applying the translation $TL(\!(\,)\!)$ of Section II.1.2 to expressions here yields expressions of the nontemporal first-order logic of [ManWa93], and each deductive operation here translates to a deductive operation there — e.g. time-reification here translates to quantifier-elimination there[1] — so the soundness established there implies soundness here.[2] (Analysis of $TL(\!(\,)\!)$ and semantics confirms that this translation preserves validity.) One point re this issue is that the Resolution rule's two restrictions re temporal variables ensure that translation of expressions before application of the rule would agree with their translations afterward.

[1]Remember that such complete translation is spurned in actual operation of the deduction-calculus here because such completely translated formulas would be unwieldy.

[2][ManWa93]'s establishment of soundness involves showing that each deductive operation preserves the condition that the originally given formula $\varphi_{\text{checking}}$ is valid if a certain formula which encapsulates the deductive tableau is valid. The deduction-calculus reports validity when a Goal which is *true* is achieved; but such a Goal happens to make that encapsulating formula obviously valid, which implies validity of $\varphi_{\text{checking}}$ by the condition being preserved; thus, the deduction-calculus is sound, i.e. it reports validity only in the case of actual validity. That encapsulating formula is as follows: If the deductive tableau comprises $n_{\text{A}} + n_{\text{G}}$ Assertions and Goals A_{i_1}, A_{i_2}, …, $A_{i_{n_\text{A}}}$, G_{j_1}, G_{j_2}, …, and $G_{j_{n_\text{G}}}$, and $\overline{\chi}_i$ or $\overline{\chi}_j$ (for $i \in \{i_1, i_2, \ldots, i_{n_\text{A}}\}$ or $j \in \{j_1, j_2, \ldots, j_{n_\text{G}}\}$) comprises all the free variables of A_i or G_j, respectively, then the encapsulating formula is:

$$[(\forall\overline{\chi}_{i_1})A_{i_1} \wedge (\forall\overline{\chi}_{i_2})A_{i_2} \wedge \ldots \wedge (\forall\overline{\chi}_{i_{n_\text{A}}})A_{i_{n_\text{A}}}] \Rightarrow [(\exists\overline{\chi}_{j_1})G_{j_1} \vee (\exists\overline{\chi}_{j_2})G_{j_2} \vee \cdots \vee (\exists\overline{\chi}_{j_{n_\text{G}}})G_{j_{n_\text{G}}}]$$

II.3.2 Relative Completeness

Szalas [SzH88] (see also the work of Szalas and Holenderski [Sz86]) showed that no sound and effective[3] deduction-calculus for full first-order temporal logic can be complete, i.e. none can prove every valid formula. So, re completeness, the best one can hope is that a deduction-calculus is able to prove some large class of formulas relative to something such as the collection of models of a particular temporal-logic specification being considered; or, that the deduction-calculus can prove as much as — or more than — another deduction-calculus. Section II.2.2 suggests that the deduction-calculus here is as powerful as the ones of [Ab89]. Truly proving this relative power is done by showing that each of the deduction-rules of [Ab89] is 'admissible' in the deduction-calculus here, i.e. that given the [Ab89]-rule's premises, the conclusion is derivable here. This task is trivial for most of [Ab89]'s rules, e.g.:

- If $\vdash \varphi$ and $\vdash (\varphi \Rightarrow \psi)$ then $\vdash \psi$.

Then, two final complicated rules re 'predicate-definition' are admitted here exactly as in [Ab89]'s proof 7.1(B). [Ab89]'s 'predicate-definition' involves predicates q which are defined via expressions $(\forall \overline{x})\big(q(\overline{x}) \Leftrightarrow \psi\langle\overline{x}\rangle\big)$.[4] That proof 7.1(B) indicates how instances of $q(\ldots)$ in a deduction can be replaced with $\psi\langle\ldots\rangle$. Naturally, the same can be done here. Thus, the deduction-calculus here is as powerful as that of [Ab89]. Similarly re [AbM90].

II.3.3 Employability to Decide Propositional Formulas

The deduction-calculus here can be used for a decision-procedure for propositional formulas of temporal logic by using expansion-rewritings (and splitting) to mimic the algorithm of Part I of this dissertation. The resulting decision-procedure closely resembles the construction in [AbM90]'s proof of completeness of their deduction-calculus for propositional temporal formulas. See also the work of Cyrluk and Narendran [CyN94] re decidability of fragments of quantifier-free predicate temporal logic.

[3] Processing is not effective if it requires impossible operations such as an infinite amount of work.
[4] A footnote on page 123 (here) gives an example of such.

Chapter II.4

Concluding Remarks for Part II

II.4.1 Advantages of This Method

- Deductive tableaux provide goal-directed proof-construction.
- Partial translation of temporal operators into first-order forms enables application of the various well-developed techniques for reasoning in predicate logic while maintaining the advantages of temporal logic — clarity and conciseness.
- Using addition in the reifying translation facilitates clarity by keeping formulas simple; it also enables application of powerful automatic methods which facilitate deduction such as Presburger decision-algorithms and associative-commutative unification.

II.4.2 Contributions of This Work

The specifically new contributions of mine in this part of this dissertation comprise:

- providing a clear and practical scheme, partial time-reification using addition, for translation of temporal operators into first-order forms.
- developing a thorough deduction-calculus which employs this translation-scheme.

Bibliography

[Ab89] Abadi·M.
"The Power of Temporal Proofs", in *Theoretical Computer Science*, Volume 65 (1989), pages 35–83.
(Cited herein on pages 123 and 133.)

[AbM90] Abadi·M. and Manna·Z.
"Nonclausal Deduction in First-Order Temporal Logic", in *Journal of the Association for Computing Machinery*, Volume 37 (1990), Number 2 (April), pages 279–317.
(Cited herein on pages 2, 7 (in a footnote), 91, 92, 106, and 133.)

[AhHU74] Aho·A., Hopcroft·J., and Ullman·J.
The Design and Analysis of Computer Algorithms, Chapter 5: "Algorithms on Graphs", Section 5: "Strong Connectivity", pages 189–195. Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.
(Cited herein on pages 34 and 73.)

[AiRS94] Aitken·J., Reichgelt·H., and Shadbolt·N.
"Resolution Theorem Proving in Reified Modal Logics", in *Journal of Automated Reasoning*, Volume 12 (1994), Number 1 (February), pages 103–129.
(Cited herein on page 91, in a footnote.)

[BaFG89] Barringer·H., Fisher·M., and Gough·G.
"Fair SMG and Linear Time Model Checking", in Sifakis·J. (editor): *Automatic Verification Methods for Finite State Systems: International Workshop* (1989, Lecture Notes in Computer Science #407). Springer-Verlag, Berlin, 1990.
(Cited herein on page 37, in a footnote.)

[Bee85] Beeson·M.
Foundations of Constructive Mathematics. Springer-Verlag, Berlin, 1985.
(Cited herein on page 18, in a footnote.)

[Ben93] Ben-Ari·M.
Mathematical Logic for Computer Science, Chapter 5: "Temporal Logic", Section 5.4: "Semantic Tableaux", pages 216–228. Prentice-Hall International (UK) Ltd., London, 1993. (See also the work of Ben-Ari, Pnueli, and Manna [BenPM83].)
(Cited herein on pages 17 and 19.)

[BenPM83] Ben-Ari·M., Pnueli·A., and Manna·Z.
"The Temporal Logic of Branching Time", in *Acta Informatica*, Volume 20 (1983), pages 207–226.
(Cited herein on page 135.)

[BuCMD90] Burch · J., Clarke · E., McMillan · K., and Dill · D.
"Sequential Circuit Verification Using Symbolic Model Checking", in *ACM/IEEE Design Automation Conference (27th, Proceedings)* (1990), pages 46–51.
(Cited herein on page 18, in a footnote.)

[BuCMDH90] Burch · J., Clarke · E., McMillan · K., Dill · D., and Hwang · L.
"Symbolic Model Checking: 10^{20} States and Beyond", in *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science* (June 1990), pages 1–13.
(Cited herein on page 88.)

[CiCCM94] Ciapessoni · E., Corsetti · E., Crivelli · E., and Migliorati · M.
"Checking Satisfiability of $\mathrm{TRIO}_{\neq}$ Specifications", in Ohlbach · H. (editor): *Temporal Logic — Proceedings of the ICTL Workshop*, Technical Report #MPI-I-94-230, pages 110–15. Max-Planck-Institut für Informatik, June 1994.
(Cited herein on page 18.)

[ClES86] Clarke · E., Emerson · E., and Sistla · A.
"Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", in *ACM Transactions on Programming Languages and Systems*, Volume 8 (1986), Number 2 (April), pages 244–263.
(Cited herein on page 73.)

[ClGH94] Clarke · E., Grumberg · O., and Hamaguchi · K.
"Another Look at LTL Model Checking", in Dill · D. (editor): *Computer Aided Verification (6th International Conference, CAV '94)* (Lecture Notes in Computer Science #818), pages 415–427. Springer-Verlag, Berlin, 1994.
(Cited herein on page 73.)

[Con86] Constable · R. et al.
Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1986.
(Cited herein on pages 18 (in a footnote) and 92 (in a footnote).)

[Coo72] Cooper · D.
"Theorem proving in arithmetic without multiplication", in *Machine Intelligence*, Volume 7 (1972), pages 91–99.
(Cited herein on page 116, in a footnote.)

[CyN94] Cyrluk · D. and Narendran · P.
"Ground Temporal Logic: A Logic for Hardware Verification", in Dill · D. (editor): *Computer Aided Verification (6th International Conference, CAV '94)* (Lecture Notes in Computer Science #818), pages 247–259. Springer-Verlag, Berlin, 1994.
(Cited herein on page 133.)

[DDHY92] Dill · D., Drexler · A., Hu · A., and Yang · C.
"Protocol Verification as a Hardware Design Aid", in *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–25.
(Cited herein on page 87.)

[EH86] Emerson · E. and Halpern · J.
"'Sometimes' and 'not never' revisited: On branching time versus linear time", in *Journal of the Association for Computing Machinery*, Volume 33 (1986), pages 151–178.
(Cited herein on page 1.)

[FiscL77] Fischer · M. and Ladner · R.
"Propositional Dynamic Logic of Regular Programs", in *Journal of Computer and System Sciences*, Volume 18 (1977), pages 194–211.
(Cited herein on pages 17 and 37 (in a footnote).)

[Fish92] Fisher · M.
"A Model Checker for Linear Time Temporal Logic", in *Formal Aspects of Computing*, Volume 4 (1992), pages 299–319.
(Cited herein on page 87.)

[Fre78] Freuder · E.
"Synthesizing Constraint Expressions", in *Communications of the ACM*, Volume 21 (1978), Number 11 (November), pages 958–966.
(Cited herein on page 18.)

[Frü94] Frühwirth · T.
"Annotated Constraint Logic Programming Applied to Temporal Reasoning", in *Programming Language Implementation and Logic Programming (6th International Symposium, Proceedings)* (Lecture Notes in Computer Science #844), pages 230–243. Springer-Verlag, Berlin, 1994.
(Cited herein on pages 2 and 91 (in a footnote).)

[GeN87] Genesereth · M. and Nilsson · N.
Logical Foundations for Artificial Intelligence. Morgan-Kaufmann, San Mateo, California, 1987.
(Cited herein on page 105.)

[Gl95] Glanz · J.
"Mathematical Logic Flushes out the Bugs in Chip Designs", in *Science*, Volume 267, pages 332–333, 20 January 1995.
(Cited herein on page 1, in a footnote.)

[Gor86] Gordon · M.
"Why higher-order logic is a good formalism for specifying and verifying hardware", in *Formal Aspects of VLSI Design*, pages 153–177. Elsevier Science Publishers, North Holland, 1986.
(Cited herein on pages 75–78.)

[Gou84] Gough · G.
"Decision Procedures for Temporal Logic". Master's thesis, Department of Computer Science, University of Manchester, UK, 1984.
(Cited herein on page 37, in a footnote.)

[He91] Henzinger · T.
"The Temporal Specification and Verification of Real-Time Systems", Report No. STAN-CS-91-1380. Department of Computer Science, Stanford University, Stanford, California, 1991.
(Cited herein on pages 1, 18, and 65.)

[HeMP94] Henzinger · T., Manna · Z., and Pnueli · A.
"Temporal Proof Methodologies for Timed Transition Systems", in *Information and Computation*, Volume 112 (1994), Number 2 (August), pages 273–337.
(Cited herein on pages 79 and 84.)

[HuC89] Hughes · G. and Cresswell · M.
An Introduction to Modal Logic. Methuen & Co. Ltd., London, 1989.
(Cited herein on page 138.)

[KeMMP93] Kesten · Y., Manna · Z., McGuire · H., and Pnueli · A.
"A Decision Algorithm for Full Propositional Temporal Logic", in Courcoubetis · C. (editor): *Computer Aided Verification (5th International Conference, CAV '93)* (Lecture Notes in Computer Science #697), pages 97–109. Springer-Verlag, Berlin, 1993.
(Cited herein on page 17.)

[KeP91] Kesten · Y. and Pnueli · A.
"An Efficient Introduction of the Past", Technical Report — Preliminary Version. Department of Computer Science, the Weizmann Institute of Science, Israel, 1991.
(Cited herein on page 18.)

[KifS92] Kifer · M. and Subrahmanian · V.
"Theory of Generalized Annotated Logic Programming and its Applications", in *Journal of Logic Programming*, Volume 12 (1992), Number 4 (April), pages 335–367.
(Cited herein on page 2.)

[KimC90] Kimura · S. and Clarke · E.
"A Parallel Algorithm for Constructing Binary Decision Diagrams", in *1990 IEEE International Conference on Computer Design (Proceedings)*, pages 220–23. IEEE Computer Society Press, 1990.
(Cited herein on page 88.)

[Kow79] Kowalski · R.
Logic for Problem Solving. North Holland, New York, 1979.
(Cited herein on page 80.)

[Koy90] Koymans · R.
"Specifying Real-Time Properties with Metric Temporal Logic", in *Journal of Real-Time Systems*, Volume 2, Number 4, pages 255–299. Kluwer Academic Publishers, 1990.
(Cited herein on pages 1 and 18.)

[Koy92] Koymans · R.
Specifying Message Passing and Time-Critical Systems with Temporal Logic (Lecture Notes in Computer Science #651). Springer-Verlag, Berlin, 1992.
(Cited herein on pages 1 and 18.)

[Kr63] Kripke · S.
"Semantical Analysis of Modal Logic I, Normal Propositional Calculi", *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, Volume 9 (1963), pages 67–96. VEB Deutscher Verlag der Wissenschaften, Berlin. (For a more accessible source, see the work of Hughes and Cresswell [HuC89].)
(Cited herein on page 17.)

[Ku92] Kumar · V.
"Algorithms for Constraint-Satisfaction Problems: A Survey", in *AI Magazine*, Volume 13 (1992), Number 1 (Spring), pages 32–44.
(Cited herein on page 18.)

[LicPZ85] Lichtenstein · O., Pnueli · A., and Zuck · L.
"The Glory of the Past", in *Logics of Programs (Proceedings)* (Lecture Notes in Computer Science #193), pages 196–218. Springer-Verlag, Berlin, 1985.
(Cited herein on pages 1, 17, 18, and 65.)

[LinC88] Lincoln · P. and Christian · J.
"Adventures in Associative-Commutative Unification (A Summary)", in Lusk · E. and Overbeek · R. (editors): *9th International Conference on Automated Deduction* (Lecture Notes in Computer Science #310), pages 358–366. Springer-Verlag, Berlin, 1988.
(Cited herein on page 111.)

[Mac77] Mackworth · A.
"Consistency in Networks of Relations", in *Artificial Intelligence*, Volume 8 (1977), pages 99–118.
(Cited herein on page 18.)

[Mak91] Maksimova · L.
"Temporal Logics with 'The Next' Operator Do Not Have Interpolation or the Beth Property", in *Siberian Mathematics Journal*, Volume 32 (1991), Number 6 (November/December), pages 989–993.
(Cited herein on page 86.)

[ManP91] Manna · Z. and Pnueli · A.
The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, 1991.
(Cited herein on pages 1, 3, and 117 (in a footnote).)

[ManWa80] Manna · Z. and Waldinger · R.
"A Deductive Approach to Program Synthesis", in *ACM Transactions on Programming Languages and Systems*, Volume 2 (1980), pages 90–121.
(Cited herein on pages 91 and 111.)

[ManWa92] Manna · Z. and Waldinger · R.
"Fundamentals of Deductive Program Synthesis", in *IEEE Transactions on Software Engineering*, Volume 18 (1992), Number 8 (August), pages 674–704.
(Cited herein on page 18, in a footnote.)

[ManWa93] Manna · Z. and Waldinger · R.
The Deductive Foundations of Computer Programming. Addison-Wesley, Reading, Massachusetts, 1993.
(Cited herein on pages 93, 105 (in a footnote), and 132.)

[ManWo84] Manna · Z. and Wolper · P.
"Synthesis of Communicating Processes from Temporal Logic Specifications", in *ACM Transactions on Programming Languages and Systems*, Volume 6 (1984), Number 1 (January), pages 68–93.
(Cited herein on page 17.)

[Mar79] Martin-Löf · P.
"Constructive Mathematics and Computer Programming", in Cohen · L. (editor): *International Congress of Logic, Methodology, and Philosophy of Science VI (Proceedings)* (1979). PWN — Polish Scientific Publishers, New York, 1982.
(Cited herein on page 18, in a footnote.)

[MccH69] McCarthy · J. and Hayes · P.
"Some Philosophical Problems from the Standpoint of Artificial Intelligence", in *Machine Intelligence*, Volume 4 (1969), pages 410–17.
(Cited herein on page 80.)

[McgMW94] McGuire · H., Manna · Z., and Waldinger · R.
"Annotation-Based Deduction in Temporal Logic", in Gabbay · D. and Ohlbach · H. (editors): *Temporal Logic (First International Conference, ICTL, Proceedings)* (Lecture Notes in Artificial Intelligence #827), pages 430–444. Springer-Verlag, Berlin, 1994.
(Cited herein on pages 1 and 91 (in a footnote).)

[Mcm93] McMillan · K.
Symbolic Model Checking. Kluwer Academic Publishers, Boston, Massachusetts, 1993.
(Cited herein on pages 17 (in a footnote) and 87.)

[Mi92] Mints · G.
A Short Introduction to Modal Logic (CSLI Lecture Notes No. 30), Section 3.4: "Indexed Formulas and Deduction Rules", pages 44ff. Center for the Study of Language and Information (CSLI), Leland Stanford Junior University, Stanford, California, 1992.
(Cited herein on page 5, in a footnote.)

[Mo74] Montanari · U.
"Networks of Constraints: Fundamental Properties and Applications to Picture Processing", in *Information Sciences*, Volume 7 (1974), pages 95–132.
(Cited herein on page 18.)

[Mu78] Murray · N.
"A Proof Procedure for Non-Clausal First-Order Logic" (technical report). Syracuse University, Syracuse, New York, 1978.
(Cited herein on page 94.)

[Mu82] Murray · N.
"Complete Nonclausal Theorem Proving", in *Artificial Intelligence*, Volume 8 (1982), pages 67–85.
(Cited herein on page 94.)

[O88] Ohlbach · H.
"A Resolution Calculus for Modal Logics", in Lusk · E. and Overbeek · R. (editors): *9th International Conference on Automated Deduction (Proceedings)* (Lecture Notes in Computer Science #310), pages 500–516. Springer-Verlag, Berlin, 1988.
(Cited herein on page 91.)

[O93] Ohlbach · H.
"Translation Methods for Non-Classical Logics — An Overview", in *Automated Deduction in Nonstandard Logics* (Technical Report #FS-93-01), pages 113–125. AAAI Press, Menlo Park, California, 1993.
(Cited herein on pages 92 and 106.)

[Pl86] Plaisted · D.
"A Decision Procedure for Combinations of Propositional Temporal Logic and Other Specialized Theories", in *Journal of Automated Reasoning*, Volume 2 (1986), pages 171–190.
(Cited herein on page 91.)

[Pn77] Pnueli · A.
"The Temporal Logic of Programs", in *18th Annual Symposium on Foundations of Computer Science* (1977), pages 46–57.
(Cited herein on page 1.)

[Pn84] Pnueli · A.
"In Transition from Global to Modular Temporal Reasoning about Programs", in Apt · K. (editor): *Logics and Models of Concurrent Systems* (Proceedings, 1984), pages 123–144. Springer-Verlag, Heidelberg, 1985.
(Cited herein on page 1.)

[Pr78] Pratt · V.
"A Practical Decision Method for Propositional Dynamic Logic", in *Proceedings of the 10th Annual Symposium on Theory of Computing* (May 1978), pages 326–337.
(Cited herein on page 17, in a footnote.)

[SheP89] Sherman · R. and Pnueli · A.
"Model Checking for Linear Temporal Logic: An Efficient Implementation" (technical report). Information Science Institute, USC, 1989.
(Cited herein on pages 17 and 75.)

[Shoh87] Shoham · Y.
"Temporal Logics in AI: Semantical and Ontological Considerations", in *Artificial Intelligence*, Volume 33 (1987), pages 89–104.
(Cited herein on pages 2 and 91.)

[Shos79] Shostak · R.
"A practical decision procedure for arithmetic with function symbols", in *Journal of the Association for Computing Machinery*, Volume 26 (1979), Number 2 (April), pages 351–360.
(Cited herein on page 116, in a footnote.)

[SiC85] Sistla · A. and Clarke · E.
"The Complexity of Propositional Linear Temporal Logics", in *Journal of the Association for Computing Machinery*, Volume 32 (1985), Number 3 (July), pages 733–749.
(Cited herein on page 73.)

[So76] Soare · R.
"The Infinite Injury Priority Method", in *Journal of Symbolic Logic*, Volume 41 (1976), pages 513–530.
(Cited herein on page 18, in a footnote.)

[So87] Soare · R.
Recursively Enumerable Sets and Degrees (A Study of Computable Functions and Computably Generated Sets). Springer-Verlag, Berlin, 1987.
(Cited herein on page 18, in a footnote.)

[St75] Stickel · M.
"A Complete Unification Algorithm for Associative-Commutative Functions", in *International Joint Conference on Artificial Intelligence (Proceedings)* (1975), pages 71–82.
(Cited herein on page 111.)

[Sz86] Szalas · A.
"Concerning the Semantic Consequence Relation in First-Order Temporal Logic", in

Theoretical Computer Science, Volume 47 (1986), pages 329–334.
(Cited herein on page 133.)

[SzH88] Szalas · A. and Holenderski · L.
"Incompleteness of First-Order Temporal Logic with Until", in *Theoretical Computer Science*, Volume 57 (1988), pages 317–325.
(Cited herein on page 133.)

[Wa89] Wallen · L.
Automated Proof Search in Nonclassical Logics. The MIT Press, Cambridge, Massachusetts, 1989.
(Cited herein on page 91.)

[WiP89] Wilk · A. and Pnueli · A.
"Specification and Verification of VLSI Systems", in *1989 IEEE International Conference on Computer-Aided Design (Proceedings)*, pages 460–63. IEEE Computer Society Press, 1989.
(Cited herein on pages 1 and 75–78.)

[Wo83] Wolper · P.
"Temporal Logic Can Be More Expressive", in *Information and Control*, Volume 56 (1983), Numbers 1/2 (January/February), pages 72–99.
(Cited herein on page 86.)

[Wo85] Wolper · P.
"The Tableau Method for Temporal Logic: An Overview", in *Logique et Analyse*, Number 110/111 (June–September 1985), pages 119–136.
(Cited herein on page 17.)

[ZSS94] Zhang · S., Sokolosky · O., and Smolka · S.
"On the Parallel Complexity of Model-Checking in the Modal Mu-Calculus", in *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 154–163. IEEE Computer Society Press, 1994.
(Cited herein on page 88.)

Index