Nexttime is Not Necessary
(Extended Abstract)

Edith Spaan
Department of Mathematics and Computer Science
University of Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam

Abstract

We investigate the propositional modal logic of knowledge and time for distributed systems. Previous results by Halpern and Vardi [HV89] and Ladner and Reif [LR] illustrate that the validity problems for a number of these logics are highly intractable; in particular they prove a number of Π^1_1-completeness results. The logics considered by the above authors contain at least two out of the three temporal logic operators: "sometimes", "nexttime", and "until". Although their proofs rely heavily on either the "nexttime" or the "until" operator, we show that the completeness results remain valid if we restrict the temporal operators to "sometimes".

1 Introduction

Recently, there has been a lot of interest in the logics of knowledge and time as a tool for analyzing the behavior of distributed systems. In [HV89], Halpern and Vardi categorize these logics in terms of two parameters: the language used and the class of distributed systems considered. The languages they consider depend on the number of processors, the absence or presence of an operator for common knowledge and the use of linear versus branching time. As in [HV89], we denote these languages by $CKL_{(m)}$, $KL_{(m)}$, $CKB_{(m)}$ and $KB_{(m)}$, where m is the number of processors, C denotes the presence of an operator for common knowledge, and L and B stand for linear and branching time. All of these languages include temporal operators for nexttime, until and sometimes.

We will now briefly describe the classes of systems considered in [HV89]. We view a distributed system as a set of possible runs of the system. We assume that runs proceed in discrete steps, and if r is a run then (r, i) (for $i \in \mathbb{N}$) describes the state of the system at the i-th step of run r. We say that a processor knows a fact φ at a given point, if φ is true at all points (r', i') that the processor considers possible at that point.

A processor does not forget if the set of runs it considers possible stays the same or decreases over time. The dual notion is no learning: we say that a processor does not learn if the set of runs it considers possible stays the same or increases over time. A system is synchronous if all processors have access to a global clock. Finally, a system has a unique initial state if no processor can distinguish $(r, 0)$ from $(r', 0)$ for all runs r and r'.

We use $\mathcal{C}$ to represent the class of all models and use subscripts nf, nl, $sync$ and uis to indicate classes of models where, respectively, all processors do not forget, all processors do not learn, where time is synchronous, and where there exists a unique initial state.

In [HV89], Halpern and Vardi completely characterize the computational complexity for all combinations of their languages and classes of models, including some results from [LR]. We will now state all combinations that are undecidable and their respective complexity class.

Theorem 1.1 (HV89)

1. *The validity problem for* $CKL_{(\geq 2)}$ *and* $CKB_{(\geq 2)}$ *is* Π_1^1*-complete with respect to* $\mathcal{C}_{(nf)}$, $\mathcal{C}_{(nf,uis)}$, $\mathcal{C}_{(nf,sync)}$, $\mathcal{C}_{(nf,nl)}$, $\mathcal{C}_{(nf,sync,uis)}$, $\mathcal{C}_{(nf,nl,sync)}$, $\mathcal{C}_{(nl,sync)}$ *and* $\mathcal{C}_{(nl)}$.
2. *The validity problem for* $CKL_{(\geq 2)}$, $KL_{(\geq 2)}$, $CKB_{(\geq 2)}$ *and* $KB_{(\geq 2)}$ *is* Π_1^1*-complete with respect to* $\mathcal{C}_{(nf,nl,uis)}$.
3. *The validity problem for* $CKL_{(\geq 2)}$, $KL_{(\geq 2)}$, $CKB_{(\geq 2)}$ *and* $KB_{(\geq 2)}$ *is co-r.e.-complete with respect to* $\mathcal{C}_{(nl,uis)}$.

Since the validity problem for linear temporal logic with the three operators mentioned earlier is PSPACE-complete, while the validity problem for linear temporal logic with just $\Diamond$ (sometimes) as temporal operator is only co-NP-complete [SC], it will be interesting to examine the impact on the complexity if we restrict the temporal operators in our languages to $\Diamond$ for linear time (resp. $\forall\Diamond$ and $\exists\Diamond$ for branching time). Let $C\overline{K}\overline{L}_{(m)}$, $K\overline{L}_{(m)}$, $C\overline{K}\overline{B}_{(m)}$ and $K\overline{B}_{(m)}$ denote the languages where $\Diamond$ (resp. $\forall\Diamond$ and $\exists\Diamond$) are the only temporal operators. Although the proofs in [LR, HV89] rely heavily on the use of either the nexttime or the until operator, it turns out that, by using new techniques, we get the same complexity results if we restrict the temporal operators to $\Diamond$ (resp. $\forall\Diamond$ and $\exists\Diamond$). Using approximately the same techniques, we can prove that the well-known Π_1^1-hardness result for two-dimensional temporal logic [Ha] goes through if we restrict the temporal operators to the sometimes operators in both directions $\Diamond_u$ and $\Diamond_r$.

The rest of the paper is organized as follows. In the next section we describe the formal model according to [HV86, HV89]; in section 3 we describe the specific problems encountered if we have only $\Diamond$ as a temporal operator; in section 4 we prove the analogue of 1 for the linear time language $C\overline{K}\overline{L}_{(\geq 2)}$, by forcing models to be gridlike; in section 5 we prove the analogue of 2 and 3 for the linear time cases and a Π_1^1 lower bound for two-dimensional linear logic, by appropriately modifying the proof of Ladner and Reif [LR]. Finally, in section 6, we prove that for all classes of models considered, the validity problems for the branching time languages $C\overline{K}\overline{B}_{(m)}$ (resp. $K\overline{B}_{(m)}$) are at least as hard as the corresponding validity problems for $C\overline{K}\overline{L}_{(m)}$ (resp. $K\overline{L}_{(m)}$).

2 Syntax and Semantics

We start by giving the syntax of languages $CKL_{(m)}$ and $CKB_{(m)}$. We assume we have a set of propositional variables Φ and define the set of $CKL_{(m)}$ and $CKB_{(m)}$ formulas as follows:

- if $p \in \Phi$ then p is a $CKL_{(m)}$ ($CKB_{(m)}$) formula.
- if φ, ψ are $CKL_{(m)}$ ($CKB_{(m)}$) formulas, then so are $\neg\varphi$ and $\varphi \wedge \psi$.
- if φ is a $CKL_{(m)}$ ($CKB_{(m)}$) formulas, then so are $K_k\varphi$ (k knows φ), $E\varphi$ (everyone knows φ) and $C\varphi$ (φ is common knowledge).
- if φ, ψ are $CKL_{(m)}$ formulas, then so are $\bigcirc\varphi$ (nexttime φ), $\Diamond\varphi$ (sometimes φ) and $\varphi U \psi$ (φ until ψ).
- if φ, ψ are $CKB_{(m)}$ formulas, then so are $\forall\bigcirc\varphi$, $\exists\bigcirc\varphi$, $\forall\Diamond\varphi$, $\exists\Diamond\varphi$, $\forall\varphi U\psi$, $\exists\varphi U\psi$.

We define T, $\vee$ and $\rightarrow$ in the usual way from $\neg$ and $\wedge$. In addition, we define for linear time $\Box\varphi$ (always φ) as an abbreviation of $\neg\Diamond\neg\varphi$, and for branching time we view $\forall\Box\varphi$ (resp. $\exists\Box\varphi$) as an abbreviation of $\neg\exists\Diamond\neg\varphi$ (resp. $\neg\forall\Diamond\neg\varphi$).

We define the sublanguages $KL_{(m)}$ (resp. $KB_{(m)}$) as the set of $CKL_{(m)}$ (resp. $CKB_{(m)}$) formulas without the C operator. By restricting the temporal operators in each language to $\Diamond$ (resp. $\forall\Diamond$ and $\exists\Diamond$), we get the corresponding languages $CK\overline{L}_{(m)}$, $K\overline{L}_{(m)}$, $CK\overline{B}_{(m)}$ and $K\overline{B}_{(m)}$.

We will now give the semantics for $CKL_{(m)}$. A linear time model M for m processors is a tuple $(R, \pi, \sim_1, \ldots \sim_m)$, where R is a set of runs, $\pi : R \times \mathbb{N} \rightarrow \mathcal{P}(\Phi)$ assigns to each point the set of propositional variables that are true at that point, and $\sim_k$ is an equivalence relation on $R \times \mathbb{N}$. Note that we use the definition from [HV86]. We define $(M, r, i) \models \varphi$ (φ is satisfied by point (r, i) of M) with induction on φ:

- $(M, r, i) \models p \Longleftrightarrow p \in \pi(r, i)$
- $(M, r, i) \models \neg\varphi \Longleftrightarrow (M, r, i) \not\models \varphi$
- $(M, r, i) \models \varphi \wedge \psi \Longleftrightarrow (M, r, i) \models \varphi \wedge (M, r, i) \models \psi$
- $(M, r, i) \models K_k\varphi \Longleftrightarrow \forall(r', i') \sim_k (r, i) : (M, r', i') \models \varphi$
- $(M, r, i) \models E\varphi \Longleftrightarrow \forall k \leq m : (M, r, i) \models K_k\varphi$
- $(M, r, i) \models C\varphi \Longleftrightarrow \forall n : (M, r, i) \models E^n\varphi$
- $(M, r, i) \models \bigcirc\varphi \Longleftrightarrow (M, r, i+1) \models \varphi$
- $(M, r, i) \models \Diamond\varphi \Longleftrightarrow \exists j \geq i : (M, r, j) \models \varphi$
- $(M, r, i) \models \varphi U \psi \Longleftrightarrow \exists i' \geq i : (M, r, i') \models \psi \wedge \forall i''\, (i \leq i'' < i' \Rightarrow (M, r, i'') \models \varphi)$

Given a model M for m processors, we define $\sim_c$ as the transitive closure of $\bigcup_{k=1}^{m} \sim_k$. Then $(M, r, i) \models C\varphi$ if and only if $\forall(r', i') \sim_c (r, i) : (M, r', i') \models \varphi$.

We will now give the semantics for $CKB_{(m)}$. A branching time model M for m processors is a tuple $(F, \pi, \sim_1, \ldots \sim_m)$ where F is a forest, π assigns to each point of F the set of propositional variables that are true at that point, and $\sim_k$ is an equivalence relation on points of F. We assume that each node in F has some successor. We will view F as a tuple $< R_F, =_F >$ where R_F is the set of the infinite branches of F that start at the root of some tree in F. (r, i) denotes the i-th node of r and $(r, i) =_F (r', i)$ if and only if (r, i) and (r', i) denote the same node of F. We will define $(M, r, i) \models \varphi$ with induction on φ. We only give the clauses for $\exists\Diamond$ and $\forall\Diamond$, the other temporal operators are defined analogously.

- $(M, r, i) \models \forall\Diamond\varphi \Longleftrightarrow \forall(r', i) =_F (r, i)\ \exists j \geq i : (M, r', j) \models \varphi$
- $(M, r, i) \models \exists\Diamond\varphi \Longleftrightarrow \exists(r', i) =_F (r, i)\ \exists j \geq i : (M, r', j) \models \varphi$

As usual, we say that a formula φ is valid with respect to a class of models $\mathcal{D}$, if and only if for all models $M \in \mathcal{D}$ and for all points (r, i) of M : $(M, r, i) \models \varphi$. A formula φ is satisfiable with respect to $\mathcal{D}$ if and only if there is some model $M \in \mathcal{D}$ and some point (r, i) of M such that $(M, r, i) \models \varphi$.

We will now define the classes of models of [HV89]:

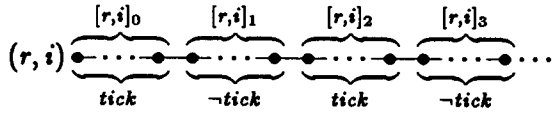
- Processor k does not forget in M if for all $r, r' \in R$ and $i, i' \in \mathbb{N}$: if $(r,i) \sim_k (r',i')$ then $\forall j \leq i \; \exists j' \leq i'$ such that $(r,j) \sim_k (r',j')$.
- Processor k does not learn in M if for all $r, r' \in R$ and $i, i' \in \mathbb{N}$: if $(r,i) \sim_k (r',i')$ then $\forall j \geq i \; \exists j' \geq i'$ such that $(r,j) \sim_k (r',j')$.
- Time is synchronous in M if for all $r, r' \in R$ and $i, i' \in \mathbb{N}$ and all processors k: $(r,i) \sim_k (r',i')$ implies that $i = i'$.
- M has a unique initial state if for all $r, r' \in R$ and all processors k: $(r,0) \sim_k (r',0)$.

We use $\mathcal{C}$ to represent the class of all models and use subscripts $nf, nl, sync, uis$ to indicate classes of models where, respectively, all processors do not forget, all processors do not learn, where time is synchronous, and where there is a unique initial state.

3 From Points to Intervals

Since all upper bounds follow directly from the corresponding upper bounds from [HV89], it will be enough to prove the corresponding lower bounds. As in [HV89], all Π^1_1 lower bounds for linear time will be proved by a reduction from the recurrence problem for nondeterministic Turing machines. A Turing machine is recurrent if and only if it has an infinite computation that starts on the empty tape and reenters its start state infinitely often. If $A_1, A_2, A_3, \ldots$ is a recursive enumeration of the 1-tape, right-infinite nondeterministic Turing machines, then $\{n \mid A_n \text{ is recurrent}\}$ is Σ^1_1-complete [HPS]. To prove a Π^1_1 lower bound for the validity problem with respect to a certain class of models $\mathcal{D}$, it will be sufficient to construct for an arbitrary 1-tape, right-infinite NTM A a formula φ_A such that A is recurrent if and only if φ_A is satisfiable with respect to $\mathcal{D}$.

In all our proofs, it is essential that the constructed formulas force runs to encode certain strings. The obvious way to encode some string on run r starting at i is by letting $(r, i+j)$ encode the j-th symbol of the string. However, if we restrict the temporal operators to just $\Diamond$, we can't distinguish adjacent points satisfying the same set of formulas. To solve this problem, we introduce a propositional variable *tick*, alternating on runs. *tick* partitions each run into an infinite number of intervals. For all n, let $[r,i]_n$ be the set of points in the n-th interval of r starting at i (Note that we start counting the intervals from 0).



We will encode strings at consecutive intervals of a run. We say that (r,i) encodes some string if and only if each point in the j-th interval of (r,i) ($[r,i]_j$) encodes the j-th element of the string. It is possible to mark a fixed number of consecutive intervals on a run by propositional constants. Let *1-int*(p, up_p) be the conjunction of the following formulas:

$\Diamond p \wedge \Box(p \wedge tick \rightarrow \Box(\neg tick \rightarrow \Box\neg p)) \wedge \Box(p \wedge \neg tick \rightarrow \Box(tick \rightarrow \Box\neg p))$
p holds somewhere at some interval, and p holds nowhere outside that interval.
$\Box(tick \wedge \Diamond(p \wedge tick) \rightarrow \Diamond(\neg tick \wedge \Diamond p) \vee p)$
$\Box(\neg tick \wedge \Diamond(p \wedge \neg tick) \rightarrow \Diamond(tick \wedge \Diamond p) \vee p)$
p holds exactly at some prefix of an interval
$\Box(p \wedge tick \rightarrow \Diamond(up_p \wedge \neg tick)) \wedge \Box(p \wedge \neg tick \rightarrow \Diamond(up_p \wedge tick))$
up_p holds at somewhere after the p interval.
$\Box(up_p \wedge \neg tick \rightarrow \Box(tick \rightarrow \Box\neg up_p)) \wedge \Box(up_p \wedge tick \rightarrow \Box(\neg tick \rightarrow \Box\neg up_p))$
if up_p holds somewhere at some interval,
then up_p holds nowhere outside that interval.
$\Box(p \rightarrow \Box((\neg up_p \wedge \Diamond up_p) \rightarrow p))$

Thus, $(r,i) \models 1\text{-}int(p, up_p)$ if and only if there is exactly 1 interval at which p holds, p holds nowhere else and up_p holds exactly at some prefix of the next interval.
Therefore if $(r,i) \models 1\text{-}int(p_0,p_1) \wedge 1\text{-}int(p_1,p_2) \wedge \ldots \wedge 1\text{-}int(p_{n-1}, up_p)$ then there are n consecutive intervals on r after i such that p_j holds at exactly the j-th interval.

Now we can define $\models$ and $\sim_k$ on intervals:

$[r,i]_n \models \varphi \iff \forall (r,j) \in [r,i]_n : (r,j) \models \varphi$
(Note that $[r,i]_n \not\models \varphi$ does not imply that $[r,i]_n \models \neg\varphi$)

$[r,i]_n \sim_k [r',i']_{n'} \iff \forall (r,j) \in [r,i]_n \exists (r',j') \in [r',i']_{n'} : (r,j) \sim_k (r',j') \wedge$
$\forall (r',j') \in [r',i']_{n'} \exists (r,j) \in [r,i]_n : (r,j) \sim_k (r',j')$

Though the specific classes considered do imply specific behavior for the epistemic relations with respect to points, not much of this behavior carries over to intervals. For example, it is perfectly possible for a model in $\mathcal{C}_{(sync)}$ not to be synchronous with respect to intervals. However, the following formula ψ_k will force that some of the structural properties of points hold for intervals as well.

$$K_k \Box((tick \rightarrow K_k tick) \wedge (\neg tick \rightarrow K_k \neg tick))$$

Lemma 3.1 *If $M \in \mathcal{C}_{(nf)}$, $(M,r,i) \models \psi_k$ and $(r,j) \sim_k (r',j')$ for some $(r,j) \in [r,i]_n (n > 0)$, then*

- *there exists some i' such that $(r,i) \sim_k (r',i')$*
- *for all i', if $(r,i) \sim_k (r',i')$ then $(r',j') \in [r',i']_n$ and $\forall n' < n : [r,i]_{n'} \sim_k [r',i']_{n'}$.*

Lemma 3.2 *If $M \in \mathcal{C}_{(nl,sync)}$ and $(r,i) \sim_k (r',i)$, then $\forall n : [r,i]_n \sim_k [r',i]_n$*

We leave the proofs of these two lemmas to the reader. They are similar to the proofs in [HV89] that force not necessarily synchronous models to be essentially synchronous.

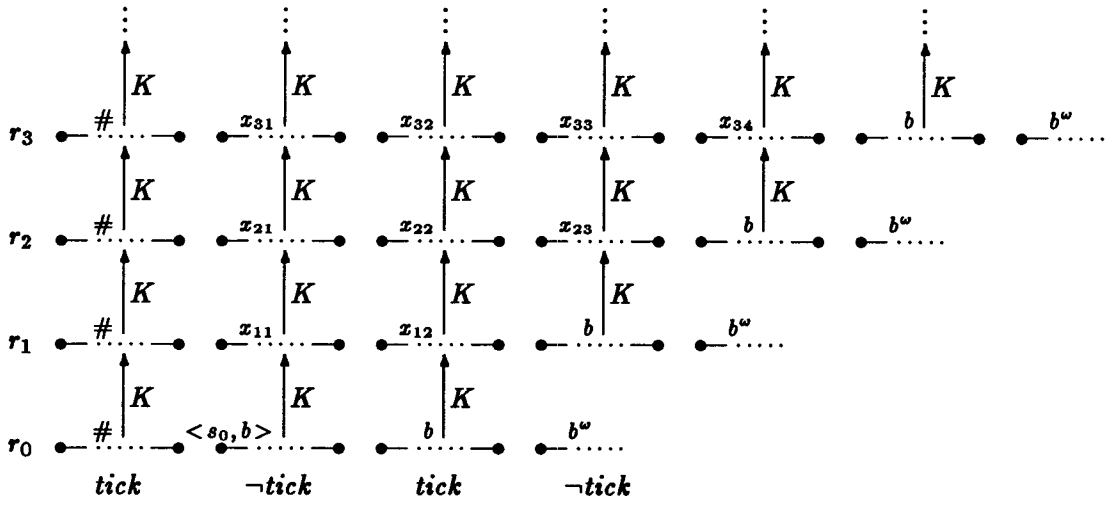
4 Forcing Models to be Gridlike

Theorem 4.1 *The validity problem for $CK\overline{L}_{(\geq 2)}$ is Π^1_1-complete with respect to $\mathcal{C}_{(nf)}$, $\mathcal{C}_{(nf,uis)}$, $\mathcal{C}_{(nf,sync)}$, $\mathcal{C}_{(nf,nl)}$, $\mathcal{C}_{(nf,sync,uis)}$ and $\mathcal{C}_{(nf,nl,sync)}$.*

Since the Π_1^1 upper bounds for these classes follow directly from [HV89], it will be enough to prove the lower bounds for two processors. Given an arbitrary 1-tape, right infinite NTM A, we will construct a formula φ_A such that:

- A recurrent $\Rightarrow$ φ_A satisfiable with respect to $C_{(nf,nl,sync)}$ and $C_{(nf,sync,uis)}$.
- φ_A satisfiable with respect to $C_{(nf)}$ $\Rightarrow$ A recurrent.

This implies a Π_1^1 lower bound for all six classes. Fix a 1-tape right-infinite NTM A. Suppose A has state space S, start state $s_0 \in S$; tape alphabet Γ; $b \in \Gamma$: the blank; and transition function δ. We use a special symbol # to mark the left side of the tape. Let CD (set of cell descriptors) be the set $\Gamma \cup \{\#\} \cup (S \times \Gamma)$. We view the IDs of A as infinite strings over CD, where $<s, a>$ denotes a cell with contents a, which is currently read by the head while A is in state s. A starts on the empty tape in state s_0, so the start ID of A (id_0) is equal to $\#<s_0, b>b^\omega$. Now suppose $id_0 \vdash id_1 \vdash id_2 \vdash \cdots$ is an infinite computation of A. Then for all n: $id_n = \#x_{n,1}x_{n,2}\ldots x_{n,n}x_{n,n+1}b^\omega$ ($x_{n,i} \in$ CD). The idea is to encode this computation in a model, by letting the runs represent the IDs (using the interval techniques of the previous section), and using the epistemic relations to simulate the transition function.



Since the encoding of IDs will be done at the intervals of runs, we start by partitioning each run into an infinite number of intervals, using the propositional variable *tick*. The following formula φ_1 will take care of this:

$$C\Box((tick \rightarrow \Diamond\neg tick) \wedge (\neg tick \rightarrow \Diamond tick))$$

The epistemic relation $\xrightarrow{K}$ is used to determine the contents of a cell at the next step of a computation. Therefore, it should not be reflexive, transitive or symmetric. As in [HV89], we use both epistemic relations $\sim_1$ and $\sim_2$ and introduce a propositional variable p_Δ to avoid reflexivity. Now we define the relation $\xrightarrow{K}$ and its associated modal operator K as follows:

$$(r,i) \xrightarrow{K} (r',i') \iff \exists r'', i'' : (r,i) \sim_1 (r'',i'') \sim_2 (r',i') \wedge (r'',i'') \models \neg p_\Delta \wedge (r',i') \models p_\Delta$$
$$K\psi := K_1(\neg p_\Delta \rightarrow K_2(p_\Delta \rightarrow \psi))$$

Let φ_2 be the conjunction of the following formulas:

$$C((p_\Delta \to \Box p_\Delta) \wedge (\neg p_\Delta \to \Box \neg p_\Delta))$$

$$C\Box C(\neg K_1 p_\Delta \wedge \neg K_2 \neg p_\Delta)$$

If $(r_0, i_0) \models \varphi_2$ and $(r, i) \sim_c (r_0, i_0)$ then the value of p_Δ on r from i upwards is constant, and by the second conjunct we can take an infinite number of $\xrightarrow{K}$ steps from each point on r after i. This will ensure that we encode an infinite computation.

Since we are interested in the behavior of $\xrightarrow{K}$ with respect to intervals, we define:

$$[r,i]_n \xrightarrow{K} [r',i']_{n'} \iff \forall (r,j) \in [r,i]_n \, \exists (r',j') \in [r',i']_{n'} : (r,j) \xrightarrow{K} (r',j') \wedge \\ \forall (r',j') \in [r',i']_{n'} \, \exists (r,j) \in [r,i]_n : (r,j) \xrightarrow{K} (r',j')$$

For the IDs to match up right, we need synchrony and no forgetting of $\xrightarrow{K}$ with respect to intervals. Let φ_3 be the following formula:

$$C\Box((tick \to Ctick) \wedge (\neg tick \to C\neg tick))$$

By lemma 3.1 and the fact that p_Δ is constant on runs we obtain the following lemma:

Lemma 4.2 *If* $(r_0, i_0) \models \varphi_3$, $(r, i) \sim_c (r_0, i_0)$, $(r, j) \xrightarrow{K} (r', j')$, *and* $(r, j) \in [r, i]_n (n > 0)$, *then*

- *there exists some* i' *such that* $(r, i) \xrightarrow{K} (r', i')$
- *for all* i', *if* $(r, i) \xrightarrow{K} (r', i')$ *then* $(r', j') \in [r', i']_n$ *and* $\forall n' < n : [r, i]_{n'} \xrightarrow{K} [r', i']_{n'}$.

Now we turn to the encoding of IDs. We will encode IDs on runs where p_Δ holds. To encode the cell descriptors, we introduce for each $x \in \mathrm{CD}$ a propositional variable p_x. Let φ_4 be the formula

$$C\Box(p_\Delta \to \bigvee_{x \in \mathrm{CD}} (p_x \wedge \neg \bigvee_{y \in \mathrm{CD}, y \neq x} p_y))$$

If $(r_0, i_0) \models \varphi_1, \ldots, \varphi_4$ and $(r, i) \sim_c (r_0, i_0)$ and p_Δ holds at (r, i) then each point on r after i encodes exactly one cell descriptor.

We say that the n-th interval of $(r, i)([r, i]_n)$ encodes $x \in \mathrm{CD}$, if each point in $[r, i]_n$ encodes x. To encode the start ID (id_0) at the first run, we introduce the following formula φ_{start} (up_s is a dummy variable):

$$p_\Delta \wedge \mathit{1\text{-}int}(p_\#, p_{<s_0,b>}) \wedge \mathit{1\text{-}int}(p_{<s_0,b>}, up_s) \wedge \Box(p_\# \vee p_{<s_0,b>} \vee p_b)$$

To simulate the transition function we just have to make sure that $\xrightarrow{K}$ points to the corresponding cell of a next ID. Suppose $id \vdash id'$. The only cells that can be affected by the transition are the cell holding the state and its neighbors. On each run we mark the 3 consecutive intervals corresponding to these cells with propositional variables *left*, *state*, *right*. Let $\varphi_{5,1}$ be the conjunction of the following formulas (up_{state} is a dummy variable):

$$C(p_\Delta \to \mathit{1\text{-}int}(left, state) \wedge \mathit{1\text{-}int}(state, right) \wedge \mathit{1\text{-}int}(right, up_{state}))$$

$$C\Box(p_\Delta \wedge \bigvee_{<s,a>\in S\times\Gamma} p_{<s,a>} \rightarrow state)$$

Now we can force the transition on all non-marked cells. Let $\varphi_{5,2}$ be the conjunction of the following formulas:

$$\bigvee_{x\in \mathrm{CD}} C\Box(p_\Delta \wedge (\neg left \wedge \neg state \wedge \neg right) \wedge p_x \rightarrow Kp_x)$$

$$C\Box(p_\Delta \wedge (\neg left \wedge \neg state \wedge \neg right) \wedge \Box p_b \rightarrow K\Box p_b)$$

Now the transition on the three marked intervals. Let $N(x,y,z)$ be the set of successor triples of $< x,y,z >$ as given by the transition function δ. Let $\varphi_{5,3}$ be the following formula:

$$C(p_\Delta \wedge \Diamond(left \wedge p_x) \wedge \Diamond(state \wedge p_y) \wedge \Diamond(right \wedge p_z) \longrightarrow \bigvee_{(x',y',z')\in N(x,y,z)} (\Box(left \rightarrow Kp_{x'}) \wedge \Box(state \rightarrow Kp_{y'}) \wedge \Box(right \rightarrow Kp_{z'})))$$

Let φ_5 be the conjunction of $\varphi_{5,1}, \varphi_{5,2}$ and $\varphi_{5,3}$. By lemma 4.2, if $(r,i) \models \varphi_1, \ldots, \varphi_5$ and (r,i) encodes some ID $id = x_0 \ldots x_{n-1} b^\omega$ of A, and for some $j \in [r,i]_m (m > n+1)$, $(r,j) \xrightarrow{K} (r',j')$ then there exists some i' such that $(r,i) \xrightarrow{K} (r',i')$ and (r',i') encodes a successor ID $y_0 \ldots y_n b^\omega$ of id.

Since by φ_2, we can take an infinite number of $\xrightarrow{K}$ steps from any point in the model, we know that we encode an infinite computation of A.

The only thing left to be done now, is to force the encoded computation to be recurrent. That is, at each time in the computation, there must be some later time where the computation is in the start state. To be able to express this requirement in a formula, we must be able to discriminate at each time those IDs which occur at some later step in the computation. Therefore, we time stamp each run that encodes an ID with the time of the computation. Say (r,i) is at time t if and only if exactly the $(t+2)$-nd and $(t+3)$-rd interval of (r,i) are marked with $time_1, time_2$. The first run is at time 0; we will mark the second and third intervals on this run with $time_1$ and $time_2$. Let φ_6 be the conjunction of the following two formulas (with up_{time} a new dummy variable):

$$C(p_\Delta \rightarrow 1\text{-}int(time_1, time_2) \wedge 1\text{-}int(time_2, up_{time})) \wedge C\Box(p_\Delta \wedge time_2 \rightarrow Ktime_1)$$

$$1\text{-}int(p_{<s_0,b>}, time_1)$$

That is, each p_Δ run is time stamped, and by lemma 4.2, if $(r,i) \sim_c (r_0,i_0)$ and (r,i) is at time t, and $(r,j) \xrightarrow{K} (r',j')$ for some $j \in [r,i]_m (m > t+2)$, then for each i' such that $(r,i) \xrightarrow{K} (r',i')$: (r',i') is at time $t+1$. By the second conjunct, (r_0,i_0) is at time 0.

To check whether an infinite computation is recurrent or not, we need to discriminate between runs that encode IDs in the start state and those that are in a different state. To this end we introduce the following formula φ_7:

$$C(\Diamond \bigvee_{x\in\Gamma} p_{<s_0,x>} \rightarrow \Box startstate) \wedge C(\Box \bigwedge_{x\in\Gamma} \neg p_{<s_0,x>} \rightarrow \Box \neg startstate)$$

If $(r_0, i_0) \models \varphi_1, \ldots, \varphi_7$ and $(r, i) \sim_c (r_0, i_0)$ and (r, i) encodes an ID, then *startstate* is constant on the run, *startstate* is true if the state of the encoded ID is the start state, false otherwise.

Finally, we state the formula to force recurrence φ_{rec}

$$C\Box(\Diamond C(p_\Delta \wedge time_2 \rightarrow startstate))$$

Let φ_A be the conjunction of φ_1 to φ_7, φ_{start} and φ_{rec}. Suppose $(M, r_0, i_0) \models \varphi_A$ for some $M \in \mathcal{C}_{(nf)}$. Then (r_0, i_0) encodes id_0. Suppose $(r, i) \sim_c (r_0, i_0)$ and (r, i) encodes some ID id at time t. By φ_{rec}, for each i' there must exist some $i'' \geq i'$ such that $(r, i'') \models C(p_\Delta \wedge time_2 \rightarrow startstate)$. In particular, there must exist some $m > 0$ and some j such that $(r, j) \in [r, i]_{t+3+m}$ and $(r, j) \models C(p_\Delta \wedge time_2 \rightarrow startstate)$. By φ_2, we can take $m \xrightarrow{K}$ steps from (r, j), say $(r, j)(\xrightarrow{K})^m(r', j')$. By φ_5, there must exist some i' such that $(r, i)(\xrightarrow{K})^m(r', i')$, (r', i') encodes some ID id' such that $id(\vdash)^m id'$ and $(r', j') \in [r', i']_{t+3+m}$. By φ_6, (r', i') is at time $t + m$, but then $(r', j') \models time_2 \wedge p_\Delta$ and therefore *startstate* is true at run r'. Thus, id' is in the start state. Since (r, i) was chosen arbitrarily, we have showed that M encodes a recurrent computation of A.

To conclude the proof of theorem 4.1, we still have to show that φ_A is satisfiable with respect to $\mathcal{C}_{(nf,nl,sync)}$ and $\mathcal{C}_{(nf,sync,uis)}$ if A is recurrent. As the proof is straightforward, we leave this to the reader.

Theorem 4.3 *The validity problem for* $CK\overline{L}_{(\geq 2)}$ *is* Π_1^1*-complete with respect to* $\mathcal{C}_{(nl,sync)}$.

We will show that formula φ_A works for $\mathcal{C}_{(nl,sync)}$ as well, i.e. A recurrent $\Leftrightarrow$ φ_A is satisfiable with respect to $\mathcal{C}_{(nl,sync)}$. By lemma 3.2, we can prove the following analogue of lemma 4.2.

Lemma 4.4 *If* $M \in \mathcal{C}_{(nl,sync)}$, $(r, i) \models \varphi_3$ *and* $(r, i) \xrightarrow{K} (r', i)$, *then* $\forall n : [r, i]_n \xrightarrow{K} [r', i]_n$

Analogous to the the proof of theorem 4.1, if $(M, r_0, i) \models \varphi_1, \ldots, \varphi_6$, $(r_0, i) \sim_c (r, i) \xrightarrow{K} (r', i)$ and (r, i) encodes some ID id at time t then, using lemma 4.4, (r', i) encodes a successor ID of id at time $t + 1$.

Suppose $(M, r_0, i) \models \varphi_A$ for some $M \in \mathcal{C}_{(nl,sync)}$. Then (r_0, i) encodes id_0. Suppose $(r, i) \sim_c (r_0, i)$ and (r, i) encodes some ID id at time t. As in the previous proof, there must exist some $m > 0$ and some j such that $(r, j) \in [r, i]_{t+3+m}$ and $(r, j) \models C(p_\Delta \wedge time_2 \rightarrow startstate)$. By φ_2, we can take $m \xrightarrow{K}$ steps from (r, i), say $(r, i)(\xrightarrow{K})^m(r', i)$. By φ_5 and φ_6, we know that (r', i) encodes some ID id' such that $id(\vdash)^m id'$ and (r', i) is at time $t + m$. By $(nl, sync)$, $(r, j) \sim_c (r', j)$ and by lemma 4.4, $(r', j) \in [r', i]_{t+m+3}$. Therefore, *startstate* is true at run r'. Thus, id' is in the start state. Since (r, i) was chosen arbitrarily, we have showed that M encodes a recurrent computation of A. Again, we leave the other direction of the equivalence to the reader.

Theorem 4.5 *The validity problem for* $CK\overline{L}_{(\geq 2)}$ *is* Π_1^1*-complete with respect to* $\mathcal{C}_{(nl)}$.

In our proof of the Π_1^1 lower bound of Theorem 4.3, it was essential that formula φ_3 forced synchrony with respect to intervals. This is not the case for models in $\mathcal{C}_{(nl)}$, since lemmas 3.2 and 4.4 do not hold for non-synchronous models. However, in [HV89] it is shown that we can force synchrony on finite prefixes of runs. This will enable us to force synchrony with respect to intervals on finite prefixes of runs. This will suffice to prove a Π_1^1 lower bound with respect to $\mathcal{C}_{(nl)}$ with minor changes to φ_A. Details are left to the full paper.

5 Variations on a Theme by Ladner and Reif

Theorem 5.1 *The validity problem for $K\overline{L}_{(\geq 2)}$ and $CK\overline{L}_{(\geq 2)}$ is Π_1^1-complete with respect to $\mathcal{C}_{(nf,nl,uis)}$.*

Since the Π_1^1 upper bounds for these classes follow directly from [HV89], it will be enough to prove the Π_1^1 lower bound for $K\overline{L}_{(2)}$. In [LR], Ladner and Reif prove that the validity problem for $KB_{(2)}$ is undecidable with respect to $\mathcal{C}_{(nf,nl,uis)}$. In particular, they construct for each deterministic Turing machine T a formula that forces a run to encode an infinite computation of T. As pointed out in [HV89], their proof can be trivially modified to obtain a Π_1^1 lower bound for $KL_{(2)}$. We will use the main idea of Ladner and Reif's proof to obtain for each nondeterministic Turing machine A a $K\overline{L}_{(2)}$ formula that encodes the recurrence problem for A.

Let A be a 1-tape right-infinite NTM. Suppose A has state space S, start state $s_0 \in S$; tape alphabet Γ; $b \in \Gamma$: the blank; and transition function δ. Let Δ be the set $\Gamma \cup \{\#, \$\} \cup (S \times \Gamma)$. We start by giving [LR]'s definitions extended to nondeterministic Turing Machines.

We view the IDs of A as finite strings of the form: $\$a_0\$a_1\$ \ldots \$a_n\$$ with $a_0 \ldots a_n \in \Gamma^*(S \times \Gamma)\Gamma^*$. A starts on the empty tape in state s_0 and we define the start ID of A id_0 as the string $\$<s_0, b>\$$. Define an infinite computation as an infinite string over Δ of the form: $\#^{m_0} id_0 \#^{m_1} id_1 \#^{m_2} \ldots$ with for each i : $m_i > 0$, $id_i \vdash id_{i+1}$, $|id_i| = 2i + 3$.

Define a function *collapse* : $\Delta^\omega \cup \Delta^* \to \Delta^\omega \cup \Delta^*$, that replaces multiple contiguous occurrences of the same symbol by one occurrence, that is:

$$collapse(a_0^{m_0} a_1^{m_1} a_2^{m_2} \ldots) = a_0 a_1 a_2 \ldots \in \Delta^\omega \text{ (if for all } i\text{: } m_i > 0, a_i \neq a_{i+1})$$
$$collapse(a_0^{m_0} a_1^{m_1} a_2^{m_2} \ldots a_r^{m_r}) = collapse(a_0^{m_0} a_1^{m_1} a_2^{m_2} \ldots a_r^{\omega}) = a_0 a_1 a_2 \ldots a_r \text{ (if for all } i\text{: } m_i > 0, a_i \neq a_{i+1})$$

Suppose σ and τ are infinite computations of the form:

$\sigma = \#\#\#\#\#\#id_0\#\#\#id_1\#\#\#id_2\#\#\#\cdots$

$\tau = \#\ \ id_0\ \#\ \ id_1\ \ \#\ \ id_2\ \ \#\ \ id_3\ \ \#\cdots$

Analogously to [LR], we can define a function $N : \Delta^6 \to \mathcal{P}(\Delta^6)$ that verifies the matching of these strings. If σ and τ are infinite computations as given, then $\forall i(\tau_i, \ldots, \tau_{i+5}) \in N(\sigma_i, \ldots, \sigma_{i+5})$. The following lemma shows how we can use N to determine if A has an infinite computation.

Lemma 5.2 (LR) *If σ, τ are infinite strings over Δ such that:*

1. $\sigma \in \#^6\$((\neg\{\#, \$\}\$)^*\#^3\$)^\omega$
2. $\tau \in (\neg\$\$)^\omega$
3. $\forall i : (\tau_i, \ldots, \tau_{i+5}) \in N(\sigma_i, \ldots, \sigma_{i+5})$
4. $collapse(\sigma) = collapse(\tau)$

Then σ and τ are infinite computations.

We will construct a formula ψ_A, such that ψ_A is satisfiable with respect to $\mathcal{C}_{(nf,nl,sync)}$ if and only if A is recurrent. As in [LR], we will encode two infinite computations on each run. Again we partition runs into an infinite number of intervals by the propositional variable *tick*. Let ψ_1 be the formula:

$$E\Box((tick \rightarrow \Diamond\neg tick) \wedge (\neg tick \rightarrow \Diamond tick))$$

If $(r_0, i_0) \models \psi_1$ then by (uis, nl), *tick* alternates on all runs.

Since we will encode two strings on each run, we need to encode 2 elements of Δ per point. Therefore we introduce for each $c \in \Delta$ two propositional variables s_c and t_c. Let ψ_2 be the conjunction of the following formulas:

$$E\Box(\bigvee_{c\in\Delta}(s_c \wedge \neg \bigvee_{d\in\Delta, d\neq c} s_d)) \wedge E\Box(\bigvee_{c\in\Delta}(t_c \wedge \neg \bigvee_{d\in\Delta, d\neq c} t_d))$$

If $(r_0, i_0) \models \psi_2$ and $(r, i) \sim_k (r_0, i_0)$ $(k \in \{1, 2\})$ then each point on r after i encodes exactly 2 elements of Δ, say a point encodes s = a and t = b if exactly s_a and t_b hold. An interval $[r, i]_n$ encodes $s = a$ [resp. $t = b$] if each point in that interval encodes $s = a$ $[t = b]$. Now we can define the encoding of strings on a run: (r, i) encodes $s^\omega = \sigma$ $[t^\omega = \tau]$ if for all n : $[r, i]_n$ encodes $s = \sigma_n$ $[t = \tau_n]$.

The formula ψ_A that we will construct will force the existence of strings σ and τ fulfilling the conditions of lemma 5.2. Following [LR], we will encode $collapse(\sigma)$ and $collapse(\tau)$ on the current run and σ and τ on other runs. We use propositional variable *coll*, constant on runs, to discriminate between the current run, where we want *coll* to hold, and the runs that encode the noncollapsed computations. The following formula ψ_3 will take care of this.

$$coll \wedge \neg K_1 coll \wedge E((coll \rightarrow \Box coll) \wedge (\neg coll \rightarrow \Box\neg coll))$$

As in [LR], we will enforce the following situation: if $(r_0, i_0) \models \psi_A$ then there exist strings σ and τ fulfilling conditions 1,2,3 and 4 of lemma 5.2 such that:

- if $(r_0, i_0) \sim_1 (r, i)$ and $(r, i) \models \neg coll$ then (r, i) encodes $s^\omega = \sigma$ and $t^\omega = \tau$
- (r_0, i_0) encodes $s^\omega = collapse(\sigma)$ and $t^\omega = collapse(\tau)$

If we have constructed ψ_A and $(r_0, i_0) \models \psi_A$, then by lemma 5.2, (r_0, i_0) encodes an infinite computation of A. We can then easily force this computation to be recurrent by adding the following conjunct ψ_{rec} to ψ_A:

$$\Box\Diamond \bigvee_{a\in\Gamma} s_{<s_0,a>}$$

We now turn to the construction of the formula ψ_A. First of all we have to make sure that if $(r_0, i_0) \sim_1 (r, i) \sim_1 (r', i')$ and $(r, i), (r', i') \models \neg coll$, then (r, i) and (r', i') encode the same strings. As a first step, we force synchrony for $\sim_1$, by the following formula ψ_4:

$$K_1\Box((tick \rightarrow K_1 tick) \wedge (\neg tick \rightarrow K_1\neg tick))$$

If $(r_0, i_0) \models \psi_1, \ldots, \psi_4$ and $(r, i) \sim_1 (r_0, i_0)$ then by (nl) for each $j_0 \geq i_0$ there exists some $j \geq i$ such that $(r, j) \sim_1 (r_0, j_0)$. By lemma 3.1, it follows that for all n : $[r, i]_n \sim_1 [r_0, i_0]_n$. We can now force all $\neg coll$ runs to encode the same strings, by formula ψ_5:

$$K_1\Box(\neg coll \wedge s_c \rightarrow K_1(\neg coll \rightarrow s_c)) \wedge K_1\Box(\neg coll \wedge t_c \rightarrow K_1(\neg coll \rightarrow t_c))$$

If $(r_0, i_0) \models \psi_1, \ldots, \psi_5$, $(r_0, i_0) \sim_1 (r, i) \sim_1 (r', i')$ and $(r, i), (r', i') \models \neg coll$ then, by ψ_4, $\forall n$: $[r, i]_n \sim_1 [r', i']_n$. By ψ_2 each point on r_0 and r encodes exactly two elements in Δ and therefore by ψ_5 $\forall n : [r, i]_n$ encodes $s = a$ $[t = b]$ if and only if $[r', i']_n$ encodes $s = a$ $[t = b]$.

We have to ensure that $\neg coll$ runs encode strings σ and τ fulfilling the conditions of lemma 5.2, i.e. $\sigma \in \#^6\$((\neg\{\#, \$\}\$)^*\#^3\$)^\omega$ and $\tau \in (\neg\$\$)^\omega$ such that $\forall i : (\tau_i, \ldots, \tau_{i+5}) \in N(\sigma_i, \ldots, \sigma_{i+5})$.

Following [LR], it can easily be seen that these conditions can be checked locally: we can construct a local condition such that if for all n this condition holds for $\sigma_n \ldots \sigma_{n+5}, \tau_n \ldots \tau_{n+5}$ (taking some extra care for the first seven symbols), then σ and τ are of the appropriate form. This is the reason why Ladner and Reif can force this situation using just one run. Obviously, one run won't suffice in our situation, since we don't have the nexttime operator. However, we can force the local condition for one interval at each run.

If $(r, i) \sim_1 (r_0, i_0)$, and $(r, i) \models \neg coll$ we use (r, i) to check the local condition for some interval $[r, i]_n$. In order to do this, we have to be able to distinguish the first 7 intervals of (r, i). We mark interval 0 to interval 6 of (r, i) by propositional variables $start_0$ to $start_6$, by the following formula ψ_6.

$$K_1(\neg coll \rightarrow start_0 \wedge \bigwedge_{k=0}^{5} 1\text{-}int(start_k, start_{k+1}) \wedge 1\text{-}int(start_6, up_{start}))$$

We can check the local condition for some interval by just looking at that interval and its 5 successors. We mark 6 consecutive intervals on each $\neg coll$ run by arg_0 to arg_5, using formula ψ_7:

$$K_1(\neg coll \rightarrow \bigwedge_{k=0}^{4} 1\text{-}int(arg_k, arg_{k+1}) \wedge (1\text{-}int(arg_5, up_{arg}))$$

It is now easy to construct a formula ψ_8 such that: if $(r_0, i_0) \models \psi_1, \ldots, \psi_8$ and $(r_0, i_0) \sim_1 (r, i)$, $(r, i) \models \neg coll$ and $[r, i]_n \models arg_0$ then $[r, i]_n$ fulfills the local condition (we leave the construction of this formula to the reader).

By ψ_5 we know that for each $(r', i') \sim_1 (r_0, i_0)$ such that $(r', i') \models \neg coll$ the following holds: $\forall n : [r, i]_n$ encodes $s = a$ $[t = b]$ if and only if $[r', i']_n$ encodes $s = a$ $[t = b]$. Therefore $[r', i']_n$ fulfills the local conditions as well. We have to make sure that each interval is checked, i.e. for each n there must be some $(r, i) \sim_1 (r_0, i_0)$ such that $(r, i) \models \neg coll$ and $[r, i]_n \models arg_0$. The following formula ψ_9 provides for this:

$$\Box \neg K_1 \neg(\neg coll \wedge arg_0)$$

Suppose $(r_0, i_0) \models \psi_1, \ldots, \psi_9$. Choose some $(r_0, j_0) \in [r_0, i_0]_n$. By ψ_9 there exists some $(r, j) \sim_1 (r_0, j_0)$ such that $(r, j) \models \neg coll \wedge arg_0$. By (nf), there is some $i \leq j$ such $(r, i) \sim_1 (r_0, i_0)$. Then by lemma 3.1, $(r, j) \in [r, i]_n$ and $[r, i]_n \models arg_0$ as required.

We have proved that if $(r_0, i_0) \models \psi_1, \ldots, \psi_9$, there exist σ and τ fulfilling conditions 1,2 and 3 of lemma 5.2 such that: for all $(r, i) \sim_1 (r_0, i_0)$ with $(r, i) \models \neg coll$: (r, i) encodes $s^\omega = \sigma$ and $t^\omega = \tau$.

In order to apply lemma 5.2, we have to ensure that condition 4 holds as well, i.e. $collapse(\sigma) = collapse(\tau)$. We will let (r_0, i_0) encode $s^\omega = collapse(\sigma)$ and $t^\omega = collapse(\tau)$ and force the two strings encoded by (r_0, i_0) to be equal. First we will force the condition for τ. Let ψ_{10} be the formula:

$$\Box(t_c \rightarrow K_1 t_c)$$

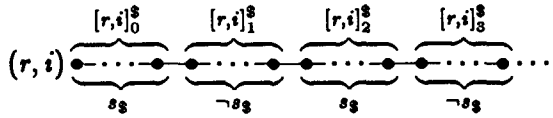
If $(r_0, i_0) \models \psi_1, \ldots, \psi_{10}$ then by ψ_3 there exists some $(r, i) \sim_1 (r_0, i_0)$ such that $(r, i) \models \neg coll$. By ψ_4, $\forall n : [r_0, i_0]_n \sim_1 [r, i]_n$. Since (r, i) encodes $t^\omega = \tau$ and each point on r_0 encodes exactly one value for t, (r_0, i_0) encodes $t^\omega = \tau = collapse(\tau)$.

We ensure that (r_0, i_0) encodes two equal strings by the following formula ψ_{11}:

$$\Box(t_c \leftrightarrow s_c)$$

If $(r_0, i_0) \models \psi_1, \ldots, \psi_{11}$ then (r_0, i_0) encodes $s^\omega = t^\omega = collapse(\tau)$.

Finally, we force (r_0, i_0) to encode $s^\omega = collapse(\sigma)$. As in [LR] we will use $\sim_2$ to simulate the collapse function for σ. Since $\sigma \neq collapse(\sigma)$, $\sim_2$ must behave differently from $\sim_1$. Therefore we partition the runs into different intervals, this time using our propositional variable $s_\$$. Let $[r, i]^\$_n$ be the n-th $s_\$$-interval of (r, i).



We can now force $\sim_2$ to be synchronous with respect to $s_\$$ intervals. Let ψ_{12} be the formula:

$$K_2 \Box((s_\$ \rightarrow K_2 s_\$) \wedge (\neg s_\$ \rightarrow K_2 \neg s_\$))$$

If $(r_0, i_0) \models \psi_1, \ldots, \psi_{12}$ and $(r, i) \sim_2 (r_0, i_0)$ then by (nl) $\forall j_0 \geq i_0$ there exists some $j \geq i$ such that $(r, j) \sim_2 (r_0, j_0)$. By lemma 3.1, it follows that $\forall n : [r, i]^\$_n \sim_2 [r_0, i_0]^\$_n$. We want the n-th $s_\$$ intervals of (r_0, i_0) and (r, i) to encode the same value for s. The following formula ψ_{13} will take care of this.

$$K_2 \Box(s_c \rightarrow K_2 s_c)$$

Suppose $(r_0, i_0) \models \psi_1, \ldots, \psi_{13}$, $(r_0, i_0) \sim_2 (r, i)$ and (r, i) encodes $s^\omega = \alpha$ and $collapse(\alpha) \in (\neg\$\$)^\omega$. Then $[r, i]^\$_n$ must encode $s = (collapse(\alpha))_n$, since the $s_\$$ intervals take adjacent identical s-symbols together. By ψ_{13}, $[r_0, i_0]^\$_n$ encodes $s = (collapse(\alpha))_n$ as well. Since we already know that (r_0, i_0) encodes $s^\omega = collapse(\tau) \in (\neg\$\$)^\omega$, the $s_\$$ and $tick$ intervals of (r_0, i_0) coincide. Thus, $[r_0, i_0]_n$ encodes $s = (collapse(\alpha))_n$ and therefore (r_0, i_0) encodes $s^\omega = collapse(\alpha)$. Since we want (r_0, i_0) to encode $collapse(\sigma)$, and $collapse(\sigma) \in (\neg\$\$)^\omega$, we just need to force the existence of some $(r, i) \sim_2 (r_0, i_0)$ such that (r, i) encodes $s^\omega = \sigma$. Let ψ_{14} be the formula:

$$\neg K_2 \neg(\neg coll \wedge arg_0 \wedge 1\text{-}int(start_0, start_1) \wedge \Box(arg_0 \rightarrow s_\#))$$

If $(r_0, i_0) \models \psi_1, \ldots, \psi_{14}$ then there exists some $(r, i) \sim_2 (r_0, i_0)$ such that $(r, i) \models \neg coll$ and $start_0$ holds exactly at $[r, i]_0$ and $[r, i]_0$ encodes $s = \#$. By (uis, nl), there must exist some j such that $(r, j) \sim_1 (r_0, i_0)$. By ψ_3, $(r, j) \models \neg coll$, and therefore (r, i) encodes $s^\omega = \sigma$ and $start_0$ holds exactly at $[r, j]_0$. But then (r, i) also encodes $s^\omega = \sigma$, and by ψ_{13} it follows that (r_0, i_0) encodes $s^\omega = collapse(\sigma)$.

Finally, let ψ_A be the conjunction of ψ_1 to ψ_{14}. If $(r_0, i_0) \models \psi_A$ then by lemma 5.2 (r_0, i_0) encodes $s^\omega = collapse(\sigma)$ and $collapse(\sigma)$ is an infinite computation of A. If (r_0, i_0) satisfies ψ_{rec} as

well, then $collapse(\sigma)$ is an infinite recurrent computation of A. Therefore, if $\psi_A \wedge \psi_{rec}$ is satisfiable with respect to $\mathcal{C}_{(nf,nl,uis)}$, then A is recurrent.

To conclude the proof of theorem 5.1 we have to check that $\psi_A \wedge \psi_{rec}$ is satisfiable with respect to $\mathcal{C}_{(nf,nl,uis)}$ if A is recurrent. Again, this is left to the reader.

Theorem 5.3 *The validity problem for $K\overline{L}_{(\geq 2)}$ and $CK\overline{L}_{(\geq 2)}$ is co-r.e.-complete with respect to $\mathcal{C}_{(nl,uis)}$.*

In the proof of theorem 4.5, we have mentioned that no learning enables us to force intervals to be synchronous with respect to finite prefixes of runs. Now we can add an extra conjunct to formula ψ_A of the previous proof to encode the halting problem. This gives us a r.e. lower bound for satisfiability and therefore a co-r.e. lower bound for validity. The corresponding upper bound follows from [HV89].

Two Dimensional Temporal Logic

We can apply the techniques of the proof of theorem 5.1 to obtain a Π^1_1 lower bound for two-dimensional temporal logic with only the two sometimes operators as temporal connectives. The models for two-dimensional temporal logic are two-dimensional grids, infinite to the right and upwards, i.e. each point is a pair (i, j) of natural numbers. Let $\overline{L}_2$ be the propositional language with operators $\Diamond_r$ (sometimes to the right) and $\Diamond_u$ (sometimes upwards) such that: $(M, i, j) \models \Diamond_r \varphi \Leftrightarrow \exists i' \geq i : (M, i', j) \models \varphi$, and $(M, i, j) \models \Diamond_u \varphi \Leftrightarrow \exists j' \geq j : (M, i, j') \models \varphi$.

Theorem 5.4 *The validity problem for $\overline{L}_{(2)}$ is Π^1_1-hard.*

We will briefly sketch how to construct a formula φ_A such that φ_A is satisfiable if and only if A is recurrent. First of all, note that if φ_A is satisfiable, then there exists a model M, such that $(M, 0, 0) \models \varphi_A$. Therefore, we will assume that the constructed formula is satisfiable in $(0, 0)$. Introduce two propositional variables $tick_r$ and $tick_u$ such that $tick_r$ alternates on horizontal runs and is constant on vertical runs, and $tick_u$ alternates on vertical runs and is constant on horizontal runs. We will use two-dimensional intervals $[(n, m)]$ $(n, m \in \mathbb{N})$ to take over the role of points:

$$[(n,m)] := \{(i,j) : (i,0) \text{ in the } n\text{-th } tick_r \text{ interval of } (0,0) \text{ and } (0,j) \text{ in the } m\text{-th } tick_u \text{ interval of } (0,0)\}$$

Now we do have a gridlike structure: for all n and m, there exist i_1, i_2, j_1, j_2 such that $[(n, m)] := \{(i, j) | i_1 \leq i \leq i_2 \wedge j_1 \leq j \leq j_2\}$

Using the same trick as in the proof of theorem 5.1, we can force the existence of strings σ and τ fulfilling conditions 1,2 and 3 of lemma 5.2, such that each horizontal run encodes $s^\omega = \sigma$ and $t^\omega = \tau$ on its consecutive horizontal $tick_r$ intervals.

We want $(0, 0)$ to encode the same strings $collapse(\sigma)$ and $collapse(\tau)$ vertically, i.e. on its consecutive $tick_u$ intervals. We use two new sets of propositional variables $\{\hat{s}_c : c \in \Delta\}$ and $\{\hat{t}_c : c \in \Delta\}$ and force the values of $\hat{s}$ and $\hat{t}$ to be constant on horizontal runs. To ensure that $(0, 0)$ encodes $\hat{t}^\omega = collapse(\tau)$, we mark the diagonal with propositional variable D, i.e. $(i, j) \models D$ if and only $(i, j) \in [(n, n)]$ for some n. Now we can force points on the diagonal to encode the same values for t and $\hat{t}$. This ensures that $(0, 0)$ encodes $\hat{t} = collapse(\tau)$.

It is easy to ensure that the strings vertically encoded on $(0,0)$ are equal. To ensure that $(0,0)$ encodes $\hat{s} = \mathit{collapse}(\sigma)$, we partition horizontal runs into intervals with $s_\$$ and vertical runs into intervals with $\hat{s}_\$$. Since the value of $s_\$$ is constant on vertical runs, and the value of $\hat{s}_\$$ is constant on horizontal runs, this gives us again a two-dimensional gridlike structure. We can mark the diagonal in this structure with D_s, and force the points where D_s holds to encode the same values for s and $\hat{s}$. This ensures that $(0,0)$ encodes $\hat{s} = \mathit{collapse}(\sigma)$. By lemma 5.2, $(0,0)$ encodes an infinite computation of A, and it is trivial to add a conjunct that forces this computation to be recurrent.

6 A Generic Reduction from Linear to Branching Time

Intuitively, the validity problems for branching time languages are harder then the corresponding validity problems for linear time. We will show that we can uniformly reduce the validity problems for $CK\overline{L}_{(m)}$ and $K\overline{L}_{(m)}$ to the corresponding validity problem for $CK\overline{B}_{(m)}$ and $K\overline{B}_{(m)}$, thus corroborating our intuition.

There is an obvious way to associate a branching time model with each linear time model and vice versa: suppose $M = (R, \pi, \sim_1, \ldots \sim_m)$ is a linear time model, then M is a branching time model as well; if $M = (F, \pi, \sim_1, \ldots \sim_m)$ is a branching time model then we define the corresponding linear time model M_L as $(R_F, \pi, \sim_1, \ldots \sim_m)$ (recall that R_F is the set of branches in F). Note that if $M \in \mathcal{D}$ where $\mathcal{D}$ is one of our sixteen classes of models, then $M_L \in \mathcal{D}$.

Theorem 6.1 *There exists a polynomial time bounded function f from $CK\overline{L}_{(m)}$ to $CK\overline{B}_{(m)}$ formulas such that:*

1. *for each linear time model M and all (r,i): $(M,r,i) \models \varphi \Rightarrow (M,r,i) \models f(\varphi)$*
2. *for each branching time model M and all (r,i): $(M,r,i) \models f(\varphi) \Rightarrow (M_L,r,i) \models \varphi$*

And if $\varphi \in K\overline{L}_{(m)}$ then $f(\varphi) \in K\overline{B}_{(m)}$.

As a first attempt, we take g to be the function that replaces all $\Diamond$ occurrences in a $CK\overline{L}_{(m)}$ formula by $\forall\Diamond$. Function g does not satisfy the conditions. The problem is that in branching time models $\exists\Diamond g(\psi)$ can hold, while $\forall\Diamond g(\psi)$ does not hold. Given a $CK\overline{L}_{(m)}$ formula φ, we will exclude this situation for all subformulas $\Diamond\psi$ of φ in all relevant points. Define a function *lin* from $CK\overline{L}_{(m)}$ to $CK\overline{B}_{(m)}$ formulas:

$$\mathit{lin}(p) = T;\ \mathit{lin}(\neg\varphi) = \mathit{lin}(\varphi);\ \mathit{lin}(\varphi \wedge \psi) = \mathit{lin}(\varphi) \wedge \mathit{lin}(\psi)$$
$$\mathit{lin}(K_k\varphi) = K_k\mathit{lin}(\varphi);\ \mathit{lin}(E\varphi) = E\mathit{lin}(\varphi);\ \mathit{lin}(C\varphi) = C\mathit{lin}(\varphi)$$
$$\mathit{lin}(\Diamond\varphi) = (\forall\Diamond g(\varphi) \leftrightarrow \exists\Diamond g(\varphi)) \wedge \forall\Box\,\mathit{lin}(\varphi)$$

By an easy induction on the structure of formula φ, we can prove the following lemma.

Lemma 6.2 *If $M = (F, \pi, \sim_1, \ldots \sim_m)$ is a branching time model such that $(M,r,i) \models \mathit{lin}(\varphi)$ then $(M,r,i) \models g(\varphi) \Leftrightarrow (M_L,r,i) \models \varphi$.*

Let $f(\varphi) := g(\varphi) \wedge lin(\varphi)$; we will prove that f fulfills the conditions of theorem 6.1. Suppose M is a linear time model and $(M, r, i) \models \varphi$. If we view M as a branching time model, then for all points (r', i') in M and all branching time formulas ψ : $(M, r', i') \models \forall\Diamond\psi \leftrightarrow \exists\Diamond\psi$. Therefore, $(M, r, i) \models lin(\varphi)$ and by lemma 6.2, $(M, r, i) \models g(\varphi)$. But then $(M, r, i) \models f(\varphi)$ as required. If M is a branching time model and $(M, r, i) \models f(\varphi)$, then by lemma 6.2 $(M_L, r, i) \models \varphi$.

Corollary 6.3 *If $\mathcal{D}$ is one of our sixteen classes of models then there is a polynomial time reduction from the validity problem for $CK\overline{L}_{(m)}$ (resp. $K\overline{L}_{(m)}$) with respect to $\mathcal{D}$ to the validity problem for $CK\overline{B}_{(m)}$ (resp. $K\overline{B}_{(m)}$) with respect to $\mathcal{D}$.*

Corollary 6.4

- *The validity problem for $CK\overline{B}_{(\geq 2)}$ is Π_1^1-complete with respect to $\mathcal{C}_{(nf)}$, $\mathcal{C}_{(nf,uis)}$, $\mathcal{C}_{(nf,sync)}$, $\mathcal{C}_{(nf,nl)}$, $\mathcal{C}_{(nf,sync,uis)}$, $\mathcal{C}_{(nf,nl,sync)}$, $\mathcal{C}_{(nl,sync)}$ and $\mathcal{C}_{(nl)}$.*
- *The validity problem for $CK\overline{B}_{(\geq 2)}$ and $K\overline{B}_{(\geq 2)}$ is Π_1^1-complete with respect to $\mathcal{C}_{(nf,nl,uis)}$.*
- *The validity problem for $CK\overline{B}_{(\geq 2)}$ and $K\overline{B}_{(\geq 2)}$ is co-r.e.-complete with respect to $\mathcal{C}_{(nl,uis)}$.*

Lower bounds by theorems 4.1, 4.3, 4.5, 5.1 and 5.3. Upper bounds by [HV89].

Acknowledgements: I'd like to thank Otto Moerbeek and Michiel Smid for their help with LaTeX, Peter van Emde Boas for his helpful comments on various versions of this paper, and most of all, Rineke Verbrugge for the discussions and comments on every draft of this paper, and for all her support.

References

[Ha] D. Harel, Recurring dominoes: making the highly undecidable highly understandable, *Proc. of the Conference on Foundations of Computing Theory*, Springer Lecture Notes in Computer Science - Vol. 158, 1983, pp. 177-194.

[HPS] D. Harel, A. Pnueli, and J. Stavi, Propositional dynamic logic of nonregular programs, *J. Comput. System Sci.*, 26, 1983, pp. 222-243.

[HV86] J.Y. Halpern and M.Y. Vardi, The complexity of reasoning about knowledge and time: extended abstract, *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, 1986, pp. 304-315.

[HV89] J.Y. Halpern and M.Y. Vardi, The complexity of reasoning about knowledge and time, I: Lower Bounds, *J. Comput. System Sci.*, 38, 1989, pp. 195-237.

[LR] R. Ladner and J.H. Reif, The logics of distributed protocols, *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the 1986 Conference*, 1986, pp. 207-221.

[SC] A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logics, *J. Assoc. Comput. Mach.*, 32, 1985, pp. 733-749.