
Calendar Logic

Hans Jürgen Ohlbach and Dov Gabbay
Department of Computer Science
Kings College
Strand
London WC2R 2LS
England
(ohlbach,dg)@dcs.kcl.ac.uk

ABSTRACT. A propositional temporal logic is introduced whose operators quantify over intervals of a reference time line. The intervals are specified symbolically, for example ‘next week’s weekend’. The specification language for the intervals takes into account all the features of real calendar systems. A simple statement which can be expressed in this language is for example: ‘yesterday I worked for eight hours with one hour lunch break at noon’. Calendar Logic can be translated into propositional logic. Satisfiability is therefore decidable. Since the translation is exponential, a tableau decision procedure for checking decidability is presented as an alternative.

Keywords: temporal logic, calendar systems, tableau calculus

1 Introduction

Many Temporal Logics [4] are variants of propositional modal logic with two sets of operators ‘always in the future’, ‘sometimes in the future’, ‘always in the past’ and ‘sometimes in the past’. Other operators are ‘until’ and ‘since’ and a ‘next’ operator for integer time structures. Using such operators it is difficult to express statements referring to real world temporal notions, as for example ‘next Friday afternoon I’ll visit you’, or ‘yesterday I worked for eight hours with one hour lunch break at noon’, where the temporal notions are those of the given calendar system, and reasoning with these formulae takes into account the real meaning of notions like ‘next Friday’, ‘afternoon’, ‘hours’ etc.

In this paper we link temporal logic to real time calendar systems. Instead of the classical temporal operators we introduce relativized operators of the kind ‘sometimes next week’ or ‘always in the office hours’. Calendar Logic is

developed in different stages. We start with a system for specifying everyday temporal notions like ‘hours’, ‘weeks’, ‘weekend’, ‘office-hour’, ‘holiday’ within a given calendar system. In this system one can check whether a given point in time lies within an interval specified by a time term, or whether a subsumption relation holds between two time terms.

In the next stage these temporal notions are integrated into a propositional modal logic with two parameterized operators ‘sometimes within τ ’ and ‘always within τ ’ where τ may be one of the previously defined temporal notions. An example for a statement in this logic is

$$[2000, year] : \langle June(x_{year}) \rangle election.$$

It expresses that some time in June in the year 2000 there is an election. $[2000, year]$ denotes the time region corresponding to the year 2000. $June(x_{year})$ is a time term. It denotes a function that takes a year co-ordinate as argument and returns a month co-ordinate. Temporal expressions like these are not built-in, but they can be defined with some primitive constructors.

A characteristic feature of this logic is that the time regions specified by the temporal operators are always finite sets of finite intervals. There is no ‘sometimes in the future’ operator which refers to an unbounded time region. Instead of this, one can say for example ‘sometimes within the next million years’. This refers to a finite time region, which makes an important difference. The restriction to finite time regions actually is the reason that Calendar Logic can be translated into classical propositional logic, although at the expense of an exponential increase of the formulae’s size. Nevertheless, this result means that satisfiability of formulae in Calendar Logic is decidable. It can be decided with any propositional decision procedure. Because of the exponential increase in the size of the formulae this may not be the most efficient method for deciding satisfiability. As an alternative we therefore provide a tableau algorithm for Calendar Logic. The tableau method has more possibilities for guiding and optimizing the proof search.

2 The Time Term Language

The lowest tiers of Calendar Logic follow the ideas presented by Ohlbach in [5]. Therefore they are only sketched in this paper. The basis is a reference time line which is isomorphic to the set of real numbers. A good approximation of such a reference time line, the number of microseconds elapsed since 1.1.1970, is available in most UNIX systems. Ignoring the effects of relativity theory, this way, each time point from the time of the big bang until the end of the universe gets a unique identifier.

Definition 2.1 (Reference Time Line) *Let \mathcal{T} be an isomorphic copy of the set of real numbers. \mathcal{T} is the reference time line. The time point 0 is called the reference origin of time. \triangleleft*

Time Units

The reference time line is used to define a set of *time units*. These are the familiar time units such as seconds, minutes, hours, days, years, etc. Each time unit partitions the reference time line into an infinite sequence of consecutive intervals. The intervals are not necessarily of the same length. For example the intervals corresponding to leap years in the time unit ‘year’ are bigger than other years. But the intervals follow each other without gaps. Therefore they are themselves isomorphic to the integers.

If U is a time unit (for example ‘day’), let \mathcal{N}_U be U ’s copy of the integers, U ’s co-ordinate system.

For example for the time unit ‘day’, we may fix day 0 as 1.1.1970 and from there count all days as positive (after 1.1.1970) or negative (before 1.1.1970) integers. A time unit U is linked to the reference time line by a function $U_{\mathcal{N}}$ which maps reference time co-ordinates to the time unit’s own co-ordinate system, and a function U_{\llbracket} which maps a time unit’s co-ordinates to the (left closed, right open) interval in the reference time line corresponding to the time unit U . For example, if $U = \text{day}$, and \mathcal{T} is the UNIX reference time line then $day_{\mathcal{N}}(1000) = 0$ (microsecond number 1000 after 1.1.1970 lies still in the first day, day 0), and $day_{\llbracket}(0) = [0, 86400000[$ (the first day, 1.1.1970, lasted from microsecond 0 till just before microsecond 86400000).¹

The functions $U_{\mathcal{N}}$ and U_{\llbracket} encode the actual calendar system. They must take into account all irregularities, in particular leap years, daylight savings time and time zones. For example time zones are encoded by having for each time zone a separate set of time units for years, days etc. They are correlated indirectly via the reference time line. We assume that for each time unit the functions $U_{\mathcal{N}}$ and U_{\llbracket} are given (cf. [3]).

Definition 2.2 (Time Units) *Let \mathcal{U} be a set of time unit symbols.*

If $U \in \mathcal{U}$ is a time unit symbol, an interpretation $\mathfrak{S}_T(U)$ is defined as the triple $(\mathcal{N}_U, U_{\mathcal{N}}, U_{\llbracket})$ where

- \mathcal{N}_U is a U -colored isomorphic copy of the set \mathbb{Z} of integers.² \mathcal{N}_U is the time unit’s own co-ordinate system.
- $U_{\mathcal{N}} : \mathcal{T} \mapsto \mathcal{N}_U$ is a function mapping the reference time axis \mathcal{T} to the U co-ordinates.
- $U_{\llbracket} : \mathcal{N}_U \mapsto \text{Int}(\mathcal{T})$ is a function mapping the elements n of the U co-ordinates to the half open interval $[b, e[$ in the reference time line which corresponds to the U co-ordinate n . $\text{Int}(\mathcal{T})$ is the set of all half-open intervals over the reference time line.

We call a finite set $i \in \text{Int}(\mathcal{T})$ of half-open intervals a time region.

¹We use the standard notation $[b, e[$ for half open intervals of real numbers. b is the lower bound and belongs to the interval, e is the upper bound and does not belong to the interval.

²A formal definition of \mathcal{N}_U would be $\mathcal{N}_U = \{(i, U) \mid i \in \mathbb{Z}\}$ with the usual structure of integer numbers imposed on \mathcal{N}_U .

If $U_{\parallel}(n) = [b, e[$ then $U_{\parallel}^b(n) = b$ denotes the beginning of the interval and $U_{\parallel}^e(n) = e$ denotes the end of the interval.

We require:

- $U_{\parallel}^e(n) = U_{\parallel}^b(n + 1)$ for all $n \in \mathcal{N}_U$
(there are no gaps between U co-ordinates³);
- $\forall t \in \mathcal{T} : t \in U_{\parallel}(U_{\mathcal{N}}(t))$ and $\forall n \in \mathcal{N}_U : U_{\mathcal{N}}(U_{\parallel}(n)) = n$
($U_{\mathcal{N}}$ and U_{\parallel} are inverse to each other). ◁

It is important to notice that sets of half open intervals which are open all at the same side, in our case the upper side, are closed under union and intersection. We shall exploit this in the algorithms below.

Temporal Notions

Time units are the basis for a specification language \mathcal{T}_U for defining temporal notions like ‘weekend’, ‘office hour’, dates etc. A suitable specification language is defined in [5], but for the purposes of this paper, it is not necessary to fix such a language. There are only a few requirements. The language must be a sorted term language where the time units are used as sort symbols and each term has a unique sort. A term $\tau(x_1, \dots, x_n)$ in this language, with free variables x_1, \dots, x_n of sort U_1, \dots, U_n and ‘range sort’ U for τ itself, is interpreted as a function mapping U_i co-ordinates to *finite* sets of U co-ordinates. A useful constructor in this language is the $U_{\text{within}}V$ -constructor. For example $\text{month}_{\text{within}}\text{-year}(y_{\text{year}}, 2)$ maps a year co-ordinate y_{year} to its second month, February (actually to the singleton set consisting of February’s month co-ordinate).

$$\text{February}(y_{\text{year}}) \stackrel{\text{def}}{=} \text{month}_{\text{within}}\text{-year}(y_{\text{year}}, 2) \quad (1)$$

$$\text{Monday}(w_{\text{week}}) \stackrel{\text{def}}{=} \text{day}_{\text{within}}\text{-week}(w_{\text{week}}, 1) \quad (2)$$

are typical examples for specifying temporal notions in this language. Using the above specification of February, we can now quite easily determine the interval corresponding to this year’s February. If t is the current time point in the reference time line, then $y_{\text{year}} \stackrel{\text{def}}{=} \text{year}_{\mathcal{N}}(t)$ yields this year’s ‘year’ co-ordinate. $t' \stackrel{\text{def}}{=} \text{year}_{\parallel}^b(y_{\text{year}})$ yields the reference time point of the beginning of this year. $m \stackrel{\text{def}}{=} \text{month}_{\mathcal{N}}(t')$ yields the month co-ordinate of the first month in this year, February. ($m' \stackrel{\text{def}}{=} (m + 2) - 1$ yields the month co-ordinate of the second month in this year, February. (m' is the result of the evaluation of $\text{month}_{\text{within}}\text{-year}(y_{\text{year}}, 2)$.) $\text{month}_{\parallel}(m')$ now computes the interval in the reference time line that corresponds to this year’s February.

Another useful constructor is the U_{s} constructor. For example $\text{hour}_{\text{s}}(x_{\text{day}})$ yields the set of *hour* co-ordinates within the day x_{day} . Or vice versa, $\text{day}_{\text{s}}(y_{\text{hour}})$

³The ancient Egyptian year had 360 days. They did know that to keep the seasons right, one needs five more days. So they had five unaccounted gap days in which the usual social conventions were abandoned. This would be difficult to model in our system.

yields the corresponding *day* co-ordinate for y_{hour} . As an extreme example, $microsecond_s(z_{year})$ yields the set of *microsecond* co-ordinates corresponding to z_{year} .

Since the time term language deals with finite sets of integers, one can also have arithmetic operations and set operators as part of the language.

Like the `U_within_V`-constructor, for which we sketched a corresponding algorithm, all other constructors in the language $\mathcal{T}_{\mathcal{U}}$ also need to have an algorithm associated with them which actually computes the corresponding co-ordinates. The combination of these algorithms yields an interpreter for the time term language. With the $U_{\mathcal{N}}$ function it is then possible to convert reference time points into co-ordinates of the time units, apply the interpreter for the time terms, and then use U_{\parallel} to convert the time unit's co-ordinates back into intervals of the reference time line.

Definition 2.3 (Time Terms) For a set \mathcal{U} of time units, let $\mathcal{T}_{\mathcal{U}}$ be a many-sorted term language (time terms) such that

- i) \mathcal{U} is the set of sort symbols;
- ii) the terms τ in $\mathcal{T}_{\mathcal{U}}$ have a unique 'range sort' $sort(\tau) \in \mathcal{U}$;
- iii) the terms τ are interpreted by a fixed interpretation \mathfrak{S} such that for each reference time point $t \in \mathcal{T}$ we have $\mathfrak{S}(\tau, t) = \{u_1, \dots, u_k\} \subseteq \mathcal{N}_{sort(\tau)}$.
- iv) $\mathfrak{S}(\tau, t)$ can effectively be computed by an appropriated algorithm.
- v) For an interval $i \subseteq \mathcal{T}$, let $\mathfrak{S}(\tau, i) \stackrel{\text{def}}{=} \bigcup_{s \in i} \mathfrak{S}(\tau, s)$. (See Def. 2.7 for an algorithm which computes $\mathfrak{S}(\tau, i)$)
- vi) Since the interpretation \mathfrak{S} is fixed, we usually write $\tau(s)$ instead of $\mathfrak{S}(\tau, s)$ and $\tau(i)$ instead of $\mathfrak{S}(\tau, i)$.
- vii) For a time term τ with sort U , a time point t and a time region i let $\tau_{\parallel}(t) \stackrel{\text{def}}{=} \bigcup_{u \in \tau(t)} U_{\parallel}(u)$ and $\tau_{\parallel}(i) \stackrel{\text{def}}{=} \bigcup_{u \in \tau(i)} U_{\parallel}(u)$. \triangleleft

$\mathfrak{S}(\tau[u_1, \dots, u_m])(t)$ computes the value of $\tau[u_1, \dots, u_m]$ at a given reference time point t by first computing the corresponding U_i co-ordinates $U_{i_{\mathcal{N}}}(t)$, binding the variables u_i to these co-ordinates and then evaluating them in the usual way. The result is either a single U co-ordinate or a set of U co-ordinates.

The τ_{\parallel} function goes one step further and turns the computed U co-ordinates back into intervals in the reference time system.

One may use the same variables in different time terms. Since there is a kind of implicit lambda binding for these variables, they have nothing to do with each other.

Notice that $\tau_{\parallel}(i)$ may be an empty time region. A natural example where this may occur is the definition of *leap-day* as a function mapping year co-ordinates to day co-ordinates:

$$\begin{aligned} \text{leap-day}(x_{year}) &\stackrel{\text{def}}{=} \\ &(\text{last_day}(\text{February}(x_{year})) = \text{day_within_month}(\text{February}(x_{year}), 29)) ? \\ &\text{last_day}(\text{February}(x_{year})) : \{\} \end{aligned}$$

(If the last day of a February in a year is the 29th then the leap day is the 29th,

otherwise the leap day is empty.) $(\text{leap-day}(x_{\text{year}}))_{\parallel}$ is empty for all non-leap years.

Using the interpreter \mathfrak{S} we can check the *instance relation* between a given reference time point t and a time term τ by checking whether $t \in \tau_{\parallel}$ holds or not.

A more complicated relation than the instance relation is the *subsumption relation*. An example might be the problem to figure out whether somebody's morning office hours in Brasil lie always within my local afternoon. Unfortunately this general subsumption relation is usually not decidable. This is due to the fact that we treat the functions $U_{\mathcal{N}}$ and U_{\parallel} as a kind of black boxes where the most peculiar things may be encoded, for example a transition to a completely new calendar system in the year 2100.

For temporal relationships from everyday life, this general subsumption relationship, however, is really too general. A restricted subsumption relation is much more natural and useful.

Definition 2.4 (Subsumption Relation) *A term τ_1 is subsumed by a term τ_2 ($\tau_1 \subseteq \tau_2$) iff for all time points t in the reference system, $\tau_{1\parallel}(t) \subseteq \tau_{2\parallel}(t)$.*

A restricted subsumption relation ($\tau_1 \subseteq_i \tau_2$) for the time region i holds between τ_1 and τ_2 iff for all time points $t \in i$, $\tau_{1\parallel}(t) \subseteq \tau_{2\parallel}(t)$. \triangleleft

Since the restricted subsumption relation still quantifies over an infinite number of time points, this definition is not yet a suitable basis for an algorithm. In order to decide restricted subsumption, we exploit that the time terms depend on the co-ordinates of the time units, and they are therefore constant over the period in the reference time line corresponding to the same co-ordinates of the time unit. For example the time term *February*(y) (definition (1)) depends on the year co-ordinate. It is therefore constant over a whole year, i.e. over all the points t in the reference time line with $\text{year}_{\mathcal{N}}(t) = y$.

Exploiting this observation, each time term $\tau[u_1, \dots, u_n]$ can be used to partition each finite interval in the reference time line into the finitely many sub-intervals where $\tau[u_1, \dots, u_n]$ represents a constant function.

As a simple example, consider i to be the week 53 in the year 1970. i overlaps partially with 1970 and partially with 1971. *February*(x_{year})(i) therefore yields the month co-ordinates for the February in 1970 and 1971. This is computed by first partitioning i into the part of the week lying in the year 1970, and the part of the week lying in 1971. For each part, *February*(x_{year}) is then evaluated separately.

This partitioning can then be used to trigger finite case analysis for checking whether τ_1 subsumes τ_2 over a time region i . i is partitioned into the set $\{i_1, \dots, i_n\}$ of sub-regions over which τ_1 and τ_2 is constant, and then for $1 \leq k \leq n$, $\tau_{1\parallel}(t_k) \subseteq \tau_{2\parallel}(t_k)$ is checked for some arbitrary chosen $t_k \in i_k$.

Definition 2.5 (Partitioning Algorithm)

For a non-empty interval $i = [i^b, i^e[\subseteq \mathcal{T}$ and a time term $\tau[u_1, \dots, u_n]$ with

$U_k \stackrel{\text{def}}{=} \text{sort}(u_k)$, $k = 1, \dots, n$, we define the partitioning of i with respect to τ as

$$\delta(i, \tau) = \{[i^b, t[\cup \delta([t, i^e[, \tau)$$

where

$$t \stackrel{\text{def}}{=} \begin{cases} \min(\{U_{k\llbracket}(U_{k\mathcal{N}}(i^b) + 1) \in i \mid 1 \leq k \leq n\}) \\ i^e \text{ if this set is empty.} \end{cases}$$

If i is a set of intervals, let $\delta(i, \tau) \stackrel{\text{def}}{=} \bigcup_{j \in i} \delta(j, \tau)$.

Since $\delta(i, \tau)$ only depends on the free variables $\{u_1, \dots, u_n\}$ in τ , we can write $\delta(i, \{u_1, \dots, u_n\})$ for $\delta(i, \tau)$.

This version can be generalized to sets $\{\tau_1, \dots, \tau_m\}$ of time terms.

If $\{u_1, \dots, u_n\}$ is the set of all free variables occurring in $\{\tau_1, \dots, \tau_m\}$, we get

$$\delta(i, \{\tau_1, \dots, \tau_m\}) \stackrel{\text{def}}{=} \delta(i, \{u_1, \dots, u_n\})$$

and this computed as indicated above. \triangleleft

The partitioning algorithm decomposes i by computing the U_k co-ordinate $U_{k\mathcal{N}}(i^b)$ corresponding to the beginning of the interval i and then checking whether the beginning of the next interval corresponding to the next U_k -coordinate still lies in i or not. If it still lies in i , this is a candidate for the lower border of the next sub-interval. The real border is determined by the time unit U_k which gives the smallest such border. (A properly implemented algorithm would of course try the most fine grained time unit U_k first, e.g. seconds before minutes before hours etc.)

As an example, suppose we want to partition the last week in 1970 (Sunday, December 27 till Saturday, January 2 1971) into the intervals where $February(x_{year})$ is constant. For the standard UNIX reference time (in thousand seconds) the algorithm yields

$$\delta([31104, 31708[, February(x_{year})) = \{[31104, 31536[, [31536, 31708[\}.$$

Notice that the partitioning function only depends on the time units corresponding to τ 's free variables, not on τ itself.

Proposition 2.6 (Soundness of the Partitioning Algorithm) *For each interval $j \in \delta(i, \tau)$ (definition 2.5) and for each $\{t_1, t_2\} \subseteq j$: $\tau_{\llbracket}(t_1) = \tau_{\llbracket}(t_2)$.*

Proof: by induction on the number of intervals in $\delta(i, \tau)$. In the base case, $\delta(i, \tau) = i$, i.e. i itself is the only component, and for $1 \leq k \leq n$, $U_{k\llbracket}(U_{k\mathcal{N}}(i^b) + 1) > i^e$. Therefore for all $t \in i$, $U_{k\mathcal{N}}(t)$ yields the same U_k co-ordinate. Since $\tau[u_1, \dots, u_n]$ depends on the U_k co-ordinates only, $\tau_{\llbracket}(t_1) = \tau_{\llbracket}(t_2)$ for all $\{t_1, t_2\} \subseteq i$.

In the induction step, $\delta(i, \tau)$ consists of more than one interval and $[i^b, s[$ is the leftmost sub-interval of the decomposition. According to the definition of δ , t is the smallest value such that $t = U_{k\llbracket}(U_{k\mathcal{N}}(i^b) + 1)$ lies still in i . That means for all $k \in \{1, \dots, n\}$: $U_{k\mathcal{N}}(t_1) = U_{k\mathcal{N}}(t_2)$ for all $t_1 \in [i^b, t[$ and $t_2 \in [i^b, t[$. Thus, $\tau_{\llbracket}(t_1) = \tau_{\llbracket}(t_2)$ for the first interval in $\delta(i, \tau)$. The induction hypothesis directly applies to the remaining intervals. \triangleleft

Proposition 2.7 (Evaluation of Time Terms over Intervals)

For a \mathcal{T}_U -formula τ and a finite half open interval $i \subseteq \mathcal{T}$:

$$\tau(i) = \bigcup_{j \in \delta(i, \tau)} \tau(j^b).$$

Proof: $\delta(i, \tau)$ partitions i into the finitely many sub-intervals where τ is a constant function. Therefore for each such sub-interval j we can just choose an arbitrary element s , in particular the first one, $s = j^b$ to compute $\tau(s)$ and join these intervals together. \triangleleft

Proposition 2.8 (Decidability)

- The instance relation is decidable. $\tau_{\parallel}(t)$ yields only finitely many intervals. Therefore just check $t \in \tau_{\parallel}(t)$, which amounts to finding some $[i^b, i^e[\in \tau_{\parallel}(t)$ with $i^b \leq t < i^i$.
- Without special assumptions about the functions $U_{\mathcal{N}}$ and U_{\parallel} , the subsumption relation is in general not decidable.
- The restricted subsumption relation $\tau_1 \subseteq_i \tau_2$ over an interval $i = [t_1, t_2[$ is decidable. Using the partitioning algorithm of Definition 2.5, partition this interval into the finitely many sub-intervals where the two time terms are constant functions, choose for each sub-interval j some point $t \in j$, and check the subsumption relation $\tau_{1\parallel}(t) \subseteq \tau_{2\parallel}(t)$ for this point. To do this, check for each $[i^b, i^e[\in \tau_{1\parallel}(t)$ whether there is some $[j^b, j^e[\in \tau_{2\parallel}(t)$ with $[i^b, i^e[\subseteq [j^b, j^e[$.⁴ \triangleleft

3 Calendar Logic

Calendar Logic \mathcal{CL} is the next stage in our system. Its syntax is very similar to the syntax of propositional multi-modal logic K_m [2, 1]. There are the parameterized modal operators $\langle \tau \rangle$ and $[\tau]$ where τ is a time term in the language \mathcal{T}_U . The intuitive meaning for a formula like

$$[February(y_{year})]sunshine$$

is ‘it was sunshine throughout this year’s February’.

This formula is interpreted in a given reference time point t as follows: from t one gets $y_{year} = year_{\mathcal{N}}(t)$, from that the interval corresponding to $February(y_{year})$, and in *all* points (semantics of the box operator) of this interval, *sunshine* must be true.

Whereas the $[\tau]$ -operator is a universal quantification over the points corresponding to $\tau(s)$, $\langle \tau \rangle$ is the dual operator and works as an existential quantification. The formula

$$\langle x_{day} + 1 \rangle go-to-cinema$$

⁴ $\tau_{\parallel}(t)$ should be computed in such a way that the sequences of intervals without gaps are comprised into one single interval.

expresses ‘tomorrow I’ll go to the cinema’. $\langle x_{day} + 1 \rangle$ denotes some particular time point tomorrow, at which *go-to-cinema* is supposed to be true. One can also express ‘tomorrow I’ll go to the cinema’ together with the extra fact ‘and I’ll stay there for two hours’ by

$$\langle x_{day} + 1 \rangle (go\text{-}to\text{-}cinema \wedge [\{y_{hour}, y_{hour} + 1\}]in\text{-}cinema).$$

This formula restricts the duration of the two hours to the interval corresponding to the beginning of the first hour till the end of the second hour. With a suitable constructor in the time term language it is also possible to refer to an interval of two hours length starting at an arbitrary point in time.

Notice that in Calendar Logic there are no classical temporal logic operators like ‘always in the future’. Instead of this, however, we may for example have an ‘always in the next century’ operator.

There is an extra feature of Calendar Logic which has no correspondence in Modal Logic. We allow for time terms at formula positions. For example

$$[x_{year}](first_day(y_{month}) \Rightarrow pay\text{-}salary)$$

expresses ‘this, year, every first day in a this year pay the salary’.

$first_day(y_{month})$ is a $\mathcal{T}_{\mathcal{U}}$ -term at a formula position. It is true at a reference time point if this time point lies in the interval specified by $first_day(y_{month})$.

If t is again a reference time point, then the interpretation of this formula works as follows: $x_{year} = year_{\mathcal{N}}(t)$ yields the current year co-ordinate. For each (semantics of $[\tau]$) reference time point t' corresponding to x_{year} one must check whether $first_day(y_{month}) \Rightarrow pay\text{-}salary$ holds at t' . With $y_{month} = month_{\mathcal{N}}(t')$ we get the month co-ordinate m of t' . The evaluation of $first_day$ for m yields a day co-ordinate. If $t' \in day_{\llbracket}(m)$ then $first_day(y_{month})$ is true for t' , and we have to check whether $pay\text{-}salary$ holds as well at t' .

We may also have subsumption relationships of time terms at formula positions. For example

$$\langle x_{day} + 1 \rangle (noon(x_{day}) \subseteq lunch\text{-}time(x_{day}) \Rightarrow ring\text{-}home)$$

expresses ‘if tomorrow’s lunch time is at noon, I’ll ring home’. $lunch\text{-}time(x_{day})$ and $noon(x_{day})$ are both time terms. The subsumption relationship is true at a time point t if the time region $noon(x_{day})_{\llbracket}(t)$ is a subset of $lunch\text{-}time(x_{day})_{\llbracket}(t)$.

Subsumption relations in both directions, i.e. $\tau_1 \subseteq \tau_2 \wedge \tau_2 \subseteq \tau_1$ can be abbreviated using the equality symbol: $\tau_1 = \tau_2$. (3) is an example for the use of equations between time terms.

Definition 3.1 (Syntax of Calendar Logic) *If $\mathcal{T}_{\mathcal{U}}$ is a language of time terms (definition 2.3) and \mathbb{P} a set of predicate symbols then the formulae of Calendar Logic $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ are built according to the following grammar:*

$$\begin{aligned} \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} := & \mathbb{P} \mid \mathcal{T}_{\mathcal{U}} \mid \mathcal{T}_{\mathcal{U}} \subseteq \mathcal{T}_{\mathcal{U}} \mid \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} \wedge \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} \mid \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} \vee \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} \mid \\ & \langle \mathcal{T}_{\mathcal{U}} \rangle \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}} \mid [\mathcal{T}_{\mathcal{U}}] \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}. \end{aligned}$$

$\tau_1 = \tau_2$ abbreviates $\tau_1 \subseteq \tau_2 \wedge \tau_2 \subseteq \tau_1$.

A statement of Calendar Logic is either $\langle n, U \rangle : \varphi$ or $[n, U] : \varphi$ where U is a time unit, n is one of U 's co-ordinates and φ is a Calendar Logic formula.

◁

A Calendar Logic formula is interpreted relative to some time point in the reference time line. For expressing a fact about a very concrete point in time, extra syntax is necessary. One could for example express a fact as a tuple $t : \varphi$ where t is a point in the reference time line, i.e. a real number. These real numbers, however, will never be explicitly accessible. They are usually deeply hidden in some time checking devices. Moreover, for the purpose of associating a particular event with the time when it happened, points in the reference time line are in general much too sharp. In most cases, the time when an event happened, can be located only within a certain interval. Therefore we choose the syntax $\langle n, U \rangle : \varphi$ or $[n, U] : \varphi$ for expressing statements about events happening at some point in time. U is a time unit (second, minute, hour etc.), and n is a concrete co-ordinate of this time unit. $\langle n, U \rangle : \varphi$ means, φ is true at *some* point in the interval corresponding to the U co-ordinate n , and $[n, U] : \varphi$ means, φ is true at *all* points in the interval corresponding to the U co-ordinate n . For example the event ‘Jack was born in 1980’ would be expressed as $\langle 10, year \rangle : born(Jack)$ where 10 is the tenth year in the UNIX reference time line starting with 1970. As another example ‘in 1996 Jack lived in London’ could be expressed as $[26, year] : lives-in(London, Jack)$ where the box operator expresses that Jack lived in London all the time in the year 1996.

Finally let’s see how our introductory example ‘yesterday I worked for eight hours with one hour lunch break at noon’ could be expressed in Calendar Logic. Let d be today’s day co-ordinate. A first formalization of this statement is

$$\langle d, day \rangle : \langle x_{day} - 1 \rangle ((\{y_{hour}, \dots, y_{hour} + 8\} \setminus noon(z_{day})) work) \\ \wedge [noon(z_{day})] lunch$$

with $noon(z_{day}) \stackrel{\text{def}}{=} \text{hour_within_day}(z_{day}, 12)$.

This statement would be true even if I started working yesterday at 11pm. To insist that the whole period of work was yesterday, one could write

$$\langle d, day \rangle : \langle x_{day} - 1 \rangle ((\{y_{hour}, \dots, y_{hour} + 8\} \setminus noon(z_{day})) work) \\ \wedge day_s(y_{hour}) = day_s(y_{hour} + 8) \wedge [noon(z_{day})] lunch \quad (3)$$

$day_s(y_{hour}) = day_s(y_{hour} + 8)$ restricts the *hour* co-ordinate y_{hour} where I began to work, such that 8 hours later is still the same day.

Definition 3.2 (Semantics of Calendar Logic)

An interpretation $\mathfrak{S} \stackrel{\text{def}}{=} (\mathfrak{S}_{\mathcal{T}_{\mathcal{U}}}, \mathcal{P})$ for the formulae of Calendar Logic $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ consists of a fixed $\mathcal{T}_{\mathcal{U}}$ -interpretation $\mathfrak{S}_{\mathcal{T}_{\mathcal{U}}}$ (definition 2.3) for the time terms $\mathcal{T}_{\mathcal{U}}$, and an assignment $\mathcal{P} : \mathbb{P} \mapsto 2^{\mathcal{T}}$ which assigns to each predicate symbol $p \in \mathbb{P}$ the set of reference time points where p is true.

For a reference time point $t \in \mathcal{T}$ we define:

$$\begin{array}{lll}
\mathfrak{S}, t \models p & \text{iff} & t \in \mathcal{P}(p) & \text{where } p \in \mathbb{P} \\
\mathfrak{S}, t \models \tau & \text{iff} & t \in \tau_{\parallel}(t) & \text{where } \tau \in \mathcal{T}_{\mathcal{U}} \\
\mathfrak{S}, t \models \tau_1 \subseteq \tau_2 & \text{iff} & \tau_{1\parallel}(t) \subseteq \tau_{2\parallel}(t) & \\
\mathfrak{S}, t \models \langle \tau \rangle \varphi & \text{iff} & \mathfrak{S}, t' \models \varphi \text{ for some } t' \in \tau_{\parallel}(t) & \\
\mathfrak{S}, t \models [\tau] \varphi & \text{iff} & \mathfrak{S}, t' \models \varphi \text{ for all } t' \in \tau_{\parallel}(t) & \\
\\
\mathfrak{S} \models \langle n, U \rangle : \varphi & \text{iff} & \mathfrak{S}, t \models \varphi \text{ for some } t \in U_{\parallel}(n) & \\
\mathfrak{S} \models [n, U] : \varphi & \text{iff} & \mathfrak{S}, t \models \varphi \text{ for all } t \in U_{\parallel}(n). &
\end{array}$$

Algorithms operating on logical formulae are usually more efficient and easier to define when the formulae are in some kind of normal form. The *negation normal form* where all negation symbols are moved inside as far as possible simplifies the treatment of negation considerably.

Definition 3.3 (Negation Normal form) *There is a negation normal form for $\mathcal{C}\mathcal{L}_{\mathcal{T}_{\mathcal{U}}}$ -formulae where all negation symbols are moved in front of the predicate symbols or the time terms. An arbitrary $\mathcal{C}\mathcal{L}_{\mathcal{T}_{\mathcal{U}}}$ -formula or statement can be brought into negation normal form using the following rewrite system:*

$$\begin{array}{l}
\neg\neg\varphi \rightarrow \varphi \\
\neg(\varphi_1 \wedge \varphi_2) \rightarrow \neg\varphi_1 \vee \neg\varphi_2 \\
\neg(\varphi_1 \vee \varphi_2) \rightarrow \neg\varphi_1 \wedge \neg\varphi_2 \\
\neg\langle \tau \rangle \varphi \rightarrow [\tau]\neg\varphi \\
\neg[\tau]\varphi \rightarrow \langle \tau \rangle \neg\varphi \\
\neg\langle n, U \rangle : \varphi \rightarrow [n, U] : \neg\varphi \\
\neg[n, U] : \varphi \rightarrow \langle n, U \rangle : \neg\varphi.
\end{array}$$

◁

It is easy to check that the negation normal form algorithm is an equivalence preserving transformation on $\mathcal{C}\mathcal{L}_{\mathcal{T}_{\mathcal{U}}}$ -formulae and statements. Putting formulae into negation normal form is only a critical operation if many equivalences are present. $\varphi \Leftrightarrow \psi$ is turned into $(\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$. This is an exponential transformation if φ or ψ themselves contain equivalences. By introducing predicate symbols as abbreviations for φ and ψ , one can avoid this — a well-known trick in predicate logic theorem proving.

In first-order predicate logic there is a particularly useful normal form, the *conjunctive normal form*, or *clause form*. It consists of conjunctions of clauses, and each clause is a disjunction of literals. Literals are positive or negative atoms $p(t_1, \dots, t_n)$. This simple normal form is not possible for $\mathcal{C}\mathcal{L}_{\mathcal{T}_{\mathcal{U}}}$ -formulae because the temporal operators do not distribute over conjunction and disjunction. Nevertheless, there is a similar normal form, which we also call *clause normal form*. In this clause form the conjunctions are moved to the toplevel as far as possible, and the disjunctions are moved down as far as possible. The clause normal form is the basis for a further transformation which eliminates the temporal operators altogether.

Definition 3.4 (Clause Normal Form) *The equivalences*

$$\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \Leftrightarrow (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3) \quad (\text{distributivity}) \quad (4)$$

$$\varphi_1 \wedge (\varphi_2 \vee \varphi_3) \Leftrightarrow (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3) \quad (\text{distributivity}) \quad (5)$$

$$[\tau](\varphi_1 \wedge \varphi_2) \Leftrightarrow [\tau]\varphi_1 \wedge [\tau]\varphi_2 \quad (6)$$

$$\langle \tau \rangle(\varphi_1 \vee \varphi_2) \Leftrightarrow \langle \tau \rangle\varphi_1 \vee \langle \tau \rangle\varphi_2 \quad (7)$$

can be used as a rewrite rules to put all $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ -formulae and statements from its negation normal form into a clause normal form.

The disjunctive clauses and conjunctive clauses of the clause normal form for $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ -formulae look like

$$\bigvee_k [\tau_k]\varphi_k \vee \bigvee_l \langle \pi_l \rangle \psi_l \vee \bigvee_m \tau_m \vee \bigvee_n (\tau_{n1} \subseteq \tau_{n2}) \vee \varphi \quad (8)$$

$$\bigwedge_k [\tau_k]\varphi_k \wedge \bigwedge_l \langle \pi_l \rangle \psi_l \wedge \bigwedge_m \tau_m \wedge \bigwedge_n (\tau_{n1} \subseteq \tau_{n2}) \wedge \varphi \quad (9)$$

where the φ_k are disjunctive clauses (due to (6)) and the ψ_l are conjunctive clauses (due to (7)), the τ_m , the τ_{n1} and the τ_{n2} are $\mathcal{T}_{\mathcal{U}}$ -terms. φ is a disjunction (conjunction) of positive or negated predicate symbols. \triangleleft

From now on we shall assume that all formulae and statements are put into clause normal form.

4 Translation of $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ into Propositional Logic

$\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ -statements $[n, U]\varphi$ and $\langle n, U \rangle\varphi$ specify a finite time region $i = U_{\llbracket}(n)$ within which φ is supposed to be valid. φ itself may contain temporal operators $[\tau]$ and $\langle \tau \rangle$ specifying other finite time ranges $\tau_{\llbracket}(i)$ within which the corresponding subformulae are supposed to hold. Since $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ -formulae are finite, there are only finitely many time regions involved, and they can all be computed as sets of half-open intervals in the reference time line. This observation is exploited to define a sequence of further transformations which finally eliminates all the temporal operators and leaves us with propositional formulae.

This means, satisfiability in $\mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}$ is decidable by any decision procedure for propositional logic. The bad news is that the translation into propositional logic may blow up the formulae exponentially. The worst case time complexity of this method for deciding satisfiability is therefore double exponential.

The translation into propositional logic consists of four different steps. For the first step two further *time region* operators $[i]$ and $\langle i \rangle$ are introduced. The parameter i is a concrete time range. $\langle [10, 100[, [200, 300[]\varphi$ for example means that φ is true at some time either between 10, and 100 or between 200 and 300 in the reference time line. $[i]\varphi$ means that φ is true in *all* points $t \in i$ and $\langle i \rangle\varphi$ means that φ is true in *some* point $t \in i$.

The translation \rightarrow_I defined below turns the temporal operators $[\tau]$ and $\langle \tau \rangle$ into time region operators $[i]$ and $\langle i \rangle$.

Example 4.1 (temporal operators to time region operators)

In these examples we assume the reference time line is counted in days, starting from the year 2000. The numbers are approximative.

Example 1: $[2000, year] : [January(x_{year})]election.$

(in January in the year 2000 there is an election.)

$[2000, year]$ denotes a time region $[0, 366[$ and the January in the year 2000 ranges from day 0 to the end of day 31. Therefore the translation yields

$[[0, 366[]][0, 32[]]election.$

Notice that the first box operator is no longer necessary. $[[0, 32[]]election$ contains the same information as the formula with the two box operators. Since the semantics of the time region operators does not depend on the actual time point, this is no coincidence.

Example 2:

$[2000, year] : \langle January(x_{year}) \rangle (sunshine \Rightarrow \langle weekend(y_{week}) \rangle visit).$

(I will visit you at some weekend in January 2000, if the sun is shining at that time).

The first two translation steps are the same as above. The result is

$[[0, 366[]]\langle [0, 32[] \rangle (sunshine \Rightarrow \langle weekend(y_{week}) \rangle) visit.$

The last translation step must take into account that there are different weekends in January 2000, and we don't know which of them the diamond operator refers to. A translation into

$$[[0, 366[]]\langle [0, 32[] \rangle (sunshine \Rightarrow \langle [5, 7[, [12, 14[, [19, 21[, [26, 29[] \rangle) visit).$$

would be wrong. This formula can be satisfied in a situation where sunshine is at the first of January and the visit would be in the last weekend of January. The meaning of the original formula, however, is more restrictive. The implication $sunshine \Rightarrow \langle weekend(y_{week}) \rangle visit$ enforces that the weekend of the week in which sunshine is true is meant, and not any weekend in January.

The original meaning of the implication can be preserved, if the translation inserts a case analysis at a higher level. To this end the time region corresponding to $January(x_{year})$ is split into the sub-ranges where $weekend(y_{week})$ is constant. Since there are four weeks in January, we get four sub-ranges. The correct translation is then

$$[[0, 366[]] (\langle [0, 7[] \rangle (sunshine \Rightarrow \langle [5, 7[] \rangle visit) \vee \\ \langle [7, 14[] \rangle (sunshine \Rightarrow \langle [12, 14[] \rangle) visit) \vee \\ \langle [14, 21[] \rangle (sunshine \Rightarrow \langle [19, 21[] \rangle) visit) \vee \\ \langle [21, 29[] \rangle (sunshine \Rightarrow \langle [26, 29[] \rangle) visit)).$$

This formula is false if the sunshine is in the first week and the visit is in the last week of January. It can only be made true if the visit is in the same week as the sunshine. \triangleleft

The temporal operators in temporal statements like $[n, U] : \varphi$ are translated recursively into time region operators. We start with turning the $[n, U]$

operator into $[U_{\parallel}(n)]$. Now the time region for which φ is supposed to hold is made explicit, namely $U_{\parallel}(n)$. In the recursive part of the translation, we deal with formulae like $[i]\langle\tau\rangle\varphi$ where i is a time region and τ a time term. The time region i is split into the sub-regions $\{i_1, \dots, i_h\} = \delta(i, \tau)$ where τ is constant. $[i]\langle\tau\rangle\varphi$ is first turned into $[i_1]\langle\tau\rangle\varphi \wedge \dots \wedge [i_h]\langle\tau\rangle\varphi$ and then into $[i_1]\langle\tau_{\parallel}(i_1)\rangle\varphi \wedge \dots \wedge [i_h]\langle\tau_{\parallel}(i_h)\rangle\varphi$. The $\tau_{\parallel}(i_k)$ are now time regions, and the recursive translation can go down into φ .

Definition 4.2 ($\mathcal{CL}_{\mathcal{T}_U}^I$) $\mathcal{CL}_{\mathcal{T}_U}^I$ is defined as an extension of Calendar Logic with two further time region operators $[i]$ and $\langle i \rangle$ where i is not a time term, but a time region (set of half-open intervals). The semantics of the operators is:

$$\begin{aligned} \mathfrak{S}, t \models [i]\varphi & \text{ iff } \mathfrak{S}, t' \models \varphi \text{ for all } t' \in i \\ \mathfrak{S}, t \models \langle i \rangle\varphi & \text{ iff } \mathfrak{S}, t' \models \varphi \text{ for some } t' \in i. \end{aligned}$$

Notice that the semantics of these operators does not depend on the actual time point t . Therefore instead of the notation $\mathfrak{S}, t \models [i] \dots$ or $\mathfrak{S}, t \models \langle i \rangle \dots$ the irrelevant t can be dropped by simply writing $\mathfrak{S} \models [i] \dots$ or $\mathfrak{S} \models \langle i \rangle \dots$. \triangleleft

Definition 4.3 (Translation of $\mathcal{CL}_{\mathcal{T}_U}$ into $\mathcal{CL}_{\mathcal{T}_U}^I$) We define rewrite rules for translating the $\mathcal{CL}_{\mathcal{T}_U}$ -operators into time region operators.

The rewrite rules for $\mathcal{CL}_{\mathcal{T}_U}$ -statements are

$$\begin{aligned} [n, U] : \varphi & \rightarrow_I [U_{\parallel}(n)]\varphi \\ \langle n, U \rangle : \varphi & \rightarrow_I \langle U_{\parallel}(n) \rangle\varphi. \end{aligned}$$

The rewrite rules for $\mathcal{CL}_{\mathcal{T}_U}$ -formulae are defined for the clause normal form (definition 3.4).

$$\begin{aligned} & [i](\bigvee_k [\tau_k]\varphi_k \vee \bigvee_l \langle\tau_l\rangle\varphi_l \vee \varphi) \\ & \quad \rightarrow_I \\ & \bigwedge_{r=1}^h [i_r](\bigvee_k [\tau_{k\parallel}(i_r)]\varphi_k \vee \bigvee_l \langle\tau_{l\parallel}(i_r)\rangle\varphi_l \vee \varphi) \\ & \quad \langle i \rangle(\bigwedge_k [\tau_k]\varphi_k \wedge \bigwedge_l \langle\tau_l\rangle\varphi_l \wedge \varphi) \\ & \quad \rightarrow_I \\ & \bigvee_{r=1}^h \langle i_r \rangle(\bigwedge_k [\tau_{k\parallel}(i_r)]\varphi_k \wedge \bigwedge_l \langle\tau_{l\parallel}(i_r)\rangle\varphi_l \wedge \varphi) \end{aligned}$$

where

- $[i]$ and $\langle i \rangle$ are time region operators;
- φ is a disjunction(conjunction) of temporal statements τ , subsumption relations $\tau_1 \subseteq \tau_2$ and propositional literals;
- $\{i_1, \dots, i_h\} \stackrel{\text{def}}{=} \delta(i, \{\tau_1, \dots\})$ and $\{\tau_1, \dots\}$ are all the time terms occurring at toplevel in the clauses and in φ . \triangleleft

Proposition 4.4 The translation \rightarrow_I is an equivalence preserving transformation.

Proof: for the $\mathcal{CL}_{\mathcal{T}_d}$ -statements:

$$\begin{aligned} \mathfrak{S} \models [n, U] \varphi & \text{ iff } \mathfrak{S}, t \models \varphi & \text{ for all } t \in U_{\parallel}(n) \\ & \text{ iff } \mathfrak{S}, t \models [U_{\parallel}(n)] \varphi & \text{ for all } t \\ \mathfrak{S} \models \langle n, U \rangle \varphi & \text{ iff } \mathfrak{S}, t \models \varphi & \text{ for some } t \in U_{\parallel}(n) \\ & \text{ iff } \mathfrak{S}, t \models \langle U_{\parallel}(n) \rangle \varphi & \text{ for all } t. \end{aligned}$$

For $\mathcal{CL}_{\mathcal{T}_d}$ -formulae:

$$\begin{aligned} \mathfrak{S} \models [i](\bigvee_k [\tau_k] \varphi_k \vee \bigvee_l \langle \tau_l \rangle \varphi_l \vee \varphi) \\ \text{iff } \mathfrak{S}, t \models (\bigvee_k [\tau_k] \varphi_k \vee \bigvee_l \langle \tau_l \rangle \varphi_l \vee \varphi) \\ \text{for all } t \in i \\ \text{iff } \mathfrak{S}, t \models (\bigvee_k [\tau_k] \varphi_k \vee \bigvee_l \langle \tau_l \rangle \varphi_l \vee \varphi) \\ \text{for all } i_r \in \delta(i, \{\tau_1, \dots\}) \text{ and for all } t \in i_r \\ \text{iff } \bigvee_k (\forall t' \in \tau_{k\parallel}(i_r) : \mathfrak{S}, t' \models \varphi_k) \text{ or } \bigvee_l (\exists t' \in \tau_{l\parallel}(i_r) : \mathfrak{S}, t' \models \varphi_l) \text{ or } \mathfrak{S}, t \models \varphi \\ \text{for all } i_r \in \delta(i, \{\tau_1, \dots\}) \text{ and for all } t \in i_r \\ \text{(this is because the time terms } \tau \text{ are constant over } i_r) \\ \text{iff } \bigvee_k (\mathfrak{S} \models [\tau_{k\parallel}(i_r)] \varphi_k) \text{ or } \bigvee_l (\mathfrak{S} \models \langle \tau_{l\parallel}(i_r) \rangle \varphi_l) \text{ or } \mathfrak{S}, t \models \varphi \\ \text{for all } i_r \in \delta(i, \{\tau_1, \dots\}) \text{ and for all } t \in i_r \\ \text{iff } \mathfrak{S} \models \bigwedge_{r=1}^h [i_r](\bigvee_k [\tau_{k\parallel}(i_r)] \varphi_k \vee \bigvee_l \langle \tau_{l\parallel}(i_r) \rangle \varphi_l \vee \varphi). \end{aligned}$$

The proof for the rule for the diamond operator is dual to the proof for the box operator. \triangleleft

It turns out that the time terms τ at formula positions and the subsumption relations $\tau_1 \subseteq \tau_2$ can be eliminated completely after this translation step. The next examples illustrate the idea.

Example 4.5 (for Eliminating Temporal Formulae) *We use the same reference time line as in example 4.1. Consider*

$$[2000, \text{year}] : \langle \text{January}(x_{\text{year}}) \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election}).$$

(some Friday in January 2000 there is an election.) The first two translation steps are as in example 4.1: $[[0, 366]] \langle [0, 32] \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election})$ In the next step, $\langle [0, 32] \rangle$ has to be decomposed into the sub-ranges where $\text{Friday}(y_{\text{week}})$ is constant. These are the four weeks in January. Thus, we get

$$\begin{aligned} [[0, 366]] (\langle [0, 7] \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election})) \vee \\ (\langle [7, 14] \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election})) \vee \\ (\langle [14, 21] \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election})) \vee \\ (\langle [21, 29] \rangle (\text{Friday}(y_{\text{week}}) \Rightarrow \text{election})). \end{aligned}$$

Now the time regions in the diamond operator can be reduced to the time regions where Friday is true. The result is

$$[[0, 366]] (\langle [5, 6] \rangle \text{election} \vee \langle [12, 13] \rangle \text{election} \vee \langle [19, 20] \rangle \text{election} \vee \langle [26, 27] \rangle \text{election})$$

(This is in fact equivalent to $[[0, 366]] \langle [5, 6[, [12, 13[, [19, 20[, [26, 27] \rangle \text{election} \rangle$.) The example is continued in 4.13. \triangleleft

Lemma 4.6 (Elimination of Temporal Formulae) *If the time terms τ , τ_1 and τ_2 are constant over a time region i then the following equivalences hold:*

$$\begin{array}{lll}
\langle i \rangle (\tau \wedge \varphi) & \Leftrightarrow & \langle i \cap \tau_{\parallel}(i) \rangle \varphi \\
\langle i \rangle (\neg \tau \wedge \varphi) & \Leftrightarrow & \langle i \setminus \tau_{\parallel}(i) \rangle \varphi \\
\langle i \rangle (\tau_1 = \tau_2 \wedge \varphi) & \Leftrightarrow & \perp \quad \text{if } \tau_{1\parallel}(i) \neq \tau_{2\parallel}(i) \\
\langle i \rangle (\tau_1 = \tau_2 \wedge \varphi) & \Leftrightarrow & \langle i \rangle \varphi \quad \text{if } \tau_{1\parallel}(i) = \tau_{2\parallel}(i) \\
\langle i \rangle (\tau_1 \neq \tau_2 \wedge \varphi) & \Leftrightarrow & \perp \quad \text{if } \tau_{1\parallel}(i) = \tau_{2\parallel}(i) \\
\langle i \rangle (\tau_1 \neq \tau_2 \wedge \varphi) & \Leftrightarrow & \langle i \rangle \varphi \quad \text{if } \tau_{1\parallel}(i) \neq \tau_{2\parallel}(i) \\
[i] (\tau \vee \varphi) & \Leftrightarrow & [i \setminus \tau_{\parallel}(i)] \varphi \\
[i] (\neg \tau \vee \varphi) & \Leftrightarrow & [i \cap \tau_{\parallel}(i)] \varphi \\
[i] (\tau_1 = \tau_2 \vee \varphi) & \Leftrightarrow & \top \quad \text{if } \tau_{1\parallel}(i) = \tau_{2\parallel}(i) \\
[i] (\tau_1 = \tau_2 \vee \varphi) & \Leftrightarrow & [i] \varphi \quad \text{if } \tau_{1\parallel}(i) \neq \tau_{2\parallel}(i) \\
[i] (\tau_1 \neq \tau_2 \vee \varphi) & \Leftrightarrow & \top \quad \text{if } \tau_{1\parallel}(i) \neq \tau_{2\parallel}(i) \\
[i] (\tau_1 \neq \tau_2 \vee \varphi) & \Leftrightarrow & [i] \varphi \quad \text{if } \tau_{1\parallel}(i) = \tau_{2\parallel}(i).
\end{array}$$

The equivalences follow straightforwardly from the semantics of the operators and the temporal formulae. \triangleleft

Lemma 4.7 (Simplification Rules) *The following equivalences hold in \mathcal{CL}^I for all formulae φ and ψ :*

$$\begin{array}{lll}
[i][j]\varphi \Leftrightarrow [j]\varphi & [i](\varphi \wedge [j]\psi) \Leftrightarrow ([i]\varphi \wedge [j]\psi) & [i](\varphi \vee [j]\psi) \Leftrightarrow ([i]\varphi \vee [j]\psi) \\
[i]\langle j \rangle \varphi \Leftrightarrow \langle j \rangle \varphi & [i](\varphi \wedge \langle j \rangle \psi) \Leftrightarrow ([i]\varphi \wedge \langle j \rangle \psi) & [i](\varphi \vee \langle j \rangle \psi) \Leftrightarrow ([i]\varphi \vee \langle j \rangle \psi) \\
\langle i \rangle [j]\varphi \Leftrightarrow [j]\varphi & \langle i \rangle (\varphi \wedge [j]\psi) \Leftrightarrow (\langle i \rangle \varphi \wedge [j]\psi) & \langle i \rangle (\varphi \vee [j]\psi) \Leftrightarrow (\langle i \rangle \varphi \vee [j]\psi) \\
\langle i \rangle \langle j \rangle \varphi \Leftrightarrow \langle j \rangle \varphi & \langle i \rangle (\varphi \wedge \langle j \rangle \psi) \Leftrightarrow (\langle i \rangle \varphi \wedge \langle j \rangle \psi) & \langle i \rangle (\varphi \vee \langle j \rangle \psi) \Leftrightarrow (\langle i \rangle \varphi \vee \langle j \rangle \psi)
\end{array}$$

and

$$\langle \emptyset \rangle \varphi \Leftrightarrow \perp \quad [\emptyset] \varphi \Leftrightarrow \top.$$

\triangleleft

The proof is straightforward. It uses that the interpretation of the time region operators does not depend on the actual time point.

These equivalences can be used as rewrite rules from left to right. The result of rewriting a \mathcal{CL}^I -formula with these rules is a modal degree one normal form. There are no nested temporal operators any more.

The different time regions in the modal operators can interact with each other. For example $[[0, 100]]p$ and $\langle [50, 60] \rangle \neg p$ are contradictory because $[50, 60[$ is a subset of $[0, 100[$. $[[0, 100]]p$ and $\langle [50, 150] \rangle \neg p$ are not contradictory because $\neg p$ might be true between 100 and 150.

There are two options for dealing with these operators. One can define special inference rules that deal with the intervals in a suitable way, or one can make the different time regions independent. For example the two formulae $[[0, 100]]p$ and $\langle [50, 150] \rangle \neg p$ are equivalent to $[[0, 50]]p \wedge [[50, 100]]p$ and $\langle [50, 100] \rangle \neg p \vee \langle [100, 150] \rangle \neg p$. The second version is obtained by decomposing the time regions into their atomic components. Now the internal structure of the decomposed time regions does not matter any more. This is a well known

trick from Boolean Algebra theory. A finite set $\{i_1, \dots, i_n\}$ of sets generates a finite atomic Boolean Algebra by closing the set under intersection, union and complement. The atoms of this Boolean Algebra consist of the 2^n intersections $\pm i_1 \cap \dots \cap \pm i_n$ (where $\pm i$ means either i or its complement). For example the set $\{[0, 100[, [50, 150[\}$ has the four atoms $[0, 50[, [50, 100[, [100, 150[$ together with the complement of $[0, 200[$. In the next definition we formalize this idea.

Definition 4.8 (Atomic Decomposition for \mathcal{CL}^I)

Let φ be a $\mathcal{CL}_{\mathcal{I}d}^I$ -formula with $M \stackrel{\text{def}}{=} \{i_1, \dots, i_k\}$ being the time regions occurring in the time operators contained in φ . The closure of M under intersection, union and complement forms a finite atomic Boolean Algebra A . Let $\text{atoms}(A)$ be the atoms of A , and let $\text{atoms}(j) \stackrel{\text{def}}{=} \{a \in \text{atoms}(A) \mid a \subseteq j\}$ be the atomic decomposition of j .

We define the following rewrite system.

$$\begin{aligned} [i]\varphi &\rightarrow_A \bigwedge_{a \in \text{atoms}(i)} [a]\varphi \\ \langle i \rangle \varphi &\rightarrow_A \bigvee_{a \in \text{atoms}(i)} \langle a \rangle \varphi. \end{aligned}$$

The result of applying \rightarrow_A exhaustively to φ is called the atomic decomposition of φ . \triangleleft

Proposition 4.9 \rightarrow_A is an equivalence preserving transformation.

Proof: This follows immediately from the fact that for each time region i , $i = \bigcup_{a \in \text{atoms}(i)} a$, and from the semantics of the time region operators. \triangleleft

The transformation \rightarrow_I , which replaces the temporal operators by time region operators, the simplification rules (lemma 4.7) and the atomic decomposition \rightarrow_A translate Calendar Logic formulae into a language which is almost propositional, except for one level of time region operators.

The simple structure of the decomposed formulae allows for a translation into function free monadic predicate logic. One can exploit that the time regions occurring in the clauses are all disjoint. There is no interaction between them, and their internal structure is irrelevant. Therefore a time region a can be translated into a unary predicate $a(s)$. It is not necessary that the truth set of the predicate a is a time region. It can be an arbitrary set in an arbitrary domain. The only restriction is that for different time regions a and b , the disjointness condition

$$\forall x a(x) \Leftrightarrow \neg b(x)$$

is guaranteed. The translation rules for the box and diamond operator are:

$$\begin{aligned} [a](p_1 \vee \dots \vee p_n) &\text{ is translated into } \forall s a(s) \Rightarrow (p_1(s) \vee \dots \vee p_n(s)). \\ \langle a \rangle (p_1 \wedge \dots \wedge p_n) &\text{ is translated into } \exists s a(s) \wedge (p_1(s) \wedge \dots \wedge p_n(s)). \end{aligned}$$

Because there are no nested time region operators in the decomposed clauses, Skolemization of the existential quantifier yields a constant symbol. Therefore the clauses are translated into the function free fragment of predicate logic.

Satisfiability of formulae is decidable in this fragment (because the Herbrand universe is finite). Thus, this is actually a proof for the decidability of Calendar Logic.

We don't go into the technical details of this translation, because one can even do better. The decomposed clauses can in fact be translated into classical propositional logic. Each formula $[a]\varphi$ is replaced with a new predicate symbol $b_{a\varphi}$ and each formula $\langle a \rangle\psi$ is replaced with a new predicate symbol $d_{a\psi}$. The new predicate symbols are not independent of each other and of the old predicate symbols. Therefore some extra axioms are needed to correlate these different variable sets.

Truth of $b_{a\varphi}$ indicates that $[a]\varphi$ is true, and therefore φ must be true for all time points in a . This is made explicit by adding an implication (cf. (12) below)

$$b_{a\varphi} \Rightarrow R_a(\varphi).$$

$R_a(\varphi)$ consistently renames all predicate symbols p occurring in φ to p_a . (p_a indicates that p is true for all time points in a .)

Truth of $d_{a\psi}$ indicates that $\langle a \rangle\psi$ is true, and therefore ψ must be true for some time point in a . This is made explicit by adding an implication (cf. (11) below)

$$d_{a\psi} \Rightarrow R_{a\psi}(\psi).$$

$R_{a\psi}(\psi)$ consistently renames all predicate symbols p occurring in ψ to $p_{a\psi}$. ($p_{a\psi}$ indicates that p is true for some particular time point in a . This point may be different for each occurrence $\langle a \rangle\psi$ in different subformulae. Therefore $p_{a\psi}$ must be different for each ψ .)

Truth of $[a]\varphi$ further implies that φ is true in the particular time points introduced by the diamond operators. Therefore for each $b_{a\varphi}$ and $d_{a\psi}$ we add (cf. (14) below)

$$b_{a\varphi} \Rightarrow R_{a\psi}(\varphi).$$

The soundness and completeness of this translation relies on the fact that the atoms a of the atomic decomposition of the original time intervals are disjoint time regions. Box and diamond operators for different (atomic) time regions a never interact with each other.

Definition 4.10 (Translation into Propositional Logic)

We define a translation $PL0$ of decomposed formulae into classical propositional logic.

Let $R_a(\varphi)$ be a function that renames φ such that every predicate symbol p occurring in φ is replaced with a new predicate symbol p_a .

Let $R_{a\psi}(\varphi)$ be a function that renames φ such that every predicate symbol p occurring in φ is replaced with a new predicate symbol $p_{a\psi}$.

The translation function $PL0'$ introduces for each formula $[a]\varphi$ a new predicate symbol $b_{a\varphi}$ and for each formula $\langle a \rangle\psi$ a new predicate symbol $d_{a\psi}$:

$$[a]\varphi \rightarrow_{PL0} b_{a\varphi} \tag{10}$$

$$\langle a \rangle\psi \rightarrow_{PL0} d_{a\psi} \tag{11}$$

Let $PL0'(\eta)$ be the result of applying the rewrite relation \rightarrow_{PL0} exhaustively to η .

The translation function $PL0$ is defined

$$PL0(\eta) \stackrel{\text{def}}{=} PL0'(\eta) \wedge \bigwedge_{b_{a\varphi}} b_{a\varphi} \Rightarrow R_a(\varphi) \quad (12)$$

$$\wedge \bigwedge_{d_{a\psi}} d_{a\psi} \Rightarrow R_{a\psi}(\psi) \quad (13)$$

$$\wedge \bigwedge_{b_{a\varphi}, d_{a\psi}} b_{a\varphi} \Rightarrow R_{a\psi}(\varphi). \quad (14)$$

◁

Lemma 4.11 (Soundness and Completeness of the PL0 Translation)

A decomposed formula η is satisfiable in $\mathcal{CL}_{\mathcal{T}_U}$ if and only if the translated formula $PL0(\eta)$ is satisfiable in propositional logic.

Proof: (\Rightarrow) Suppose \mathfrak{S} satisfies η . We define a propositional interpretation \mathfrak{S}' and show that it satisfies the translated formula.

If $\mathfrak{S} \models [a]\varphi$ then $\mathfrak{S}'(b_{a\varphi}) \stackrel{\text{def}}{=} \text{true}$, otherwise *false*. (15)

If $\mathfrak{S} \models \langle a \rangle \psi$ then $\mathfrak{S}'(d_{a\psi}) \stackrel{\text{def}}{=} \text{true}$, otherwise *false*. (16)

If $\mathfrak{S} \models [a]\varphi$ then $\mathfrak{S}, t \models \varphi$ for all $t \in a$. We choose one $t_0 \in a$ and define for each propositional variable p occurring in φ :

if $\mathfrak{S}, t_0 \models p$ then $\mathfrak{S}'(p_a) = \text{true}$, otherwise *false*. (17)

If $\mathfrak{S} \models \langle a \rangle \psi$ then $\mathfrak{S}, t \models \psi$ for some $t \in a$. We define for each propositional variable p occurring in $\{\psi, \varphi_1, \dots, \varphi_m\}$ where $\{\varphi_1, \dots, \varphi_m\} \stackrel{\text{def}}{=} \{\varphi \mid \mathfrak{S} \models [a]\varphi\}$:

if $\mathfrak{S}, t \models p$ then $\mathfrak{S}'(p_{a\psi}) = \text{true}$, otherwise *false*. (18)

This guarantees that \mathfrak{S}' gives the same truth values to the variables $b_{a\varphi}$ and $d_{a\psi}$ as \mathfrak{S} does to $[a]\varphi$. and $\langle a \rangle \psi$. Therefore $PL0'(\eta)$ is satisfied by \mathfrak{S}' . (17) guarantees that if $\mathfrak{S}'(b_{a\varphi}) = \text{true}$ then \mathfrak{S}' satisfies $R_a(\varphi)$, i.e. \mathfrak{S}' satisfies (12). (18) guarantees that

- if $\mathfrak{S}'(d_{a\psi}) = \text{true}$ then \mathfrak{S}' satisfies $R_{a\psi}(\psi)$, i.e. \mathfrak{S}' satisfies (13) and
- if $\mathfrak{S}'(b_{a\varphi}) = \text{true}$ then \mathfrak{S}' satisfies $R_{a\psi}(\varphi)$, i.e. \mathfrak{S}' satisfies (14).

Thus, \mathfrak{S}' satisfies $PL0(\eta)$.

(\Leftarrow) The proof of this direction exploits that the time regions occurring in the decomposed clauses are all disjoint, and that the reference time line is a dense time structure. In integer like time structures it would not work because there might not be enough time points in a time region to satisfy all relevant diamond operators.

Suppose a propositional interpretation \mathfrak{S}' satisfies $PL0(\eta)$. We construct a temporal interpretation \mathfrak{S} for η .

Suppose $\mathfrak{S}'(b_{a\varphi}) = \text{true}$. In this case $\mathfrak{S}'(R_a(\varphi)) = \text{true}$ (12). For each $p_a \in R_a(\varphi)$ we define: if $\mathfrak{S}'(p_a) = \text{true}$ the $\mathfrak{S}, t \models p$ for each $t \in a$. (This is possible without getting conflicts for some $t \in a$ because all the time regions a are

disjoint.) This definition together with $\mathfrak{S}'(R_a(\varphi)) = true$ ensures $\mathfrak{S}, t \models \varphi$ for all $t \in a$. Thus, $\mathfrak{S} \models [a]\varphi_k$.

Suppose $\mathfrak{S}'(d_{a\psi}) = true$. In this case $\mathfrak{S}'(R_{a\psi}(\psi)) = true$ (13), and for all $b_{a\varphi}$ where $\mathfrak{S}'(b_{a\varphi}) = true$ we have $\mathfrak{S}'(R_{a\psi}(\varphi)) = true$ as well (14). That means $\mathfrak{S}'(R_{a\psi}(\psi) \wedge R_{a\psi}(\varphi_1) \wedge \dots \wedge R_{a\psi}(\varphi_m)) = true$ where $\{\varphi_1, \dots, \varphi_m\} = \{\varphi \mid \mathfrak{S}'(b_{a\varphi}) = true\}$.

For each $d_{a\psi}$ we choose a different time point $t_{a\psi} \in a$ as the time point where ψ is to be made true. This is possible because the reference time line is a dense structure. Therefore there are always enough time points to choose from. When constructing the interpretation of ψ in $t_{a\psi}$, we exploit (14) to make sure that for all formulae $[a]\varphi$ which are satisfied in \mathfrak{S} by the above construction, φ is satisfied in $t_{a\psi}$. For each $p_{a\psi}$ occurring in $R_{a\psi}(\psi) \wedge R_{a\psi}(\varphi_1) \wedge \dots \wedge R_{a\psi}(\varphi_m)$ we define: if $\mathfrak{S}'(p_{a\psi}) = true$ then $\mathfrak{S}, t_{a\psi} \models p$. This, together with $\mathfrak{S}'(R_{a\psi}(\psi) \wedge R_{a\psi}(\varphi_1) \wedge \dots \wedge R_{a\psi}(\varphi_m)) = true$ guarantees $\mathfrak{S}, t_{a\psi} \models \psi \wedge \varphi_1 \wedge \dots \wedge \varphi_m$. Thus, $\mathfrak{S} \models \langle a \rangle \psi$. This means, $b_{a\varphi}$ and $d_{a\psi}$ have the same truth value in \mathfrak{S}' as $[a]\varphi$ and $\langle a \rangle \psi$ have in \mathfrak{S} . Thus, $\mathfrak{S} \models \eta$. \triangleleft

Theorem 4.12 *Satisfiability of $\mathcal{CL}_{\mathcal{T}_U}$ -formulae is decidable.*

Proof: The translation of $\mathcal{CL}_{\mathcal{T}_U}$ -formulae into propositional logic is satisfiability and unsatisfiability preserving. Therefore theoremhood of φ can be decided by deciding theoremhood of the translated formula, and this is decidable in propositional logic. \triangleleft

Example 4.13 (for the translation into propositional logic)

In example 4.5 the statement

$$[2000, year] : \langle January(x_{year}) \rangle (Friday(y_{week}) \Rightarrow election)$$

was reduced to

$$[[0, 366]] (\langle [5, 6] \rangle election \vee \langle [12, 13] \rangle election \vee \langle [19, 20] \rangle election \vee \langle [26, 27] \rangle election)$$

The outermost box operator is superfluous (lemma 4.7) and can be dropped. The remaining time regions are all disjoint. Therefore they themselves are the atoms in the atomic decomposition. The propositional version is therefore simply

$$\begin{aligned} & (d_{[5,6]election} \vee d_{[12,13]election} \vee d_{[19,20]election} \vee d_{[26,27]election}) \wedge \\ & d_{[5,6]election} \Rightarrow election_{[5,6]} \wedge \\ & d_{[12,13]election} \Rightarrow election_{[12,13]} \wedge \\ & d_{[19,20]election} \Rightarrow election_{[19,20]} \wedge \\ & d_{[26,27]election} \Rightarrow election_{[26,27]}. \end{aligned}$$

5 A Tableau System for Calendar Logic

The translation of $\mathcal{CL}_{\mathcal{T}_U}$ -formulae into propositional logic may increase the size of the formulae exponentially. Therefore it could be more efficient for a calculus to work on the original and not on the translated syntax. As an alternative to the translation method we provide a tableau system for deciding the satisfiability of $\mathcal{CL}_{\mathcal{T}_U}$ -formulae.

A tableau system makes explicit the information contained implicitly in the formulae and does a systematic case analysis for disjunctive information. Starting with a formula or a list of formulae, the tableau system operates by choosing a tableau rule and some formulae to which the rule is applicable and modifies the formulae in the way the rule specifies. Disjunctive rules split the whole tableaux system at that stage into different branches corresponding to the different cases to be investigated. The layout of the tableau is therefore tree like. The tableau rules can operate on different branches simultaneously. A branch is *closed* as soon as a contradiction is found. This is checked with some extra rules, the *closing conditions*. A branch which is not closed, but to which no rule is applicable any more, is an *open branch*. An open branch describes a model for the original formula. If all branches close then the initial formula set is not satisfiable.

Tableau entries are usually formulae of the given logic labelled with some extra information which is related to the semantics of the logic. In our case the labels are time regions, or *reference time symbols*. An entry $i : \varphi$ where i is an interval expresses that the formula φ is true at all points in the interval. An entry $t : \varphi$ where t is a reference time symbol expresses that φ is true at some point in time, and t is the name of this point (t is not a number). In all cases these reference time symbols t are constrained by an entry $t \in i$ where i is an interval.

A fourth kind of tableau entries are tuples $(i : \varphi_1 \vee \dots \vee \varphi_n \mid t_1, \dots, t_k)$. They are used to control the processing of disjunctions. It means that $i : \varphi_1 \vee \dots \vee \varphi_n$ is supposed to hold, and in the current branch the symbols t_i are used for testing the consistency of $t_i : \varphi_i$.

Definition 5.1 (Tableau Entries) *A tableau entry can be*

- i) $t : \varphi$ (*point statement or point entry*) where t is a symbol denoting a reference time point (a reference time symbol) and φ is a $\mathcal{CL}_{\mathcal{T}_U}$ formula, or
- ii) $i : \varphi$ (*interval statement or interval entry*) where i is time region and φ is a $\mathcal{CL}_{\mathcal{T}_U}$ formula, or
- iii) $t \in i$ (*membership statement or constraint for t*) where t is a reference time symbol and i is a time region, or
- iv) $(i : \varphi_1 \vee \dots \vee \varphi_n \mid t_1, \dots, t_k)$ (\vee -control entry) where $i : \varphi_1 \vee \dots \vee \varphi_n$ is an interval statement and t_1, \dots, t_k are reference time symbols.

We shall always use the letters t and i to distinguish reference time symbols from time regions. ◁

Definition 5.2 (Semantics of Tableau Entries) A tableau interpretation $\mathcal{E} = (\mathfrak{S}, \mathbb{T})$ consists of a $\mathcal{CL}_{\mathcal{T}_U}$ -interpretation $\mathfrak{S} = (\mathfrak{S}_{\mathcal{T}_U}, \mathcal{P})$ (definition 3.2) and a component \mathbb{T} mapping reference time symbols to reference time points.

$$\begin{aligned} \mathcal{E} \models t : \varphi & \quad \text{iff} \quad \mathfrak{S}, \mathbb{T}(t) \models \varphi \\ \mathcal{E} \models i : \varphi & \quad \text{iff} \quad \mathfrak{S}, s \models \varphi \text{ for all } s \in i \\ \mathcal{E} \models t \in i & \quad \text{iff} \quad \mathbb{T}(t) \in i \\ \mathcal{E} \models [n, U] : \varphi & \quad \text{iff} \quad \mathfrak{S}, s \models \varphi \text{ for all } s \in U_{\parallel}(n) \\ \mathcal{E} \models \langle n, U \rangle : \varphi & \quad \text{iff} \quad \mathfrak{S}, s \models \varphi \text{ for some } s \in U_{\parallel}(n). \end{aligned}$$

If $\mathcal{E} \models \varphi$, we say \mathcal{E} satisfies (is a model of) φ . ◁

Definition 5.3 ($\mathcal{CL}_{\mathcal{T}_U}$ -statements \rightarrow tableau entries) The following replacement rules turn a set of $\mathcal{CL}_{\mathcal{T}_U}$ -statements (definition 3.1) into tableau entries:

$$\begin{aligned} \langle n, U \rangle : \varphi & \rightarrow \begin{array}{l} t : \varphi \\ t \in U_{\parallel}(n) \end{array} \\ [n, U] : \varphi & \rightarrow U_{\parallel}(n) : \varphi. \end{aligned} \quad \triangleleft$$

It is straightforward to prove that a set of $\mathcal{CL}_{\mathcal{T}_U}$ -statements has a model if and only if the corresponding transformed set of tableau entries has a model.

The tableau rules below exploit the semantics of the $\mathcal{CL}_{\mathcal{T}_U}$ -connectives in most cases in a quite natural way. There are, however, a few tricky parts. The connective which is most difficult to handle turned out to be the disjunction in connection with intervals. A tableau entry $i : \varphi_1 \vee \dots \vee \varphi_n$ expresses that at each point in the interval i one of the φ_k is true. Since intervals consist of infinitely many points, an exhaustive case analysis for all points is not possible. Instead of this, these entries are processed by triggering a computation which figures out the sub-intervals $i_k \subseteq i$ for which it is consistent to assume that φ_k is true. If it turns out that the union of all these sub-intervals is not i itself, then a closing condition applies because points in i have been detected for which none of the φ_k can be true.

Definition 5.4 (Tableau Rules) The tableau rules for checking the consistency of a set of $\mathcal{CL}_{\mathcal{T}_U}$ statements consists of extension rules, transformation rules and closing conditions. The extension rules just add new facts or new branches in the usual tableau style. The transformation rules operate destructively at certain parts of the tableau. The closing conditions are to be checked whenever something has been changed in the tableau. As soon as they detect an inconsistency they declare a branch to be closed.

The tableau rules below are displayed in a quite self explaining style. $\text{left} \rightarrow \text{right}$ indicates a transformation rule which replaces the left part with the right part. $\text{left} \rightarrow \text{right}_1 \mid \dots \mid \text{right}_n$ means deleting left and splitting the current branch into n new branches with right_k in the k -th branch instead of left. $\text{left} \rightarrow \emptyset$ means simply deleting left.

Extension rules are displayed as

$$\frac{\begin{array}{c} \varphi_1 \\ \vdots \\ \varphi_n \end{array}}{\psi_1 \mid \dots \mid \psi_k.}$$

This means that if there are terms matching $\varphi_1 \dots \varphi_n$ in the branch then split the branch into k sub-branches (k may of course be 1) and add ψ_i to the i -th branch. Most of the rules come with additional restrictions for their applicability.

• **Inference Rules**

$$(\wedge_{\exists}) \quad t : \varphi_1 \wedge \varphi_2 \quad \rightarrow_{\wedge_{\exists}} \quad \begin{array}{l} t : \varphi_1 \\ t : \varphi_2. \end{array} \quad (t \text{ is a reference time symbol.})$$

$$(\wedge_{\forall}) \quad i : \varphi_1 \wedge \varphi_2 \quad \rightarrow_{\wedge_{\forall}} \quad \begin{array}{l} i : \varphi_1 \\ i : \varphi_2. \end{array} \quad (i \text{ is a time region}).$$

$$(\vee_{\exists}) \quad t : \varphi_1 \vee \varphi_2 \quad \rightarrow_{\vee_{\exists}} \quad t : \varphi_1 \mid t : \varphi_2.$$

$$(\vee_{\forall}) \quad \begin{array}{c} i : \varphi_1 \vee \varphi_2 \\ t \in j \end{array} \quad \frac{}{t \in (i \cap j) \mid t \in (i \cap j) \mid t \in (j \setminus i)} \quad \begin{array}{l} t : \varphi_1 \\ t : \varphi_2 \end{array}$$

if $i \cap j \neq \emptyset$ and $t : \varphi_1$ and $t : \varphi_2$ are not in the branch.

$$(\vee_{\forall}^1) \quad i : \varphi_1 \vee \dots \vee \varphi_n \quad \rightarrow_{\vee_{\forall}^1} \quad \begin{array}{l} i : \varphi_1 \vee \dots \vee \varphi_n \\ (i : \varphi_2 \vee \dots \vee \varphi_n \mid t) \\ t : \varphi_1 \\ t \in i \end{array} \quad \left| \quad i : \varphi_2 \vee \dots \vee \varphi_n \right.$$

if *i*) none of the φ_i are time terms, *ii*) none of the φ_i is itself a disjunction, *iii*) the (\vee_{\forall}^k) -rule is not applicable, and *iv*) this rule has not previously been applied to $i : \varphi_1 \vee \dots \vee \varphi_n$ in the branch. t is a fresh new reference time symbol.⁵

$$(\vee_{\forall}^k) \quad \begin{array}{c} i : \varphi_1 \vee \dots \vee \varphi_n \\ (i : \varphi_k \vee \dots \vee \varphi_n \mid t_1, \dots, t_l) \end{array} \quad \rightarrow_{\vee_{\forall}^k} \quad \begin{array}{l} i : \varphi_1 \vee \dots \vee \varphi_n \\ (i : \varphi_{k+1} \vee \dots \vee \varphi_n \mid t_1, \dots, t_l, t) \\ t : \varphi_{k+1} \\ t \in i \end{array} \quad \left| \quad i : \varphi_1 \vee \dots \vee \varphi_k \vee \varphi_{k+2} \dots \vee \varphi_n \right.$$

⁵This rule initiates the processing of disjunctions. The left branch introduces a new reference time symbol t which is initially constrained to i . The constraint for t may be narrowed down by successive rule applications. If a contradiction is detected then φ_1 is false for all elements in i and the right branch must be processed. $(i : \varphi_2 \vee \dots \vee \varphi_n \mid t)$ is control information passed on to the (\vee_{\forall}^k) -rule and the (\vee_{\forall}) -closing condition.

if this rule has not previously been applied to $(i : \varphi_k \vee \dots \vee \varphi_n \mid t_1, \dots, t_l)$ and if no other rule except (\vee_{\forall}^1) is applicable. t is again a fresh new reference time symbol. The right branch is superfluous if $k = n$.⁶

$$(\diamond_{\exists}) \quad \begin{array}{l} t \in i \\ t : \langle \tau \rangle \varphi \end{array} \rightarrow_{\diamond_{\exists}} \quad \begin{array}{c} t \in i_1 \quad \left| \dots \right| \quad t \in i_n \\ t_1 : \varphi \quad \left| \dots \right| \quad t_n : \varphi \\ t_1 \in \tau_{\llbracket i_1} \quad \left| \dots \right| \quad t_n \in \tau_{\llbracket i_n} \end{array}$$

where $\{i_1, \dots, i_n\} = \{j \in \delta(i, \tau) \mid \tau_{\llbracket j} \neq \emptyset\}$ and the t_k are fresh new reference time symbols.

$$(\diamond_{\forall}) \quad i : \langle \tau \rangle \varphi \rightarrow_{\diamond_{\forall}} \quad \begin{array}{c} t_1 : \varphi \\ t_1 \in \tau_{\llbracket i_1} \\ \vdots \\ t_n : \varphi \\ t_n \in \tau_{\llbracket i_n} \end{array}$$

where $\{i_1, \dots, i_n\} = \{j \in \delta(i, \tau) \mid \tau_{\llbracket j} \neq \emptyset\}$ and the t_k are fresh new reference time symbols.

$$(\square_{\exists}) \quad \begin{array}{l} t \in i \\ t : [\tau] \varphi \end{array} \rightarrow_{\square_{\exists}} \quad \begin{array}{c} t \in i_1 \quad \left| \dots \right| \quad t \in i_n \\ \tau_{\llbracket i_1} : \varphi \quad \left| \dots \right| \quad \tau_{\llbracket i_n} : \varphi \end{array}$$

where $\{i_1, \dots, i_n\} = \{j \in \delta(i, \tau) \mid \tau_{\llbracket j} \neq \emptyset\}$ and this set is not empty.

$$(\square_{\forall}) \quad i : [\tau] \varphi \rightarrow_{\square_{\forall}} \quad \tau_{\llbracket i} : \varphi$$

if $\tau_{\llbracket i} \neq \emptyset$.

$$(\setminus) \quad \begin{array}{l} i : p \\ t : \neg p \\ t \in j \end{array} \rightarrow_{\setminus} \quad \begin{array}{l} i : p \\ t : \neg p \\ t \in (j \setminus i) \end{array} \quad \text{and} \quad \begin{array}{l} i : \neg p \\ t : p \\ t \in j \end{array} \rightarrow_{\setminus} \quad \begin{array}{l} i : \neg p \\ t : p \\ t \in (j \setminus i) \end{array}$$

if $i \cap j \neq \emptyset$.

$$(\cap) \quad \begin{array}{l} t \in i \\ t \in j \end{array} \rightarrow_{\cap} \quad t \in (i \cap j)$$

$$(\tau_+) \quad \begin{array}{l} t : \tau \\ t \in i \end{array} \rightarrow_{\tau_+} \quad t \in (i \cap \tau_{\llbracket i})$$

$$(\tau_-) \quad \begin{array}{l} t : \neg \tau \\ t \in i \end{array} \rightarrow_{\tau_-} \quad t \in (i \setminus \tau_{\llbracket i})$$

⁶This rule continues the processing of disjunctions initiated by the (\vee_{\forall}^1) -rule. It is applied only when no other rule is applicable any more and the branch would normally be declared open. It triggers the investigation to figure out the sub-interval of i where it is consistent to assume that φ_{k+1} holds. This is checked with a fresh reference time symbol t with initial constraint i . The constraint for i may subsequently be narrowed down. If a contradiction occurs then φ_{k+1} must be false in the entire interval i , and the right branch must be processed. The way we presented this rule suggests that the (\vee_{\forall}^1) -rule is applied again to $i : \varphi_1 \vee \dots \vee \varphi_k \vee \varphi_{k+2} \vee \dots \vee \varphi_n$, processing $\varphi_1, \dots, \varphi_k$ anew. With an appropriate bookkeeping this can of course be avoided.

$$\begin{aligned}
(\tau_{\vee}^+) \quad & i : \tau \vee \varphi \rightarrow_{\tau_{\vee}^+} i \setminus \pi_{\parallel}(i) : \varphi \\
& \text{where } \tau \in \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}. \\
(\tau_{\vee}^-) \quad & i : \neg\tau \vee \varphi \rightarrow_{\tau_{\vee}^-} \pi_{\parallel}(i) : \varphi \\
& \text{if } \tau \in \mathcal{CL}_{\mathcal{T}_{\mathcal{U}}}. \\
(\tau_{\top}) \quad & i : \tau \rightarrow_{\tau_{\top}} \emptyset \text{ if } \tau \in \mathcal{T}_{\mathcal{U}} \text{ and } i \subseteq \tau_{\parallel}(i). \\
(\tau_{\neg\top}) \quad & i : \neg\tau \rightarrow_{\tau_{\neg\top}} \emptyset \text{ if } \tau \in \mathcal{T}_{\mathcal{U}} \text{ and } i \cap \tau_{\parallel}(i) = \emptyset. \\
(\subseteq_{\vee}) \quad & i : \tau_1 \subseteq \tau_2 \rightarrow_{\subseteq_{\vee}} \emptyset \\
& \text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and for all } j \in \delta(i, \{\tau_1, \tau_2\}): \tau_{1\parallel}(j) \subseteq \tau_{2\parallel}(j) \\
(\not\subseteq_{\vee}) \quad & i : \tau_1 \not\subseteq \tau_2 \rightarrow_{\not\subseteq_{\vee}} \emptyset \\
& \text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and for all } j \in \delta(i, \{\tau_1, \tau_2\}): \tau_{1\parallel}(j) \neq \tau_{2\parallel}(j) \\
(\subseteq_{\exists}) \quad & \frac{t : \tau_1 \subseteq \tau_2}{t \in i} \rightarrow_{\subseteq_{\exists}} \frac{t : \tau_1 \subseteq \tau_2}{t \in (i \setminus j)} \\
& \text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and } j \in \delta(i, \{\tau_1, \tau_2\}) \text{ with } \tau_{1\parallel}(j) \neq \tau_{2\parallel}(j) \\
(\not\subseteq_{\exists}) \quad & \frac{t : \tau_1 \not\subseteq \tau_2}{t \in i} \rightarrow_{\not\subseteq_{\exists}} \frac{t : \tau_1 \neq \tau_2}{t \in (i \setminus j)} \\
& \text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and } j \in \delta(i, \{\tau_1, \tau_2\}) \text{ with } \tau_{1\parallel}(j) = \tau_{2\parallel}(j).
\end{aligned}$$

• **Closing Conditions**

$$\begin{aligned}
(\emptyset) \quad & \frac{t \in \emptyset}{\text{closed}} \\
(\neg) \quad & \frac{t : p}{t : \neg p} \\
& \text{closed} \\
(\cap_{\neg}) \quad & \frac{i : p \quad \text{if } i \cap j \neq \emptyset.}{j : \neg p} \\
& \text{closed} \\
(\diamond) \quad & \frac{t \in i \quad \text{if } \pi_{\parallel}(i) = \emptyset.}{t : \langle \tau \rangle \varphi} \\
& \text{closed} \\
(\tau_{\vee}) \quad & \frac{i : \tau}{\text{closed}} \quad \text{where } \tau \in \mathcal{T}_{\mathcal{U}} \text{ and } i \not\subseteq \tau_{\parallel}(i). \\
(\tau_{\neg\vee}) \quad & \frac{i : \neg\tau}{\text{closed}} \quad \text{where } \tau \in \mathcal{T}_{\mathcal{U}} \text{ and } i \cap \tau_{\parallel}(i) \neq \emptyset. \\
(\vee_{\vee}) \quad & \frac{i : \varphi_1 \vee \dots \vee \varphi_n}{(i : | t_1, \dots, t_l)} \quad \text{7} \\
& \frac{t_1 \in i_1 \quad \text{if } i_1 \cup \dots \cup i_n \neq i.}{\vdots} \\
& \frac{t_l \in i_n}{\text{closed}}
\end{aligned}$$

⁷This rule finishes the processing of disjunctions started with the (\vee_{\vee}^1) -rule and continued

$$\begin{array}{l}
(\subseteq_{\perp}) \frac{i : \tau_1 \subseteq \tau_2}{\text{closed}} \\
\text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and for some } j \in \delta(i, \{\tau_1, \tau_2\}): \tau_{1\parallel}(j) \not\subseteq \tau_{2\parallel}(j) \\
(\not\subseteq_{\perp}) \frac{i : \tau_1 \not\subseteq \tau_2}{\text{closed}} \\
\text{if } \tau_1 \in \mathcal{T}_{\mathcal{U}}, \tau_2 \in \mathcal{T}_{\mathcal{U}}, \text{ and for some } j \in \delta(i, \{\tau_1, \tau_2\}): \tau_{1\parallel}(j) \subseteq \tau_{2\parallel}(j).
\end{array}$$

If A is a set of tableau entries, let $\text{Tab}(A)$ be the set of open branches resulting from exhaustive application of the tableau rules. \triangleleft

Example 5.5 (for a Tableaux Proof) We illustrate the whole procedure with a very simple example. It nevertheless shows most of the aspects of our system. Moreover, it shows that, although there is disjunctive information, the tableau algorithm performs in a very straightforward deterministic way.

Suppose we have the following statements:

- i) last week it was rainy or foggy;
- ii) from Monday till Wednesday last week it was not rainy;
- iii) from Wednesday till Sunday last week it was not foggy.

This is clearly unsatisfiable because on Wednesday it was neither rainy nor foggy.

First of all, we have to fix a reference time line. To simplify the numbers, point 0 of the reference time line is assumed to be at the beginning of the last week. The time unit day is arranged at the reference time line such that day 0 (Monday last week) covers the interval $[0, 10[$, day 1 (Tuesday last week) covers the interval $[10, 20[$, etc. Time unit week is arranged that week 0 (last week) covers the interval $[0, 70[$ etc. The current point in time (now) is supposed to be at reference time point, say, 105, which is Thursday this week, day number 10.

The above statements are encoded in Calendar Logic as follows:

- (1) $\langle 10, \text{day} \rangle : [y_{\text{week}} - 1](r \vee f)$
- (2) $\langle 10, \text{day} \rangle : \langle y_{\text{week}} - 1 \rangle [\{Mo(x_{\text{week}}), Tu(x_{\text{week}}), We(x_{\text{week}})\}] \neg r$
- (3) $\langle 10, \text{day} \rangle : \langle y_{\text{week}} - 1 \rangle [\{We(x_{\text{week}}), \dots, So(x_{\text{week}})\}] \neg f$

$Mo(x_{\text{week}})$ is specified as the first day in the week, $Tu(x_{\text{week}})$ as the second day in the week etc.

First of all the statements (1), (2) and (3) have to be turned into tableau entries using the replacement rules of Definition 5.3. The result is

- (4) $t_1 : [y_{\text{week}} - 1](r \vee f)$
- (5) $t_1 \in [100, 110[$
- (6) $t_2 : \langle y_{\text{week}} - 1 \rangle [\{Mo(x_{\text{week}}), Tu(x_{\text{week}}), We(x_{\text{week}})\}] \neg r$
- (7) $t_2 \in [100, 110[$
- (8) $t_3 : \langle y_{\text{week}} - 1 \rangle [\{We(x_{\text{week}}), \dots, So(x_{\text{week}})\}] \neg f$
- (9) $t_3 \in [100, 110[$.

with the (\forall_{\neq}^k) -rule.

Processing the formulae top down we start by applying the (\Box_{\exists}) -rule to entry (4) and (5). $\delta([100, 110[, x_{week} - 1) = [100, 110[$ because the week co-ordinate is 1 for all time points between 100 and 110 (this week is week 1). $(x_{week} - 1)([100, 110[) = [0, 70[$. (week 0 ranges from time point 0 till time point 70). The result of the (\Box_{\exists}) -rule application is therefore

$$(10) \quad [0, 70[: (r \vee f).$$

In a very similar way we apply the (\Diamond_{\exists}) -rule to (6) and (7) and to (8) and (9):

$$(11) \quad t_4 : [\{Mo(x_{week}), Tu(x_{week}), We(x_{week})\}] \neg r$$

$$(12) \quad t_4 \in [0, 70[$$

$$(13) \quad t_5 : [\{We(x_{week}), \dots, So(x_{week})\}] \neg f$$

$$(14) \quad t_5 \in [0, 70[.$$

Using the definition of $Mo(x_{week})$ etc., we apply the (\Box_{\exists}) -rule to (11), (12) and to (13), (14). The result is:

$$(15) \quad [0, 30[: \neg r$$

$$(16) \quad [20, 70[: \neg f.$$

The next step is to initiate the investigation of the disjunction (10) with the (\vee_{\forall}^1) -rule.

We get the two branches:

$$\begin{array}{l|l} (10) \quad [0, 70[: (r \vee f) & (20) \quad [0, 70[: f \\ (17) \quad ([0, 70[: f, t_6) & \\ (18) \quad t_6 : r & \\ (19) \quad t_6 \in [0, 70[. & \end{array}$$

The right branch (20) is immediately closed by the (\cap_{\neg}) -closing condition (together with (16)). Using (15) and (18) in the left branch, (19) is narrowed down by the (\wedge) -rule to

$$(21) \quad t_6 \in [30, 70[$$

No further rule except (\vee_{\forall}^1) is applicable. This rule yields

$$(10) \quad [0, 70[: (r \vee f)$$

$$(22) \quad ([0, 70[:, t_6, t_7)$$

$$(23) \quad t_7 : f$$

$$(24) \quad t_7 \in [0, 70[.$$

Using (16) and (23), (24) is narrowed down by the (\wedge) -rule to

$$(25) \quad t_7 \in [0, 20[.$$

The closing condition (\vee_{\forall}) now applies to (22), (21) and (25) because $([0, 20[\cup [30, 70[) \neq [0, 70[$. The missing part where $r \vee f$ cannot hold is $[20, 30[$, which was Wednesday last week, as expected.

6 Termination, Soundness and Completeness

Theorem 6.1 (Termination of the Tableau System) *The application of the tableau rules (definition 5.4) to a finite set of tableau entries eventually terminates.*

Proof: Termination is proved by showing that the tableau rules can be applied only finitely many times. First of all we need to show that only finitely many reference time symbols can be generated by the tableau rules. The key observation is that all rules which generate new reference time symbols, and these are the (\forall_{\forall}^1) -rule, the (\forall_{\forall}^k) -rule, the (\diamond_{\exists}) -rule and the (\Box_{\exists}) -rule, generate tableau entries $t : \varphi$ where the φ is smaller in size than the formulae in the statements to which the rule is applied. Furthermore there are no rules at all which generate formulae which are larger in size than the formulae to which the rule is applied. Therefore the generation of new reference time symbols is bound to terminate eventually.

Since the maximal number of reference time symbols ever generated in one branch is finite, the (\forall_{\forall}) -rule can be applied only finitely often.

The (\forall_{\forall}^1) -rule and the (\forall_{\forall}^k) -rule and together process each disjunction only once and therefore terminate once a disjunction is ‘worked off’.

All the other rules destructively replace complex entries with entries which are strictly smaller in size than the original formulae to which they are applied. Therefore their application terminates as well. \triangleleft

Theorem 6.2 (Soundness) *If a finite set A of Calendar Logic statements is satisfiable then $\text{Tab}(A)$ has some open branches.*

Proof: Soundness is proved by induction on the number of rule applications. To this end we have to show that if a tableau interpretation $\mathcal{E} = (\mathfrak{S}, \mathbb{T})$ satisfies a branch B of a tableau before the application of one of the tableau rules 5.4 then either \mathcal{E} or some modification \mathcal{E}' satisfies one of the successor branches of the rule application.

We check each rule separately.

(\wedge_{\exists}) -rule, (\wedge_{\forall}) -rule, (\forall_{\exists}) -rule: obvious.

(\forall_{\forall}) -rule: If $\mathcal{E} \models i : \varphi_1 \vee \varphi_2$ and $\mathcal{E} \models t \in j$ and $i \cap j \neq \emptyset$ then either $\mathbb{T}(t) \in (i \setminus j)$, in which case $\mathcal{E} \models t : \varphi_1$ or $\mathcal{E} \models t : \varphi_2$, or $\mathbb{T}(t) \in (j \setminus i)$, which means $\mathcal{E} \models t \in (j \setminus i)$.

(\forall_{\forall}^1) -rule: If $\varphi_1 \vee \dots \vee \varphi_n$ holds for all $t \in i$ then either φ_1 holds for some $t \in i$, in which case assigning $\mathbb{T}(t) \stackrel{\text{def}}{=} s$ satisfies the left branch, or φ_1 holds for no $t \in i$, in which case $\varphi_2 \vee \dots \vee \varphi_n$ must hold for all $t \in i$. Therefore one of the successor branches is satisfied.

(\forall_{\forall}^k) -rule: The argument is analogous to the argument in the previous case.

(\diamond_{\exists})-rule: If $\mathcal{E} \models t \in i$ then $\mathbb{T}(t) \in i$. Since $\delta(i, \tau)$ partitions i , $\mathbb{T}(t) \in i_k$ for some $i_k \in \delta(i, \tau)$, i.e. $\mathcal{E} \models t \in i_k$. Furthermore, since $\mathcal{E} \models t : \langle \tau \rangle \varphi$, there is some $s \in \tau_{\parallel}(t) (= \tau_{\parallel}(i_k))$ with $\mathfrak{S}, s \models \varphi$. Assigning $\mathbb{T}(t_k) \stackrel{\text{def}}{=} s$ satisfies $t_k : \varphi$.

(\diamond_{\forall})-rule: If $\mathcal{E} \models i : \langle \tau \rangle \varphi$ then $\mathfrak{S}, t \models \langle \tau \rangle \varphi$ for all $s \in i$. That means, for each $s \in i$ there is some $s' \in \tau_{\parallel}(s)$ with $\mathcal{E}, s' \models \varphi$. This means in particular for each $s_k \in i_k (\subseteq i)$ there is some $s'_k \in \tau_{\parallel}(s_k)$ with $\mathcal{E}, s'_k \models \varphi$ for $1 \leq k \leq n$. Assigning $\mathbb{T}(t_k) \stackrel{\text{def}}{=} s'_k$ therefore satisfies the new entries.

(\square_{\exists})-rule: The argument is analogous to the argument for the (\diamond_{\exists})-rule.

(\square_{\forall})-rule: If $\mathcal{E} \models i : [\tau] \varphi$ then for all $s \in i$, $\mathfrak{S}, s \models [\tau] \varphi$, i.e. for all $s' \in \tau_{\parallel}(s)$, $\mathfrak{S}, s' \models \varphi$. This means $\mathcal{E} \models \tau_{\parallel}(i) : \varphi$.

(\backslash)-rule and all the remaining rules are straightforward to check. \triangleleft

Lemma 6.3 (Closed Branches) *A closed branch in a tableau generated with the rules 5.4 cannot have a model.*

Proof: A branch B is closed if one of the closing conditions apply. For the closing conditions (\emptyset), (\neg), (\cap_{\neg}) and (\diamond) it is obvious that the branch cannot have a model.

(τ_{\forall})-condition: $\forall t \in i : t \in \tau_{\parallel}(t)$ is impossible if $i \not\subseteq \tau_{\parallel}(i)$.

($\tau_{\neg\forall}$)-condition: $\forall t \in i : t \notin \tau_{\parallel}(t)$ is equivalent to $\neg \exists t \in i : t \in \tau_{\parallel}(t)$ which is impossible if $i \cap \tau_{\parallel}(i) \neq \emptyset$.

(\vee_{\forall})-condition: This condition is triggered in a situation where for each $k \in \{1, \dots, n\}$ ($i \setminus i_k$) : φ_k has been refuted, and therefore the original constraint $t_k \in i$ introduced by the (\vee_{\forall}^1)-rule and the (\vee_{\forall}^k)-rule, has been reduced to $t_k \in i_k$. Therefore for some $t \in i \setminus (i_1 \cup \dots \cup i_n)$ $\neg \varphi_1 \wedge \dots \wedge \neg \varphi_n$ must hold, which contradicts that for all $t \in i$, $\varphi_1 \vee \dots \vee \varphi_n$ must hold.

(\subseteq_{\perp})-condition and ($\not\subseteq_{\perp}$)-condition: obvious. \triangleleft

Proposition 6.4 (Constrained reference time symbols) *If a set A of tableau entries contains no reference time symbols, then all reference time symbols in $Tab(A)$ are constrained by a unique $t \in i$ entry in the corresponding branch.*

Proof: The tableau rules which generate reference time symbols are the (\vee_{\forall}^1)-rule, the (\vee_{\forall}^k)-rule, the (\diamond_{\exists})-rule and the (\diamond_{\forall})-rule. They introduce constraints for the reference time symbols. The (\cap)-rule restricts these constraints to the smallest possible intervals. In $Tab(A)$ they are therefore unique. \triangleleft

Lemma 6.5 (Canonical Model for Open Branches)

If a set A of tableau entries contains no reference time symbols then all open branches in $Tab(A)$ have a model.

Proof: Let B_m be an open branch in $Tab(A)$. The canonical model $\mathcal{E} \stackrel{\text{def}}{=} ((\mathfrak{S}_{\mathcal{T}_U}, \mathcal{P}), \mathbb{T})$ for B_m is defined as follows:

If $t_1 \in i_1, \dots, t_n \in i_n$ are all the membership relations contained in B_m then we choose one element $n_k \in i_k$ such that all the n_k are different, and assign $\mathbb{T}(t_k) = n_k$. This is possible because i) none of the i_k is empty because otherwise the closing condition (\emptyset) would have been applied; ii) the time line is dense such there are enough points to choose from, and iii) by the (\cap)-rule, the interval i_k is uniquely determined for each $t_k \in i_k$.

For the predicate symbols p we assign

$$\mathcal{P}(p) = \{\mathbb{T}(t) \mid 't : p' \in B_m\} \cup \bigcup_{'i:p' \in B_m} i.$$

We have to show that \mathcal{E} really satisfies all entries. Let $B_1 \rightarrow \dots \rightarrow B_m$ be the sequence of tableau branches derived from the initial set B_1 of tableau entries and ending with the open branch B_m . We show by induction on the structure of the $\mathcal{CL}_{\mathcal{T}_U}$ -formulae that \mathcal{E} satisfies all entries in *all* branches B_1, \dots, B_m .

Base cases:

- i) Membership relations $t \in i$:
if $'t \in i' \in B_m$ then $\mathbb{T}(t) \in i$ holds by construction. If $'t \in i' \in B_k, k \neq m$ then $'t \in i'' \in B_m$ for some $i' \subseteq i$. This is the case because none of the rules removes a membership relation. But the rules (\setminus), (\cap), (τ_+) and (τ_-) may shrink the interval i . None of the rules increases the interval i . Since $\mathbb{T}(t) \in i'$ holds by construction, and $i \subseteq i'$, $\mathbb{T}(t) \in i$ must hold as well.
- ii) Positive predicate symbols p :
For $'t : p' \in B_m$, $\mathbb{T}(t) \in \mathcal{P}(p)$ is true by construction. None of the rules eliminates entries $t : p$. Therefore this holds for the entries $'t : p' \in B_k, k \neq m$ as well. The same arguments apply to entries $'i : p'$.
- iii) Negative predicate symbols p :
Consider $'t : \neg p' \in B_m$, $'t : p' \notin B_m$, because otherwise the closing condition (\neg) would have been applied. We also have to check that $'i : p' \notin B_m$, with $\mathbb{T}(t) \in i$. $'t : \neg p'$ is accompanied by a constraint $t \in j$ for some j (proposition 6.4). The (\setminus)-rule makes sure that j and i are disjoint. Therefore $\mathbb{T}(t) \in j$ and $\mathbb{T}(t) \notin i$. Thus, $\mathbb{T}(t) \notin \mathcal{P}(p)$, i.e. $\mathcal{E} \models t : \neg p$.

Now consider $'i : \neg p' \in B_m$, Due to the closing condition (\cap_-) there is no $'j : p' \in B_m$, with $j \cap i \neq \emptyset$. Again due to the (\setminus)-rule there is no $'t : p' \in B_m$ with constraint $t \in j$ which has a non-empty intersection with i . Therefore by construction, $\mathcal{P}(p) \cap i = \emptyset$, and this means $\mathcal{E} \models i : p$.

For entries $'t : \neg p' \in B_k$, and $'i : \neg p' \in B_k, k \neq m$, the same argument as in the positive case applies.

iv) Time terms:

Consider $\langle t : \tau \rangle \in B_k$ for some $\tau \in \mathcal{CL}_{\mathcal{T}_i}$ and $1 \leq k \leq m$. t is constrained by some $\langle t \in i \rangle \in B_k$. The (τ) -rule then yields $\langle t \in (i \cap \tau_{\perp}(i)) \rangle \in B_l$ for some $l \geq k$. As in the case above, we can conclude $\mathbb{T}(t) \in (i \cap \tau_{\perp}(i))$ and therefore $\mathbb{T}(t) \in \tau_{\perp}(t)$, i.e. $\mathcal{E} \models t : \tau$.

Now consider $\langle i : \tau \rangle \in B_k$. We must have $i \subseteq \tau_{\perp}(i)$ because otherwise the closing condition (τ_{\vee}) would apply. This means $\mathcal{E} \models i : \tau$.

The arguments for negated time terms are analogous.

v) Subsumption Relations:

Consider $t : \tau_1 \subseteq \tau_2$ with constraint $t \in i$. The (\subseteq_{\exists}) -rule guarantees that all the time points t in the interval i where $\tau_{1\perp}(s) \not\subseteq \tau_{2\perp}(s)$ are successively eliminated. The $(\not\subseteq_{\exists})$ -rule achieves the same effect for inequations $t : \tau_1 \subseteq \tau_2$.

Statements $i : \tau_1 \subseteq \tau_2$ and $i : \tau_1 \neq \tau_2$ are either valid, in which case the $(\subseteq_{\vee\top})$ -rule and the $(\neq_{\vee\top})$ -rule eliminates them, or they are not valid, in which case the closing conditions (\subseteq_{\perp}) or $(\not\subseteq_{\perp})$ apply.

Induction steps:

\wedge Consider $\langle t : \varphi_1 \wedge \varphi_2 \rangle \in B_k$. The (\wedge_{\exists}) -rule generates $\langle t : \varphi_1 \rangle \in B_l$ and $\langle t : \varphi_2 \rangle \in B_l$ for some $l \geq k$, to which the induction hypothesis applies.

The same argument holds for entries $\langle i : \varphi_1 \wedge \varphi_2 \rangle \in B_k$.

\vee Consider $\langle t : \varphi_1 \vee \varphi_2 \rangle \in B_k$. The (\vee_{\exists}) -rule generates $\langle t : \varphi_1 \rangle \in B_l$ or $\langle t : \varphi_2 \rangle \in B_l$ for some $l \geq k$, to which the induction hypothesis applies.

Now consider $\langle i : \varphi_1 \vee \dots \vee \varphi_n \rangle \in B_k$. The (\vee_{\vee}^1) -rule and the (\vee_{\vee}^k) -rule, together with the rules which shrink constraints for reference time symbols, generate the following entries: $\langle t_1 \in i_1 \rangle \in B_m, \langle t_1 : \varphi_1 \rangle \in B_{k_1}, \dots, \langle t_n \in i_n \rangle \in B_m, \langle t_n : \varphi_n \rangle \in B_{k_n}, k_l \geq k$. The induction hypothesis applies to all these entries. Due to the (\vee_{\vee}) -closing condition, $i_1 \cup \dots \cup i_n = i$.

Let $s \in i$. We have to show that $\mathcal{E}, t \models \varphi_1 \vee \dots \vee \varphi_n$. If $s = \mathbb{T}(t)$ for some reference time symbol t , then $\langle t \in j \rangle \in B_k$ and, due to the (\vee_{\vee}) -rule, either $\langle t \in (j \setminus i) \rangle \in B_{k'}$, which cannot be the case because $s = \mathbb{T}(t)$ and $s \in i$, or $\langle t : \varphi_l \rangle \in B_{k''}$ for some $1 \leq l \leq n$ and $k'' \geq k$, to which the induction hypothesis applies. Therefore $\mathcal{E}, t \models \varphi_1 \vee \dots \vee \varphi_n$.

If there is no t with $s = \mathbb{T}(t)$, we can argue as follows: Since $i_1 \cup \dots \cup i_n = i$, $s \in i_k$ for some $1 \leq k \leq n$ must be the case. Furthermore there is $\langle t_k \in i_k \rangle \in B_m$. In the construction of \mathbb{T} , we assigned $\mathbb{T}(t_k)$ to an arbitrary point in i_k . The only restriction was that the assignment must be different to the assignment of the other reference time symbols. Since there is no other t with $\mathcal{E}(t) = s$, we can assume that the assignment for t_k is just $\mathbb{T}(t_k) = s$. Together with the induction hypotheses, which guarantees, $\mathcal{E} \models t_k : \varphi_k$, we can then conclude. $\mathcal{E}, t \models \varphi_1 \vee \dots \vee \varphi_n$.

$t : \langle \tau \rangle$ Consider ' $t : \langle \tau \rangle \varphi' \in B_k$ together with the constraint $t \in i$. $\tau_{\parallel}(i) \neq \emptyset$, otherwise the (\diamond) -closing condition would apply. The (\diamond_{\exists}) -rule generated ' $t \in i_l' \in B_{k'}$, ' $t_l : \varphi' \in B_{k'}$ and ' $t_l \in \tau_{\parallel}(i_l)' \in B_{k'}$ for some $k' \geq k$ and $i_l \in \delta(i, \tau)$. This means $\mathbb{T}(t_k) \in \tau_{\parallel}(\mathbb{T}(t))$ and together with the induction hypothesis, $\mathcal{E} \models t : \langle \tau \rangle \varphi$.

$i : \langle \tau \rangle$ Consider ' $i : \langle \tau \rangle \varphi' \in B_k$. $\tau_{\parallel}(i) \neq \emptyset$, otherwise the (\diamond) -closing condition would apply. Let $s \in i$. We have to show $\mathcal{E}, t \models \langle \tau \rangle \varphi$, i.e. there is some $s' \in \tau_{\parallel}(s)$ with $\mathcal{E}, t' \models \varphi$.

The (\diamond_{\forall}) -rules yields for all $i_l \in \delta(i, \tau)$. ' $t_l : \varphi' \in B_{k'}$, ' $t_l \in \tau_{\parallel}(i_l)' \in B_{k'}$ for $k' \geq k$, to which the induction hypothesis applies. Since δ partitions i , there must be some $i_l \in \delta(i, \tau)$ with $s \in i_l$. Since τ is constant on i_l , $\mathbb{T}(t_k) \in \tau_{\parallel}(s)$ and the induction hypothesis guarantees $\mathcal{E}, \mathbb{T}(t_k) \models \varphi$.

$t : [\tau]$ Consider ' $t : [\tau] \varphi' \in B_k$ with the constraint $t \in i$. We have to show $\mathcal{E}, t \models \varphi$, for every $s \in \tau_{\parallel}(\mathbb{T}(t))$.

The (\square_{\exists}) -rule yields for some $i_l \in \delta(i, \tau)$ and some $k' \geq k$, ' $t \in i_l' \in B_{k'}$ and ' $\tau_{\parallel}(i_l) : \varphi' \in B_{k'}$. The decomposition δ , together with the induction basis applied to ' $t \in i_l'$ guarantees $\tau_{\parallel}(i_l) = \tau_{\parallel}(\mathbb{T}(t))$. With the induction hypothesis applied to ' $\tau_{\parallel}(i_l) : \varphi'$ this means $\mathcal{E}, t \models \varphi$, for every $s \in \tau_{\parallel}(\mathbb{T}(t))$.

$i : [\tau]$ Consider ' $i : [\tau] \varphi' \in B_k$. The (\square_{\forall}) -rule which generates $\tau_{\parallel}(i) : \varphi$ and the induction hypothesis guarantee $\mathcal{E} \models i : [\tau] \varphi$.

◁

Theorem 6.6 (Completeness of the Tableau System) *If a finite set A of Calendar Logic statements has no model, then $Tab(A)$ has only closed branches.*

Proof: If there was an open branch B_m in $Tab(A)$ then the canonical model for B_m would also satisfy all the formulae in the initial tableau (lemma 6.5), which is not possible because A has no model. ◁

7 Summary

We have developed a propositional temporal logic as a multi-modal logic where the temporal operators range over finite intervals. These intervals are specified in some language for expressing everyday temporal notions, as for example 'last weekend', 'next Friday', 'tomorrow'. Calendar Logic is quite independent of the particular structure of this time term language. It is only required that the interpretation of the time terms yields finite sets of finite intervals in a linear dense reference time line.

Satisfiability is decidable for this logic by translating them into propositional logic and using a decision procedure for propositional logic. Since the translation is exponential, we presented as an alternative a Tableaux decision procedure. The \square and \diamond -operators as well as the other connectives need to be

processed only once in each branch of the tableau. If only one branch is examined at a time, then only linear memory space is therefore needed. The factor in the space complexity depends on the size of the time intervals and on the granularity of the time units occurring in the initial formula. As an extreme case consider the term $i : \langle ms(s) \rangle p$ where the interval i ranges over, say a millennium. If $ms(s)$ denotes the set of millisecond co-ordinates within a second, then the \diamond_{\forall} rule would generate about 31 billion (one for each second in the millennium) entries $t_k : p, t_k \in i_k$ where i_k is the set of millisecond co-ordinates in the given second, again 1000 numbers. It is up to the user of such a logic to avoid this explosion.

The version of Calendar Logic presented in this paper is still of limited use. In combination with other logical systems it becomes more interesting for practical purposes. A combination with Boolean Algebra terms for reasoning about sets and set-theoretic relationships has been presented in [7]. In Gabbay et al's book *Temporal Logic: mathematical foundations and computational aspects* [6], Ohlbach has developed a combination of Calendar Logic with other modal type logics, in particular Description Logics and modal logics of knowledge and belief. Satisfiability for the combined logic can be decided with a resolution type calculus. This calculus also works for the pure Calendar Logic. It is therefore also an alternative for the translation into propositional logic and the tableau system presented in this paper.

References

- [1] Alexander Chergov and Michael Zakharayashev. *Modal Logic*, volume 35 of *Oxford Logic Guides*. Oxford University Press, 1997.
- [2] B. F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, Cambridge, 1980.
- [3] Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, 1997.
- [4] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal logic: mathematical foundations and computational aspects. Vol. 1.*, volume 28 of *Oxford logic guides*. Oxford University Press, Oxford, 1994.
- [5] Hans Jürgen Ohlbach. About real time, calendar systems and temporal notions. In H. Barringer and D. Gabbay, editors, *Proc. of ICTL 1997*, 1997.
- [6] Hans Jürgen Ohlbach. Calendar Logic. A chapter in Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds *Temporal Logic: mathematical foundations and computational aspects. Vol. 2.*, Oxford University Press, 1998.
- [7] Hans Jürgen Ohlbach and Jana Koehler. How to extend a formal system with a Boolean Algebra component. In P.H. Schmidt W. Bibel, editor,

Automated Deduction. A Basis for Applications, volume III, pages 57–75.
Kluwer Academic Publishers, 1998.