Temporal Logic, Automata, and Classical Theories

— An Introduction —

Notes for the Sixth European Summer School
in Logic, Language, and Information,
Copenhagen 1994

Mads Dam
Swedish Institute of Computer Science

1 Introduction and Overview

The area of intersection between temporal logic, automata on finite and infinite objects, and classical first- or restricted second-order logics is one of considerable richness. All these areas have long and venerable traditions in mathematics, logic, and theoretical computer science, and their intimate relationships have been realised for quite some time. Indeed, automata of infinite objects were invented for the purpose of just answering decidability issues in classical first- or restricted second-order logics. However, the area has remained open to new points-of-view and insights, and very fundamental questions have yet to be both asked and answered.

The point of departure here is that of *programs*, the *computations* that programs give rise to, and *logics* expressing properties of programs in terms of their computations. These notions are far from well determined. Indeed it is one of our central aims to throw light on what are "good" (say, "tractable", or "complete") notions of computation or property. However, having this point of departure is helpful to cut down what are immense subject areas to more manageable sizes, and it should be kept in mind in the sequel that we raise and answer questions guided by our judgment of their relevance to our point of departure.

The concepts of programs, computations, and their properties come in wide ranges of flavours, dictated, to some extent, by the applications one has in mind. Typically programs are identified with some sort of state transition system that may be labelled in various ways to record interaction ports or types, primitive properties holding of states, or value assignments to identifiers. Computations then can be viewed as linear runs, traces, or history sequences, or it may be viewed more appropriate to view computations as trees to keep information about possible future or past choices. The choice of properties, then, can be guided by the desired expressive power, or by algorithmic or proof theoretic manageability, by model checking concerns, by their relation to program structure, or by their "executability". Then we haven't even touched upon issues like real time or second-or higher order parameter passing which are getting more and more serious attention elsewhere in the literature.

Our aim here is to chart out some initial features in this vast field of choices. Starting from very basic notions we shall be looking at more and more refined notions of computation and computation property, acquiring in the process an ever deeper understanding of the structure of computations, what constitutes suitable notions of computation property, and the expressive power and complexity of logics that embody those properties. Two parameters push this process. Concrete program verification exercises give some information as to the adequacy or otherwise of a given logic. Information of this type must, however, remain partial. More complete analyses can be made by comparing temporal logics with each other, or with other formalisms. Prime candidates for such external comparisons are first- or restricted second-order classical logics, derived, as they often are, in a very natural way by viewing the underlying computation structures as structures in the classical sense. A key line of inquiry is thus to answer questions such as: Which classical logic does this temporal logic correspond to? Automata have turned out to be highly useful in answering questions of this kind. Of course, the usefulness of automata in temporal logic is already well established. In particular they have been used with very considerable success to produce efficient decision procedures for a large variety of temporal logics, and we shall have occasion to look at some central results in this area in the course of these notes. However, it may be useful to keep in mind that our primary objective is to find new ways of throwing light on the expressive power of various temporal logics and not the issue of efficient decidability per se.

We start our investigations with a well understood case, namely that of linear time temporal logic. Here programs are identified with transition systems, and computations are taken to be infinite runs, or traces through transition systems. In this case we have a number of very strong and elegant results that set the agenda for the currently much more open-ended area of branching time logic. For branching time logics it is much less clear what are the "right" concepts of computation, and even less what are the "right" automata or classical theories with which to compare expressive power. Due in part to this situation the study of expressiveness in branching time logic has tended to focus more at comparing temporal logics with each other rather than with external theories. This situation is reflected by the later part of the note dealing with branching time logics.

2 Literature

We only point out here a few background references in these vast fields. Many other references are scattered around the note as appropriate to the material. Good texts on automata theory are e.g. [29, 37, 62]. The paper by Thomas in the handbook of Theoretical Computer Science [55] gives an excellent overview of automata on infinite objects. Eilenberg's two volumes [13] are older but more encyclopedic. In the area of (modal and) temporal logic good background texts are e.g. [30, 56, 21]. For surveys Stirling's paper [51] is highly recommendable. Recommendable too are [15, 39]. Finally, on the topic of classical second-order theories, a useful reference is Gurevich's chapter [22].

Acknowledgements

Colin Stirling helped initiate the writing of these notes. Thomas's handbook chapter [55] provided lots of inspiration. Thanks to David Turner for valuable suggestions, in particular concerning the expressive completeness proof for linear time temporal logic with until.

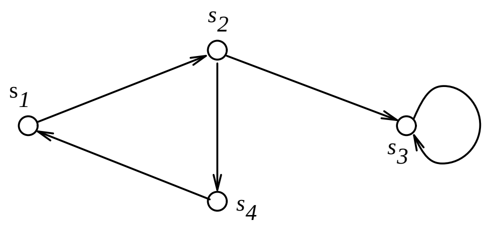

Figure 1: Example transition system

3 Programs and Transition Systems

We start by looking at the fundamental concepts of transition systems, how they are used to give operational semantics to simple programming languages, and conclude with a brief discussion of modal logic in this context.

Definition 3.1 *(Transition system)* A *transition system* (TS) is a structure

$$\mathcal{T} = (S, \rightarrow)$$

with

1. S a set of *states* $s \in S$, and
2. $\rightarrow \subseteq S \times S$ the *transition relation*

States are alternatively known as possible worlds, and the transition relation as the accessibility relation between possible worlds. But lots of other terminology exists. We usually use an infix notation for $\rightarrow$, writing $s_1 \rightarrow s_2$ instead of $(s_1, s_2) \in \rightarrow$. Finite transition systems have obvious graphical representations. For instance, the transition system in fig. 1 has four states, $s_1, \ldots, s_4$ and the transition relation has as elements the pairs (s_1, s_2), (s_2, s_3), (s_2, s_4), (s_3, s_3), and (s_4, s_1).

Examples of transition systems abound in the literature. Just think of automata, rewriting systems, or Kripke models. In many cases some sort of labelling of transitions or states is needed. Labelling states result in modal models.

Definition 3.2 *(Modal model)* Consider a set of primitive predicate symbols $q \in Q$. A *modal model* is a structure

$$\mathcal{T} = (S, \rightarrow, V)$$

with

1. $(S, \rightarrow)$ a transition system, and

2. $V : q \mapsto 2^S$ is a *valuation* mapping q into the set $V(q) \subseteq S$ of states for which q is assumed to hold.

The primitive predicate symbols are known alternatively as propositional variables, or atomic propositional letters, or something in that direction. We normally presuppose Q to form a countably infinite set. We give a little example to show how modal models can be used to give operational semantics to "while"-programs in the style of Plotkin [45].

Example 3.3 *(*`while`*-programs)* Assume given a set Ide of program identifiers. Let i, i_1, i_2 etc. range over Ide. A rudimentary language of while programs is given by the following abstract syntax

$$\begin{array}{rcl} BExp & ::= & Ide = Ide \,\big|\, \neg BExp \,\big|\, BExp \wedge BExp \\ Cmd & ::= & \texttt{skip} \,\big|\, Ide := BExp \,\big|\, Cmd; Cmd \,\big| \\ & & \texttt{if}\ BExp\ \texttt{then}\ Cmd\ \texttt{else}\ Cmd \,\big|\, \texttt{while}\ Cmd\ \texttt{do}\ Cmd \end{array}$$

The set of states S, now, is taken to be the set

$$\{(c, \sigma) \mid c \in Cmd, \sigma \in Ide \to_{\mathrm{fin}} 2\} \cup (Ide \to_{\mathrm{fin}} 2)$$

where $Ide \to_{\mathrm{fin}} 2$ is the set of finite maps from Ide to $2 = \{0, 1\}$. The update operation $\sigma[Ide := n]$ is defined by

$$\sigma[i := n](i') = \begin{cases} n & \text{if } i = i' \\ \sigma(i') & \text{otherwise} \end{cases}$$

Note that σ extends in a canonical way to all boolean expressions $b \in BExp$ by structural induction:

$$\begin{array}{rcl} \sigma(i_1 = i_2) & = & \begin{cases} 1 & \text{if } \sigma(i_1) = \sigma(i_2) \\ 0 & \text{otherwise} \end{cases} \\ \sigma(\neg b) & = & \begin{cases} 1 & \text{if } \sigma(b) = 0 \\ 0 & \text{otherwise} \end{cases} \\ \sigma(b_1 \wedge b_2) & = & \begin{cases} 1 & \text{if } \sigma(b_1) = \sigma(b_2) = 1 \\ 0 & \text{otherwise} \end{cases} \end{array}$$

We identify $BExp$ with Q and define

$$V(b) = \{(c, \sigma), \sigma \mid \sigma(b) = 1\}.$$

We are now in a position to define the transition relation. This we do by a set of inference rules determining $\to$ by induction in the structure of commands, for instance:

$$\frac{\cdot}{(\texttt{skip}, \sigma) \to \sigma} \qquad \frac{\cdot}{(i := b, \sigma) \to \sigma[Ide := \sigma(b)]}$$

$$\frac{(c_1,\sigma)\to(c_1',\sigma')}{(c_1;c_2,\sigma)\to(c_1';c_2,\sigma')} \qquad \frac{(c_1,\sigma)\to\sigma'}{(c_1;c_2,\sigma)\to(c_2,\sigma')}$$

$$\frac{\sigma(b)=1 \qquad (c_1,\sigma)\to s}{(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\sigma)\to s} \qquad \frac{\sigma(b)=0 \qquad (c_2,\sigma)\to s}{(\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2,\sigma)\to s}$$

$$\frac{\sigma(b)=1 \quad (c;\texttt{while}\ b\ \texttt{do}\ c,\sigma)\to s}{(\texttt{while}\ b\ \texttt{do}\ c,\sigma)\to s} \qquad \frac{\sigma(b)=0}{(\texttt{while}\ b\ \texttt{do}\ c,\sigma)\to\sigma}$$

Instead of labelling states it is also possible to label transitions, allowing closer analyses of issues relating to communication, synchronisation, and control.

Definition 3.4 *(Labelled transition system, labelled modal model)* Let Σ be a nonempty set of labels, or actions. A *labelled transition system* (LTS) is a structure

$$\mathcal{T}=(S,\xrightarrow{a}_{a\in\Sigma})$$

such that for each $a\in\Sigma$, $(S,\xrightarrow{a})$ is a transition system. A *labelled modal model* is a structure $(S,\xrightarrow{a}_{a\in\Sigma},V)$ such that for each $a\in\Sigma$, $(S,\xrightarrow{a},V)$ is a modal model.

Also labelled transition systems abound in the literature. Notable examples are the many variants of automata, and process algebras such as CCS [42], CSP [28] (or ACP, or LOTOS, or...). Here we give an example based upon CCS for later reference. We return to the topic of automata later in the note.

Example 3.5 *(CCS [42])* Assume given a set of *names*, $x,y,z,\ldots$. An *action* is either a name x, a *co-name* $\overline{x}$, or the constant τ, a special symbol used to represent an unobservable action resulting from the simultaneous occurrence of a name and a co-name. By convention, $\overline{\overline{x}}=x$. α is used to range over actions, l to range over actions distinct from τ, and L to range over sets of actions with the property that $\tau\notin L$ and if $x\in L$ then also $\overline{x}\in L$. The language of finite CCS *process expressions* [1] is determined by the following abstract syntax

$$P ::= 0 \,\big|\, \alpha.P \,\big|\, P+P \,\big|\, P\mid P \,\big|\, P\backslash L \,\big|\, X \,\big|\, \mathit{fix}X.P$$

where X ranges over process variables. A *process*, then, is a process expression without free occurrences of process variables.

Intuitively, $+$ is used for choice, $\mid$ for parallel composition, *fix* for infinite processes, and $\backslash$ for restriction. Thus $P\backslash L$ is meant to prevent P from executing transitions labelled by actions in L. More formally, processes are given semantics in terms of labelled transition systems by the following set of rules:

$$\frac{\cdot}{\alpha.P\xrightarrow{\alpha}P}$$

[1] We do not follow Milner's presentation to the letter.

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L} \quad (\alpha \notin L)$$

$$\frac{P[fixX.P/X] \xrightarrow{\alpha} P'}{fixX.P \xrightarrow{\alpha} P'}$$

We do not here enter deeper into discussions of the meaning or rationale of these operators. For this we refer to Milner's book [42].

4 Modal Logic

We start our look at program logics with modal logic. Since we can give operational semantics to programs in terms of transition systems a natural first approximation of the notion of computation may be simply some form of transition system together with a distinguished current state. As a language for expressing properties of transition systems modal logic contains classical propositional connectives to express properties of states together with operators for expressing universal or existential properties of successors of states along the transition relations. Modal logic is thus a natural first candidate for a logic for expressing properties of computations.

The language of modal formulas $\phi, \psi \in \mathcal{F}(\Box)$ is determined by the following abstract syntax:

$$\phi ::= q \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid [a]\phi$$

where $q \in Q$ and $a \in \Sigma$. Intuitively, $[a]\phi$ expresses of a state s that all s' such that $s \xrightarrow{a} s'$ satisfies ϕ. To disambiguate we assume $[a]$ to have strongest binding power and otherwise adopt standard conventions. Standard abbreviations include boolean disjunction and the "diamond":

$$\begin{aligned} \phi \vee \psi &\triangleq \neg(\neg\phi \wedge \neg\psi) \\ <a>\phi &\triangleq \neg[a]\neg\phi \end{aligned}$$

Also the boolean truth- and falsehood constants $\top$ and $\bot$ can be defined by $\top \triangleq q \vee \neg q$ for some arbitrary q, and $\bot \triangleq \neg\top$. In case Q is empty $\top$ needs to be specifically introduced. In the unlabelled case where Σ is a singleton set $\{a\}$, say, $\Box$ ($\Diamond$) is used in place of $[a]$ ($<a>$). In the case where Q is empty, $\mathcal{F}(\Box)$ is known alternatively as Hennessy-Milner logic (from [25]).

Definition 4.1 *(Semantics of $\mathcal{F}(\Box)$)* Let $\mathcal{T}$ be a labelled modal model. The relation $\mathcal{T}, s \models \phi$ is defined by induction in the structure of ϕ by

$s \models q$ iff $s \in V(q)$

$s \models \neg\phi$ iff $s \not\models \phi$

$s \models \phi \wedge \psi$ iff $s \models \phi$ and $s \models \psi$

$s \models [a]\phi$ iff $\forall s'$ if $s \xrightarrow{a} s'$ then $s' \models \phi$

When the model $\mathcal{T}$ is understood from the context we normally abbreviate $\mathcal{T}, s \models \phi$ by $s \models \phi$.

Exercise 4.2 *(Very easy)* Check that

$s \models \phi \vee \psi$ iff $s \models \phi$ or $s \models \psi$

$s \models <a>\phi$ iff $\exists s'$ such that $s \xrightarrow{a} s'$ and $s' \models \phi$

Remark 4.3 Modal logic is a very big and interesting field. Look for instance at the background references given earlier for more information. Very roughly there are two main issues of contention: Completeness and definability. *Completeness* (in this setting) has to do with axiomatising validity for classes of transition systems: ϕ is *valid* for the class $\mathcal{K}$ if for any $\mathcal{T} \in \mathcal{K}$, any modal model based on $\mathcal{T}$, and any state s in that model, $s \models \phi$. Axiomatising validity (again very roughly) means giving axioms and rules of inference for deriving formulas; such an axiomatisation is sound and complete for some class if all and only those formulas valid for that class are derivable. A class is axiomatisable if there is a sound and complete axiomatisation for it, and an axiomatisation is complete if there is a class for which it is sound and complete. Not all classes are axiomatisable (for instance the class of irreflexive transition systems, i.e. transition systems for which $\forall s.s \not\rightarrow s$) and not all axiomatisations are complete (for instance Blok, in a landmark paper [3], shows that there are $2^{\aleph_0}$ modal logics satisfied by the same class of transition systems). *Definability* is a different issue: A class $\mathcal{K}$ is definable if there is a formula ϕ such that $\mathcal{K}$ is exactly the class of TS's for which ϕ is valid. A formula may define a class without giving a sound and complete axiomatisation for it, and (vice versa) a formula may (together with a few standard axioms and rules) give a sound and complete axiomatisation for a class without defining it.

These issues are quite different from the one of primary concern here, namely the power of formulas in a given logic to express properties relative to a given *model*. From this point of view modal logic is unsatisfactory, at least if one is interested in logics for expressing properties of the ongoing behaviour of programs as is the case in concurrency. Consider for instance the property "always q" of a state s:

$$\forall s' \text{ if } s \rightarrow^* s' \text{ then } s' \models q$$

where $\rightarrow^*$ is the usual reflexive, transitive closure of $\rightarrow$. This property is inexpressible in $\mathcal{F}(\Box)$, that is, there is no formula $\phi \in \mathcal{F}(\Box)$ such that for all s, $s \models \phi$ iff s has the property "always q".

Exercise 4.4 *(Easy)* Prove this last statement. The following may be helpful: The *modal depth* of a $\phi \in \mathcal{F}(\Box)$, $|\phi|$, is given by

$$\begin{aligned} |q| &= 0 \\ |\neg\phi| &= |\phi| \\ |\phi \wedge \psi| &= \max(|\phi|, |\psi|) \\ |\Box\phi| &= |\phi| + 1 \end{aligned}$$

The capability of ϕ to "look into" the future of s is bounded by $|\phi|$. But "always q" requires an unbounded capability.

At this point we then turn to runs and traces through models to look at properties that depend on behaviour which is infinite or not bounded into the future. But modal logic is not dismissed quite that easily, of course. Adding recursive definitions to modal logic, for instance, causes a dramatic change in its expressive power, and we shall have occasion to return to modal logic in these terms later. Secondly, for *sequential* programming languages where the transition relations $\xrightarrow{a}$ can be taken to be from initial to final states there is no need to look more than one step into the future, and modal logic is quite adequate. Hoare logic [27, 2] and dynamic logic [46, 33] are well-known examples of program logics for sequential programming languages based on modal logic.

5 Linear Time Temporal Logic

One way of increasing the expressiveness of modal logic is to change the emphasis from *states* to runs, or traces through model. This can be done by a process of "unravelling". This is easiest when models are *total*, i.e. when for every s there is some a and s' s.t. $s \xrightarrow{a} s'$, so that attention can be restricted to infinite runs. This may be achieved by simply adding an auxillary state (and label, if necessary) to stand for termination.

Proviso 5.1 *From now on, unless otherwise stated, all models and TS's are assumed to be total.*

Definition 5.2 *(Runs)* A run through the model $\mathcal{T} = (S, \{\xrightarrow{a}\}_{a\in\Sigma}, V)$ is an ω-sequence $r = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots$. For such an r, and $i \in \omega$

$r(i) = s_i$ is the i'th state

$r^i = s_i \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{a_{i+2}} \cdots$ is the i'th suffix of r (and a run itself)

$\lambda(r, i) = a_i$ is the i'th label, $i > 0$

If models are not total we will have, in general, to consider both finite and infinite runs.

Linear time temporal logics are logics of runs (through models). A property such as "always q" can easily be interpreted as a modal property of runs. For instance, from a modal model $\mathcal{T} = (S, \rightarrow, V)$ we can derive the set of models $\{\mathcal{T}(r) \mid r$ a run through $\mathcal{T}\}$ where the states of $\mathcal{T}(r)$ is the set of suffixes of r, $V_{\mathcal{T}(r)}(q) = \{r \mid r(0) \in V(q)\}$, and $r \rightarrow r'$ if and only if r' is a suffix of r. Then "always q" is rendered as just $\Box q$ with respect to the set of models $\mathcal{T}(r)$.

However, it is important (for both conceptual and technical reasons) to maintain the distinction between models and runs through them, and we therefore introduce the specialised operator G in place of $\Box$. In fact we shall also refine the language of formulas somewhat by introducing nexttime operators $a.$ stating properties of single transitions. Thus the language of linear time temporal formulas $\phi, \psi \in \mathcal{F}(\bigcirc, G)$ is determined as follows:

$$\phi ::= q \mid \neg\phi \mid \phi \wedge \phi \mid a.\phi \mid G\phi$$

The linear time correlate of $\Diamond$ is derived by $F\phi \overset{\Delta}{=} \neg G \neg \phi$. In the unlabelled case (which is the more usual one) we use $\bigcirc$ instead of $a.$.

Definition 5.3 *(Semantics of $\mathcal{F}(\bigcirc, G)$)* The relation $r \models \phi$ for $\phi \in \mathcal{F}(\bigcirc, G)$ is determined by:

$r \models q$ iff $r(0) \in V(q)$

$r \models \neg\phi$ iff $r \not\models \phi$

$r \models \phi \wedge \psi$ iff $r \models \phi$ and $r \models \psi$

$r \models a.\phi$ iff $\lambda(r, 1) = a$ and $r^1 \models \phi$

$r \models G\phi$ iff for all $i \in \omega$, $r^i \models \phi$

Then $s \models \phi \in \mathcal{F}(\bigcirc, G)$ iff for all runs from s (i.e. with $r(0) = s$), $r \models \phi$.

So F expresses "eventually", and as a property of states, e.g. "always q" is rendered simply as Gq.

Remark 5.4 Having the nexttime operators around is quite natural in the labelled case where individual transitions are often regarded as observable, as in the case of CCS. In the unlabelled case this issue is somewhat more controversial, since one might want formulas to be preserved under stuttering, as in Lamport's Temporal Logic of Actions (c.f. [35]). This means that if $r(0) = r(1)$ then it ought to be the case for all ϕ that $r \models \phi$ if and only if $r^1(\phi)$ which it is not in general with $\bigcirc$ around.

Exercise 5.5 *(Very easy)* $\mathcal{F}(\bigcirc, G)$ is not *compact* (in the sense of 1st order logic). Why?

6 Expressiveness of $\mathcal{F}(\bigcirc, G)$

It is instructive at this stage to gain some initial familiarity with the type of properties one can express in linear time temporal logic. We give a few examples based on [20].

Example 6.1 *(Partial correctness)* Let $\surd$ be an atomic proposition holding for a state just in case it is terminal. The formula

$$q_1 \supset G(\surd \supset q_2)$$

expresses the property that whenever the precondition q_1 holds of s and s' is a terminal state reachable from s then the postcondition q_2 holds of s'. This is an example of an *invariance*, or *safety*, property.

Example 6.2 *(Total correctness)* Similarly

$$q_1 \supset F(\surd \wedge q_2)$$

expresses that any computation starting in a state satisfying q_1 terminates in a state satisfying q_2. This is an example of an *eventuality*, or *liveness* property.

Example 6.3 *(Fairness)* The notion of fairness comes in many flavours. One very weak notion asserts that a continuous holding of a request q_1 eventually forces a response q_2: $Gq_1 \supset Fq_2$. A stronger version states that if a resource is infinitely often requested (q_1) then it is eventually granted (q_2): $GFq_1 \supset Fq_2$.

Exercise 6.4 *(Easy)*

1. A slight strengthening of (the first part of) 6.3 could be that if a request holds almost always then a response will eventually occur. Express this.

2. Determine induction principles for G and F.

7 Linear Time Temporal Logic with Until

Is $\mathcal{F}(\bigcirc, G)$ expressive enough? One way of answering this question in the negative is to give examples of properties that we should clearly like to express which are not expressible in $\mathcal{F}(\bigcirc, G)$.

We introduce a new, binary operator U with the semantics

$$r \models \phi U \psi \text{ iff } \exists i \in \omega \,.\, r^i \models \psi \text{ and } \forall j : 0 \leq j < i \,.\, r^j \models \phi$$

and let $\mathcal{F}(\bigcirc, U)$ be the language(s) with the (labelled) nexttime operator(s) and U. Note that F and G are definable by

$$\begin{aligned} F\phi &\triangleq \top U \phi \\ G\phi &\triangleq \neg F \neg \phi = \neg(\top U \neg \phi) \end{aligned}$$

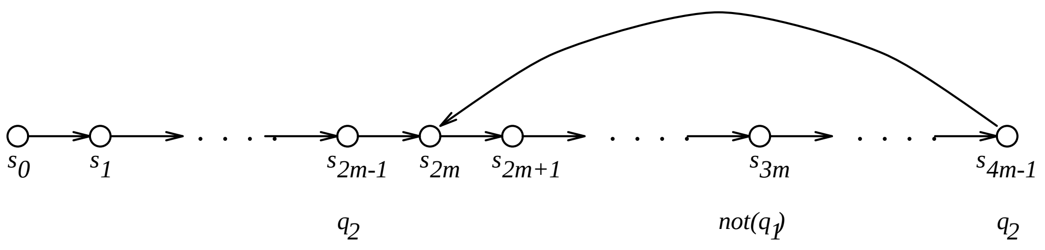

Figure 2: Model for proof of Theorem 7.3

Example 7.1 *(Precedence and strict fairness)*

1. Precedence: The property that q_2 will never hold unless preceded by q_1 is expressible by
$$\Pr(q_1, q_2) \triangleq Fq_2 \supset (\neg q_2 U q_1)$$
2. Strict fairness: We can now express a FIFO discipline like "if q_1 precedes q_2 then q_1' precedes q_2'" by
$$\Pr(q_1, q_2) \supset \Pr(q_1', q_2')$$

Exercise 7.2 (Recommended) Express the following property: Every occurrence of q_2 is preceded by a q_1 which happened after the last q_2, if any.

We already know that $\mathcal{F}(\bigcirc, U)$ is at least as expressive as $\mathcal{F}(\bigcirc, G)$. In fact:

Theorem 7.3 *(Kamp [32]) $\mathcal{F}(\bigcirc, U)$ is strictly more expressive than $\mathcal{F}(\bigcirc, G)$.*

PROOF: The proof is a modification of a proof by Stirling [51]. Consider the (unlabelled) model of fig. 2. The figure is to be interpreted thus: The model has states $s_0, \ldots, s_{4m-1}$, and the transition relation has the property that $s_i \rightarrow s_j$ if and only if either $j = i + 1$ or $j = 4m \perp 1$ and $i = 2m$. The variable q_1 is assumed to hold for all states except s_{3m}, and q_2 is assumed to hold for just the states s_{2m-1} and s_{4m-1}. Let r be the uniquely determined run starting in state s_0. We show the following:

Claim 7.4 *For $0 < i \leq m$ and all $\phi \in \mathcal{F}(\bigcirc, G)$ containing fewer than i occurrences of $\bigcirc$,*
$$r^{m-i} \models \phi \textit{ iff } r^{3m-i} \models \phi$$

Given 7.4 the proof is complete since $r \models q_1 U q_2$ and $r^{2m} \not\models q_1 U q_2$.

The claim is proved by induction in the structure of ϕ. The difficult case is for G. We prove yet another little Lemma, stating that as long as the number of occurrences of $\bigcirc$ in ϕ is strictly less than i and $j \geq i$ then $r^{2m-i} \models \phi$ if and only if $r^{2m-j} \models \phi$.

Exercise 7.5 Prove the last Lemma and, using it, prove Claim 7.4.

□

8 The First-order Theory of Linear Order

Is $\mathcal{F}(\bigcirc, U)$ expressive enough, or can we (and should we) do better? The question is unclear: How can our answers be measured? Here are a few possibilities:

1. At the outset our motivation was properties of interest for computing systems. If we can establish an important property of such systems not expressible in $\mathcal{F}(\bigcirc, U)$ then the issue is clearly settled in the negative. But what if we can not?
2. Then some other criteria must be found. Leave personal preference/elegance of expression aside. These are clearly important features, but macros may be able to deal with those to a large extent. How about decidability (this is in fact a very important one to many people)? Or, if decidable, the (upper, lower bound of the) complexity of the decision procedure (an important one too). But these criteria are still negative.
3. One half (the technical one) of the solution is to compare the expressive power of $\mathcal{F}(\bigcirc, U)$ (or whatever other temporal logic we may be interested in) with those of other important theories "talking about" the same kinds of structures.

For $\mathcal{F}(\bigcirc, U)$ a very strong result would be to show that it is exactly as expressive as its meta-language. We then say that, with respect to this meta-language, $\mathcal{F}(\bigcirc, U)$ is *expressively complete*. For $\mathcal{F}(\bigcirc, U)$, unlike the situation for many other temporal logics, the meta-language is quite easily identified. It is the first-order language $\mathcal{F}_1(<)$ of the natural numbers under $<$. Formulas Φ, Ψ in this language are generated by:

$$\Phi ::= q(x) \mid x_1 = x_2 \mid x_1 < x_2 \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists x.\Phi$$

where x ranges over individual variables and q over primitive unary predicates (which we do not distinguish from the atomic propositions of the object logics). Where confusion may otherwise result we use $\doteq$ in place of the syntactical equality symbol $=$. Standard abbreviations such that $\vee$, $\supset$, and $\forall$ apply. For clarity we generally use Φ, Ψ to range over formulas in the meta-languages, and reserve ϕ, ψ to range over formulas in the object logics.

We assume familiarity with basic notions of first-order logic such as free variables, alpha-conversion, substitution, structure. In particular, $\Phi(x_1, \ldots, x_m)$ is the formula Φ with free variables among $x_1, \ldots, x_m$.

Definition 8.1 *(Model, truth, equivalence)* A *model* $\mathcal{M}$ for $\mathcal{F}_1(<)$ is a structure $(\omega, <, V)$ such that $(\omega, <)$ is the natural numbers under the strict less-than ordering, and for each $q \in Q$, $V(q) \subseteq \omega$. The relation $\mathcal{M} \models_\varepsilon \Phi$ where ε is a mapping assigning a natural number $\varepsilon(n)$ to each individual variable x, is defined by induction in the structure of Φ:

$\mathcal{M} \models_\varepsilon q(x)$ iff $\varepsilon(x) \in V(q)$

$\mathcal{M} \models_\varepsilon x_1 = x_2$ iff $\varepsilon(x_1) = \varepsilon(x_2)$

$\mathcal{M} \models_\varepsilon x_1 < x_2$ iff $\varepsilon(x_1) < \varepsilon(x_2)$

$\mathcal{M} \models_\varepsilon \neg\Phi$ iff not $\mathcal{M} \models_\varepsilon \Phi$

$\mathcal{M} \models_\varepsilon \Phi \wedge \Psi$ iff $\mathcal{M} \models_\varepsilon \Phi$ and $\mathcal{M} \models \Psi$

$\mathcal{M} \models_\varepsilon \exists x.\Phi$ iff exists n such that $\mathcal{M} \models_{\varepsilon[x\mapsto n]} \Phi$

In the last clause, $\varepsilon[x \mapsto n]$ is the familiar update of ε acting like ε for arguments distinct from x and returning n otherwise. If $\Phi = \Phi(x_1, \ldots, x_m)$ and $\mathcal{M} \models_\varepsilon \Phi$ then we say that Φ is *true in the model* $\mathcal{M}$ *at instances* $\varepsilon(x_1), \ldots, \varepsilon(x_m)$. Formulas Φ and Ψ are *equivalent* (with respect to $(\omega, <)$ if for all V and ε, $(\omega, <, V) \models_\varepsilon \Phi$ iff $(\omega, <, V) \models_\varepsilon \Psi$; in symbols, $\Phi \equiv \Psi$.

When discussing $\mathcal{F}_1(<)$-models we sometimes refer to an $n \in \omega$ as a *point* instead of a state. Also, because of the nature of $\mathcal{F}_1(<)$-models, confusion will rarely result from identifying the point n with the run $n, n+1, \ldots$. We give a few example properties expressible in $\mathcal{F}_1(<)$.

Example 8.2 *(Properties expressible in $\mathcal{F}_1(<)$)* We view an $\mathcal{F}_1(<)$-model $\mathcal{M} = (\omega, <, V)$ as determining the underlying $\mathcal{F}(\bigcirc, G)$-model $(\omega, <, V)$. Thus formulas in $\mathcal{F}_1(<)$ and $\mathcal{F}(\bigcirc, G)$ can be compared by comparing their capacity for describing properties of $(\omega, <, V)$.

1. $\Phi(x) \wedge \forall y.\neg(y < x)$ expresses that $\Phi(x)$ is true (in $(\omega, <, V)$) at instance 0. We abbreviate $\Phi(x) \wedge \forall y.\neg(y < x)$ as $\Phi(0)$.

2. $x < y \wedge \neg\exists z.x < z \wedge z < y$ expresses that y is the *successor* of x.

3. $\exists y.\Phi(y) \wedge x < y \wedge \neg\exists z.x < z \wedge z < y$ expresses that $\Phi(x')$ is true at point $x+1$. We abbreviate this as $\Phi(\mathrm{succ}(x))$.

4. $q(x)$ expresses that q is true at the x'th point.

5. $x < y \vee x = y$ expresses that y is greater than or equal to x. This is abbreviated $x \leq y$.

6. $\forall y.x \leq y \supset \Phi(y)$ expresses that $\Phi(y)$ is true at any point y greater than or equal to x.

7. $\exists y.x \leq y \wedge \Psi(y) \wedge \forall z.0 \leq z \wedge z < y \supset \Phi(z)$ expresses that, as viewed from the x'th point, $\Phi(z)$ is true until $\Psi(y)$.

In fact we obtain

Theorem 8.3 *For any $\mathcal{F}(\bigcirc, U)$-formula ϕ there is an $\mathcal{F}_1(<)$-formula $\phi^\dagger(x)$ such that for all $m \in \omega$,*

$$\omega^m \models \phi \textit{ iff } \phi^\dagger(x) \textit{ is true at } m$$

(where we (ambiguously) identify ω with the infinite run $0 < 1 < \cdots$)

PROOF: Most of the cases are dealt with in example 8.2. If in doubt complete the proof yourself. □

This Theorem is verifying the intuitive insigth obtained from just inspecting the definition of the satisfaction relations that $\mathcal{F}_1(<)$ can indeed be viewed as the meta-language for $\mathcal{F}(\bigcirc, U)$. This is not a deep insight. Much deeper (and rather surprising, perhaps) is the existence of a translation the other way round. In a slightly different setting (of both past and future modalities) this result is due to Kamp ([32]). Gabbay et al [20] modified Kamp's result to the present setting of future time modalities only, and we follow (by and large) their proof.

Remark 8.4 There are other ways of establishing related results. Gabbay [19] shows that there i a finite basis (i.e. that a finite number of temporal connectives suffice) for a given time structure if and only if any first-order formula over that structure is equivalent to one with a bounded number of bounded variables. Note that this is a corollary of the present result. Immerman and Kozen [31] shows this boundedness condition to hold for linear orders (with bound 3).

We prove the following Theorem.

Theorem 8.5 *(Expressive completeness of $\mathcal{F}(\bigcirc, U)$) For every formula $\Phi(x)$ of $\mathcal{F}_1(<)$ there is an $\mathcal{F}(\bigcirc, U)$-formula ϕ s.t. $\Phi(x)$ is true at 0 if and only if $\omega \models \phi$.*

The restriction to the point 0 is evidently needed. This restriction can be lifted by restricting attention to future formulas.

Definition 8.6 *(Future formulas)* An $\mathcal{F}_1(<)$-formula $\Phi(x)$ is a *future formula* if all quantifiers are bounded backwards by x, i.e.

1. whenever $\forall y.\Psi$ is a subformula of $\Phi(x)$ then Ψ has the form $x < y \supset \Psi'$, and
2. whenever $\exists y.\Psi$ is a subformula of $\Phi(x)$ then Ψ has the form $x < y \wedge \Psi'$.

Theorem 8.7 *For all future formulas $\Phi(x)$ there is an $\mathcal{F}(\bigcirc, U)$-formula ϕ s.t. for all $m \in \omega$,*

$$\omega^m \models \phi \textit{ if and only if } \Phi(x) \textit{ is true at } m$$

We prove this Theorem by successive rewritings of future formulas into normal forms until a point is reached where the translation into $\mathcal{F}(\bigcirc, U)$ is immediate.

9 Special Formulas

The first step is to rewrite any $\mathcal{F}_1(<)$-formula into a form which allows attention later to be restricted to formulas $\Phi(x,y)$ which depend only on the interval between x and y (allowing $y=\infty$ and $x=\bot\infty$; these are used only for abbreviations and can easily be eliminated). The special forms are the sets $\mathrm{At}(x)$, $\mathrm{Bet}(x,y)$, $\mathrm{Bef}(x)$ and $\mathrm{Aft}(x)$:

$$\mathrm{At}(x) = \{(\neg)q(x) \mid q \in Q\}$$

where the $(\neg)q(x)$ means the $\neg$ is optional. The remaining sets are defined inductively:

$$\begin{aligned}
\mathrm{Bet}_0(x,y) &= \emptyset \\
\mathrm{Bet}_{n+1}(x,y) &= \{(\neg)\exists z.x<z<y \wedge \Phi_1 \wedge \cdots \wedge \Phi_m \mid \\
&\quad\ \forall i.\Phi_i \in \mathrm{Bet}_n(x,z) \cup \mathrm{At}(z) \cup \\
&\quad\ \mathrm{Bet}_n(z,y)\}
\end{aligned}$$

In the limit:

$$\begin{aligned}
\mathrm{Bet}(x,y) &= \bigcup_{n\in\omega} \mathrm{Bet}_n(x,y) \\
\mathrm{Bef}(x) &= \mathrm{Bet}(\bot\infty, x) \\
\mathrm{Aft}(x) &= \mathrm{Bet}(x,\infty)
\end{aligned}$$

Definition 9.1 *(Sequencing formula, special formula)*

1. The formula π is a *sequencing formula* of $x_1,\ldots,x_m$ if π has the form $x'_1R_1\cdots R_{m-1}x'_m$ where each R_i is either $\doteq$ or $<$ and $x'_1,\ldots,x'_m$ is a permutation of $x_1,\ldots,x_m$. Then π *sequences* $x_1,\ldots,x_m$ *as* $x'_1,\ldots,x'_m$.

2. The formula $\Phi(x_1,\ldots,x_m)$ is *special* if it has the form $\pi \wedge \Phi_1 \wedge \cdots \wedge \Phi_k$, where π is a sequencing formula of $x_1,\ldots,x_m$, π sequences $x_1,\ldots,x_m$ as $x'_1,\ldots,x'_m$, say, and each Φ_i is in one of $\mathrm{Bef}(x'_1)$, $\mathrm{At}(x'_1)$, $\mathrm{Bet}(x'_1,x'_2)$, $\mathrm{At}(x'_2)$, …, $\mathrm{Bet}(x'_{m-1},x'_m)$, $\mathrm{At}(x'_m)$, $\mathrm{Aft}(x'_m)$.

Lemma 9.2 *Every $\mathcal{F}_1(<)$-formula $\Phi(x_1,\ldots,x_m)$ is equivalent to a formula of the form*

$$\Phi_1(x_1,\ldots,x_m) \vee \cdots \vee \Phi_k(x_1,\ldots,x_m)$$

with each $\Phi_i(x_1,\ldots,x_m)$ a special formula.

PROOF: By induction in the structure of Φ.
(i) $\Phi = x_i \doteq x_j$. In this case

$$\begin{aligned}
\Phi \equiv\ & \bigvee\{\pi(x_1,\ldots,x_m) \mid \pi(x_1,\ldots,x_m) \text{ special, and} \\
& \pi(x_1,\ldots,x_m) \text{ and } \Phi \text{ are consistent}\}.
\end{aligned}$$

Here we say that Ψ_1 and Ψ_2 are *consistent* if the conjunction $\Psi_1 \wedge \Psi_2$ has a model.
(ii) $\Phi = \neg\Phi'$. By the induction hypothesis Φ is equivalent to a formula of the form $\neg(\Phi'_1 \vee \cdots \vee \Phi'_k)$ where each Φ'_i is a special formula, of the form, say,

$$\Phi'_i = \pi_i \wedge \Phi'_{i,1} \wedge \cdots \wedge \Phi'_{i,n_i}$$

with each $\Phi'_{i,j}$ a member of the sets Bef, At, etc. as specified. Then

$$\Phi \equiv \bigwedge\{\neg\pi_i \vee \bigvee\{\neg\Phi'_{i,j} \mid 1 \leq j \leq n_i\} \mid 1 \leq i \leq k\}$$

The rewriting is completed by replacing each $\neg\pi_i$ by the disjunction of all π' s.t. $\neg\pi_i$ and π' are consistent and then using the distributive law to rewrite into the desired format.

Exercise 9.3 Fill in the details.

(iii) $\Phi \equiv \exists x.\Phi'$. By the induction hypothesis Φ is equivalent to a formula of the form $\exists x. \bigvee\{\Phi'_i \mid 1 \leq i \leq k\}$ with each disjunctive clause of the form

$$\Phi'_i = \pi_i \wedge \bigwedge\{\Phi'_{i,j} \mid 1 \leq j \leq n_i\}$$

where x may occur freely in the π_i, $\Phi'_{i,j}$. The existential quantifier distributes over $\vee$, so

$$\Phi \equiv \bigvee\{\exists x.\pi_i \wedge \bigwedge\{\Phi'_{i,j} \mid 1 \leq j \leq n_i\} \mid 1 \leq i \leq k\}$$

and it suffices to rewrite each disjunctive clause.
First for the sequencing formulas: Let

$$\pi_i = x'_1 R_1 \cdots R_m x'_{m+1}$$

and $x = x'_l$. Suppose for simplicity that $1 < l < m+1$. The case where one of R_{l-1}, R_l (one is always defined) is $\doteq$ is left as an **exercise**, so assume both are $<$. Then $\pi'_i = x'_1 R_1 \cdots R_{l-2} x'_{l-1} < x'_{l+1} R_{l+1} \cdots R_m x'_{m+1}$ is a sequencing formula as desired.
Next for the $\Phi'_{i,j}$: We can obtain the form

$$\begin{aligned} \exists x.\Phi'_i \quad \equiv \quad & \pi'_i \wedge \Phi''_i \wedge \exists x'_l . x'_{l-1} < x'_l < x'_{l+1} \wedge \\ & \bigwedge\{\Phi'_{i,j} \mid \Phi'_{i,j} \in \mathrm{Bet}(x'_{l-1}, x'_l) \cup \mathrm{At}(x'_l) \cup \mathrm{Bet}(x'_l, x'_{l+1})\} \end{aligned}$$

where Φ''_i is the conjunction of all those conjuncts of Φ'_i not in any of the $\mathrm{Bet}(y,z)$, $\mathrm{At}(y)$ for y or z equal to x'_l.

The result follows if we can show $\mathrm{Bet}_n(y,z)$ *cumulative*, i.e. that $\mathrm{Bet}_n(y,z) \subseteq \mathrm{Bet}_{n+1}(y,z)$.

Exercise 9.4 *(Easy)* Show this.

It follows that

$$\exists x'_l . x'_{l-1} < x'_l < x'_{l+1} \wedge \bigwedge\{\Phi'_{i,j} \mid \Phi'_{i,j} \in \mathrm{Bet}(x'_{l-1}, x'_l) \cup \mathrm{At}(x'_l) \cup \mathrm{Bet}(x'_l, x'_{l+1})\} \\ \in \mathrm{Bet}(x'_{l-1}, x'_{l+1})$$

and the case is completed.

Exercise 9.5 Do the cases for $\Phi = x_i < x_j$, $\Phi = q(x)$, $\Phi = \Phi_1 \wedge \Phi_2$.

□

10 Decomposition Formulas

We continue to the second intermediate stage of the rewriting.

To complete the proof we need only consider special formulas of one free variable—i.e. a formula of the form $\pi \wedge \Phi_1 \wedge \cdots \wedge \Phi_k$ where each Φ_i is in either $\mathrm{At}(x)$ or $\mathrm{Aft}(x)$. In this case π is vacuous, and any member of $\mathrm{At}(x)$ is trivially translatable into $\mathcal{F}(\bigcirc, U)$. It therefore suffices to consider formulas in $\mathrm{Bet}(x, y)$ where y may be ∞. This is the purpose of decomposition formulas.

Definition 10.1 *(Decomposition Formulas)* A *Decomposition formula* $\delta(x, y)$ is any formula of the form $x = y$ or

$$x < y \wedge \exists z_0, \ldots, z_n . (x = z_0 < \cdots < z_n = y) \wedge \\ \bigwedge\{\phi_i^\dagger(z_i) \mid 1 \leq i < n\} \wedge \\ \bigwedge\{\forall u . (z_{i-1} < u < z_i) \supset \psi_i^\dagger(u)\}$$

and in case $y = \infty$ we may also have conjuncts of the form

$$\forall u . \exists v > u . \gamma_i^\dagger(v)$$

where the ϕ_i, ψ_i, γ_i are all members of $\mathcal{F}(\bigcirc, U)$.

The normal form Lemma we are looking for is the following.

Lemma 10.2 *(Decomposition) Every formula in Bet(x, y) is equivalent to a finite disjunction of decomposition formulas.*

Exercise 10.3 *(Easy)* Establish the expressive completeness result from the Decomposition Lemma.

Proof: (Of decomposition lemma). If $\Phi \in \mathrm{Bet}(x,y)$ then $\Phi \in \mathrm{Bet}_n(x,y)$ for some n. We prove the lemma by induction in n.

The base case is trivial.

So let $n = n' + 1$. Then

$$\Phi = (\neg)\exists z.(x < z < y \wedge \Phi_1 \wedge \cdots \wedge \Phi_m)$$

where, say,

$$\begin{array}{rcl} \Phi_1, \ldots, \Phi_{m_1} & \in & \mathrm{Bet}_{n'}(x,z) \\ \Phi_{m_1+1}, \ldots, \Phi_{m_2} & \in & \mathrm{At}(z) \\ \Phi_{m_2+1}, \ldots, \Phi_{m_3} & \in & \mathrm{Bet}_{n'}(z,y) \end{array}$$

By the induction hypothesis every Φ_i can be rewritten into a disjunction of decomposition formulas, and—using distribution of $\vee$ over $\wedge$ and $\exists$ over $\vee$—we can therefore rewrite Φ into the form

$$\Phi \equiv (\neg)\bigvee \exists z.(x < z < y \wedge \Phi'_1 \wedge \cdots \wedge \Phi'_m)$$

with each Φ'_i in either $\delta(x,z)$, $\mathrm{At}(z)$ or $\delta(z,y)$, as appropriate.

Exercise 10.4 *(Easy)* Show that each

$$\exists z.(x < z < y \wedge \Phi'_1 \wedge \cdots \wedge \Phi'_m)$$

can be rewritten into an equivalent decomposition formula.

We thus have rewritten Φ into the form

$$\Phi \equiv (\neg)\bigvee_i \Phi_i$$

where each Φ_i is a decomposition formula. If the $\neg$ is not present the proof is done. So assume it is. Then $\Phi \equiv \bigwedge_i \neg\Phi_i$.

Exercise 10.5 *(Recommended)* Show that $\bigwedge_j \Phi_j$ is equivalent to a decomposition formula if each Φ_j is.

So the remaining part of the proof boils down to the following key Lemma. Note, by the way, that the proof given here differs from the sketch given in [20].

Lemma 10.6 *If Φ is a decomposition formula then $\neg\Phi$ is equivalent to a disjunction of decomposition formulas.*

Proof: For the case $y \neq \infty$ it suffices to consider formulas of the form

$$\begin{array}{rcl} \Phi & = & x < y \wedge \forall z_0, \ldots, z_n.(x = z_0 < \cdots < z_n = y) \supset \\ & & \bigvee_{i=1}^{n-1} \phi_i^{\dagger}(z_i) \vee \bigvee_{i=1}^{n} \exists u.(z_{i-1} < u < z_i \wedge \psi_i^{\dagger}(u)) \end{array}$$

Exercise 10.7 Why?

Suppose first that

$$\Phi \equiv x < y \wedge \forall z_0, \ldots, z_n.(x = z_0 < \cdots < z_n = y) \supset \bigvee_{i=1}^{n-1} \phi_i^\dagger(z_i) \qquad (1)$$

Then Φ is equivalent to a disjunction $\Phi_1 \vee \Phi_2$ where Φ_1 is the disjunction of formulas

$$\begin{array}{l} x < y \wedge \exists z_0, \ldots, z_n.(x = z_0 < \cdots < z_n = y) \wedge \\ \bigwedge_{i=1}^{n-1} \Phi_i'(z_i) \wedge \bigwedge_{i=1}^{n} \forall u.(z_{i-1} < u < z_i) \supset \Phi_i''(u) \end{array} \qquad (2)$$

and Φ_2 is the disjunction of formulas

$$x < y \wedge \exists z_0, \ldots, z_m.(x = z_0 < \cdots < z_m = y) \wedge \forall u.(z_{i-1} < u < z_i) \supset \perp$$

whenever $0 < m < n$.

For Φ_1 we require that (Φ_i', Φ_i'') is identical to either $(\phi_i^\dagger, \phi_i^\dagger)$ or $(\top, \perp)$, and that not all (Φ_i', Φ_i'') are $(\top, \perp)$. Both Φ_1 and Φ_2 are clearly disjunctions of decomposition formulas.

Exercise 10.8 *(Recommended)* Show that $\Phi_1 \vee \Phi_2$ implies Φ.

For the other direction we assume that (1) holds. Write this as the sequence

$$(\phi_1^\dagger, \phi_1^\dagger) \cdots (\phi_n^\dagger, \phi_n^\dagger).$$

Whenever $\phi_i^\dagger$ is not required for (1) to hold replace the pair $(\phi_i^\dagger, \phi_i^\dagger)$ by $(\top, \perp)$. This gives a configuration of pairs (Φ_i', Φ_i''). A formula of the form (2) results. If this formula is not true (for given x and y) and $x < y$ then

$$\begin{array}{c} \forall z_0, \ldots, z_n.(x = z_0 < \cdots < z_n = y) \supset \\ \bigvee_{i=1}^{n-1} \neg\Phi_i'(z_i) \vee \bigvee_{i=1}^{n} \exists u.(z_{i-1} < u < z_i) \wedge \neg\Phi_i''(u) \end{array}$$

We thread a chain from x to y which refutes (1). Suppose we have picked z_{i-1}. If $\Phi_i'' = \perp$ then we let $z_i = z_{i-1} + 1$. But then $\neg\exists u.(z_{i-1} < u < z_i)$, a contradiction. Suppose on the other hand that $(\Phi_i', \Phi_i'') = (\phi_i^\dagger, \phi_i^\dagger)$. Then there is some z_i such that $\neg\phi_i^\dagger(z_i)$ which we choose and proceed.

So assume instead that (1) fails. Then Φ is equivalent to the disjunction for $1 \leq i \leq n$ of the following formulas (which we for readability write partly out in english): $x < y$ and there is a chain $x = z_0, \ldots, z_n = y$ and

$$\exists u.(z_{i-1} < u < z_i) \wedge \psi_i^\dagger(u)$$

and for all m_1, m_2 s.t. $n = m_1 + m_2 + 1$ and Φ_1 and Φ_2, where these last two formulas have the forms

$$\begin{aligned}
\Phi_1 &= x < u \wedge \forall z_0, \ldots, z_{m_1}.(x = z_0 < \cdots z_{m_1} = u) \supset \\
&\quad \bigvee \cdots \vee \bigvee \cdots \\
\Phi_2 &= u < y \wedge \forall z_0, \ldots, z_{m_2}.(u = z_0 < \cdots z_{m_2} = y) \supset \\
&\quad \bigvee \cdots \vee \bigvee \cdots
\end{aligned}$$

We can now use the induction hypothesis and rewrite Φ_1 and Φ_2 into a disjunction of decomposition formulas, and a simple rewriting then completes the proof.

Exercise 10.9 *(Easy)* Do this.

Exercise 10.10 *(For the non-quitters)* Do the proof for $y = \infty$.

This completes the proof of the Decomposition Lemma □

And thus also the Expressive Completeness Theorem. □

11 Expressiveness of $\mathcal{F}(\bigcirc, U)$ Revisited

So in a certain technically precise sense $\mathcal{F}(\bigcirc, U)$ is expressively complete. That is, it can express all "desirable" properties, where we understand "desirable" as be synonymous with "being expressible in the 1st order language of $(\omega, <)$ with unary predicate constants".

But, as Wolper [60] discovered, this does not mean that it is "really" expressively complete. Consider the CCS processes determined by the recursive process expressions

$$\begin{aligned}
X &= \mathit{check}.Y \\
Y &= \mathit{check}.X + \mathit{work}.X.
\end{aligned}$$

A simple synchronous parallel composition [2] $\|$ can be defined on CCS processes governed by the rule

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q'}{P \parallel Q \overset{\{a,b\}}{\rightarrow} P' \parallel Q'}$$

Consider the process $X \parallel Y$. The $\mathcal{F}(\bigcirc, G)$-formula

$$\{\mathit{check}\}.\top \vee \{\mathit{check}, \mathit{work}\}.\top$$

expresses that a parallel subprocess performs the action *check*. Clearly $X \parallel Y \models G(\{\mathit{check}\}.\top \vee \{\mathit{check}, \mathit{work}\}\top)$. This is true because $X \models \mathit{even}(\mathit{check}.\top)$ and

[2]The operation is only needed to make the example compact. A similar, but more lengthy, example can be made in CCS proper without much difficulty

$Y \models odd(check.\top)$ where $even\phi$ is the property that ϕ is true of all the even states (and possibly also elsewhere), and $odd\phi$, similarly, is the property that ϕ is true of all the odd states (and possibly also elsewhere). However, as Wolper showed, *even* and *odd* are not expressible in $\mathcal{F}(\bigcirc, U)$. This observation led Lichtenstein et al [38] to conclude that $\mathcal{F}(\bigcirc, U)$ is insufficient for modular reasoning.

To see that e.g. *even* is not expressible in $\mathcal{F}(\bigcirc, U)$, let r_n be any run with the property that whenever $i \neq n$ then $r_n^i \models q$, and $r_n^n \models \neg q$.

Theorem 11.1 *(Wolper) Let $\phi \in \mathcal{F}(\bigcirc, U)$. If ϕ has no more than n occurrences of $\bigcirc$ then for all $i, j > n$, $r_i \models \phi$ iff $r_j \models \phi$.*

Proof: By structural induction. Let $\phi = \phi_1 U \phi_2$ and suppose that $r_i \models \phi$. Then $r_i^l \models \phi_2$ for some $l \in \omega$ and whenever $0 \leq k < l$ then $r_i^k \models \phi_1$. Suppose first that $l \leq i$. Then $r_i^l = r_{i-l}$. If $i \perp l > n$ then $r_j \models \phi_2$ by the induction hypothesis, so also $r_j \models \phi$. If $i \perp l \leq n$ then notice that $r_j^{j-i+l} \models \phi_2$ and whenever $j \perp n \leq k < j \perp i + l$ then $r_j^k \models \phi_1$. It remains to show that $r_j^k \models \phi_1$ also when $0 \leq k < j \perp n$. But this follows from the induction hypothesis since $r_i^{i-(n+1)} \models \phi_1$. The case for $l > i$ is left to the reader.

Exercise 11.2 *(Easy)* Complete the proof.

□

Corollary 11.3 *The property $even(q)$ is not expressible in $\mathcal{F}(\bigcirc, U)$.*

Proof: For a contradicition suppose $\phi \in \mathcal{F}(\bigcirc, U)$ expresses this property, and let m be the number of occurrences of $\bigcirc$ in ϕ. Pick any k s.t. $2k \perp 1 > m$. Then $r_{2k} \models \phi$ iff $r_{2k-1} \models \phi$, and this is impossible. □

So can we extend $\mathcal{F}(\bigcirc, U)$ in a tractable way such that properties such as for instance $even(q)$ becomes expressible? The answer is yes, and that with a remarkable level of stability. This is summarised by the "equation":

$$\begin{aligned}
&\mathcal{F}(\bigcirc, G, \forall) \\
&\quad = \mathcal{F}(\text{Reg})(= \text{ETL}) \\
&\quad = \mathcal{F}(\bigcirc, \mu)(= \nu\text{TL}) \\
&\quad = S1S \\
&\quad = \omega\text{Reg} \\
&\quad = \{\omega \perp \text{languages accepted by Büchi automata}\} \\
&\quad = \cdots
\end{aligned}$$

These equations and the constructions that are involved in their proof form much of the core material of our subject area, and we shall therefore devote substantial attention to them in the next sections.

12 The Linear Time μ-calculus

We start our search for tractable extensions of $\mathcal{F}(\bigcirc, U)$ with the linear time μ-calculus. This logic is obtained, intuitively, by adding a facility for defining new operators by recursion. To see the rationale for this observe that our familiar operators G, F, and U satisfies the equations

$$\begin{aligned} G\phi &= \phi \wedge \bigcirc G\phi \\ F\phi &= \phi \vee \bigcirc F\phi \\ \phi U \psi &= \psi \vee (\phi \wedge \bigcirc(\phi U \psi)) \end{aligned}$$

Howewer, these equations do not determine these operators uniquely, for also

$$\begin{aligned} \bot &= \phi \wedge \bot \\ \top &= \phi \vee \top \\ (\phi U \psi) \vee G\phi &= \psi \vee (\phi \wedge \bigcirc((\phi U \psi) \vee G\phi)) \end{aligned}$$

However, if an equation $X = \phi(X)$ has the property that ϕ is monotone in X then we can guarantee that ϕ possesses a least and a greatest fixed point.

Exercise 12.1 *(Easy)* Show that $G\phi$ is the greatest solution of the equation $X = \phi \wedge \bigcirc X$ and that $F\phi$ is the least solution of the equation $X = \phi \vee \bigcirc X$. Which solution of the equation $X = \psi \vee (\phi \wedge \bigcirc X)$ is $\phi U \psi$? Give $even\phi$ as a solution of an equation.

The existence of least and greatest fixed points follows from the well-known Knaster-Tarski fixed point Theorem. It is worthwhile to actually prove this Theorem (in a form which is slightly specialised to fit our purpose). Let $\mathcal{P}(A)$ be the set of all subsets of the set A. A mapping $f : \mathcal{P}(A) \to \mathcal{P}(A)$ is *monotone* if whenever $A_1 \subseteq A_2 \subseteq A$ then $f(A_1) \subseteq f(A_2)$.

Theorem 12.2 *Any monotone mapping $f : \mathcal{P}(A) \to \mathcal{P}(A)$ possesses a least fixed point, μf, and a greatest fixed point, νf.*

PROOF: We give a characterisation of μf as the limit of an increasing chain of approximations. For each ordinal λ we define the set $f^{\uparrow,\lambda} \subseteq A$ by transfinite recursion by the clauses

$$\begin{aligned} f^{\uparrow,0} &= \emptyset \\ f^{\uparrow,\lambda+1} &= f(f^{\uparrow,\lambda}) \\ f^{\uparrow,\lambda} &= \bigcup_{\lambda'<\lambda} f^{\uparrow,\lambda'} && \text{for}\lambda \text{ a limit ordinal} \end{aligned}$$

Let then $f^{\uparrow} = \bigcup_{\lambda} f^{\uparrow,\lambda}$.

First, a transfinite induction shows that $f^{\uparrow,\lambda} \subseteq f(f^{\uparrow,\lambda})$ for all λ. For

$$\emptyset \subseteq f(\emptyset)$$

and for successor ordinals,

$$f^{\uparrow,\lambda'+1} = f(f^{\uparrow,\lambda'}) \subseteq f(f(f^{\uparrow,\lambda'})) = f(f^{\uparrow,\lambda'+1})$$

by the induction hypothesis and monotonicity of f, and finally for limit ordinals λ,

$$\begin{array}{rcll} f^{\uparrow,\lambda} & = & \bigcup_{\lambda'<\lambda} f^{\uparrow,\lambda'} & \\ & \subseteq & \bigcup_{\lambda'<\lambda} f(f^{\uparrow,\lambda}) & \text{By the induction hypothesis} \\ & \subseteq & f(\bigcup_{\lambda'<\lambda} f^{\uparrow,\lambda}) & \text{By monotonicity of } f \\ & = & f(f^{\uparrow,\lambda}). & \end{array}$$

It follows from this that if $\lambda' < \lambda$ then $f^{\uparrow,\lambda'} \subseteq f^{\uparrow,\lambda}$. Thus, since A is a set, there must be some ordinal λ_{fix} such that $f^{\uparrow,\lambda_{fix}+1} = f^{\uparrow,\lambda_{fix}}$. Clearly $f^{\uparrow} = f^{\uparrow,\lambda_{fix}}$, and $f^{\uparrow}$ is a fixed point of f. It remains to show it is the least fixed point. But another simple transfinite induction shows that if A' is any fixed point[3] then $f^{\uparrow,\lambda} \subseteq A'$. Thus also $f^{\uparrow} \subseteq A'$, and we are done.

The proof for the existence of greatest fixed points is entirely symmetric. For completeness we give the iterative construction of the fixed point:

$$\begin{array}{rcll} f^{\downarrow,0} & = & A & \\ f^{\downarrow,\lambda+1} & = & f(f^{\downarrow,\lambda}) & \\ f^{\downarrow,\lambda} & = & \bigcap_{\lambda'<\lambda} f^{\downarrow,\lambda'} & \text{for } \lambda \text{ a limit ordinal} \end{array}$$

□

From the proof of 12.2 we obtain the following

Corollary 12.3 *Let $f : \mathcal{P}(A) \to \mathcal{P}(A)$ be any monotone mapping on a set A. Then*

$$\begin{array}{rcccl} \mu f & = & f^{\uparrow} & = & \bigcap\{A' \subseteq A \mid f(A') \subseteq A'\} \\ \nu f & = & f^{\downarrow} & = & \bigcup\{A' \subseteq A \mid A' \subseteq f(A')\} \end{array}$$

□

We now turn to introducing a linear time temporal logic with a facility for recursive definitions of properties. Formulas $\phi, \psi \in \mathcal{F}(\bigcirc, \mu)$ are given by the grammar

$$\phi ::= q \mid X \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid a.\phi \mid \mu X.\phi$$

[3]It suffices to assume that A' is a *post fixed point* such that $f(A') \subseteq A'$

where X (Y and Z) range over propositional *variables*. We introduce the abbreviation $a : \phi$ as $\neg a.\neg\phi$.

The difference between variables X and atomic proposition symbols q is that semantically X ranges over arbitrary sets of *runs* whereas q ranges over arbitrary sets of *states*. This distinction becomes crucially important later, but not so in the present (linear time) setting. We therefore deal only with variables for the time being.

The intention is to interpret a formula $\phi(X_1, \ldots, X_n)$ as an n-ary mapping on sets of runs, and $\mu X.\phi(X_1, \ldots, X_n, X)$ as the operator that given sets of runs $A_1, \ldots, A_n$ returns the least fixed point of the mapping

$$\lambda A.\phi(A_1, \ldots, A_n, A)$$

using an intuitive notation. However, not all such mappings are monotone. A simple example is the mapping $\lambda A.\neg(A)$. Indeed $\lambda A.\neg(A)$ does not possess any fixed point at all, neither least nor greatest. The standard way of guaranteeing that mappings are monotone is to impose a *syntactic monotonicity condition* stating that for the formula $\mu X.\phi$ to be well-formed any occurrence of X in ϕ must be within the scope of an even number of negation symbols, and then extending well-formedness to arbitrary formulas ψ by requiring all subformulas of ψ of the form $\mu X.\phi$ to be well-formed.

Definition 12.4 *(Semantics of $\mathcal{F}(\bigcirc, \mu)$)* The set $\|\phi\|V$ for $\phi \in \mathcal{F}(\bigcirc, \mu)$ is determined by:

$$\|X\|V = V(X)$$

$$\|\neg\phi\|V = \overline{\|\phi\|V} = \{r \mid r \notin \|\phi\|V\}$$

$$\|\phi \wedge \psi\|V = \|\phi\|V \cap \|\psi\|V$$

$$\|a.\phi\|V = \{r \mid \lambda(r,1) = a \text{ and } r^1 \in \|\phi\|V\}$$

$$\|\mu X.\phi\|V = \bigcap\{A \mid \|\phi\|V[X \mapsto A] \subseteq A\}$$

Then the satisfaction relation is derived by $r \models \phi$ (with respect to the valuation V) if and only if $r \in \|\phi\|V$.

In case ϕ is closed (i.e. does not contain any free occurrences of propositional variables X) then we write $\|\phi\|$ in place of $\|\phi\|V$.

Exercise 12.5 *(Recommended)*

1. Show that whenever $\phi(X_1, \ldots, X_n)$ is well-formed then $\phi(X_1, \ldots, X_n)$ determines an n-ary mapping

$$\lambda A_1, \ldots, A_n.\|\phi\|V[X_1 \mapsto A_1] \cdots [X_n \mapsto A_n]$$

which is monotone in each argument. Conclude that $\|\phi\|V$ is well-defined.

2. Show that the greatest fixed point operator $\nu X.\phi$ may be defined by $\nu X.\phi \triangleq \neg\mu Y.\neg\phi[X \mapsto \neg Y]$. That is, show that with this definition the following condition is derived:

$$\|\nu X.\phi\|V = \bigcup\{A \mid A \subseteq \|\phi\|V[X \mapsto A]\}$$

   We sometimes use σ as a meta-variable ranging over $\{\mu, \nu\}$.

There are formulas that fail the syntactic monotonicity condition but still possesses fixed points, even least and greatest ones. The following examples are due to Banieqbal and Barringer.

Exercise 12.6 *(Recommended)* Consider the formulas

1. $Y \wedge (\bigcirc X) \wedge (\bigcirc \bigcirc \neg X)$
2. $Y \wedge (\bigcirc \neg X)$
3. $Y \wedge (\bigcirc \neg Y) \wedge (\bigcirc \neg X) \wedge (\bigcirc \bigcirc X)$
4. $Y \wedge X \wedge (\bigcirc \neg X)$

Show that (1) has a least and greatest solution $\bot$, that (2) has no solution, that the greatest solution of (3) exists and coincide with $\nu Y.Y \wedge (\bigcirc\neg X) \wedge (\bigcirc \bigcirc X)$, and that (4) has solutions but no greatest solution.

13 A Tableau System for $\mathcal{F}(\bigcirc, \mu)$

Even though it may seem very natural to describe temporal properties by extremal fixed points, the linear time μ-calculus (as other μ-calculi) is nonetheless difficult to use in practice. This is true, in particular, when fixed points alternate.

Example 13.1 What is the meaning of the formula $\nu X.\mu Y.(Z \wedge X) \vee \bigcirc Y$? What about $\mu Y.\nu X.(Z \vee \bigcirc Y) \wedge \bigcirc X$?

Furthermore, proving properties of this logic tends to get bogged down in details concerning variables, substitution and subformulas. It is helpful to develop some form of proof system for checking whether or not $r \models \phi$ that can bring out the intuition concerning least and greates fixed points, namely that

1. $r \models \nu X.\phi$ if and only if no contradiction is reached by assuming $r \models X$ where X is a solution of the equation $X = \phi$
2. $r \models \mu X.\phi$ if and only if $r \models X$ can be established from the assumption that X is a solution of the equation $X = \phi$

We present a proof system based on the tableau system for the socalled modal μ-calculus due to Stirling and Walker [52]. Closely related approaches are due to Streett and Emerson [53] and Vardi [57].

The key idea is to introduce *constants* U (V, W) to name occurrences of fixed point formulas. In particular, a constant is a *μ-constant* if it names a least fixed point formula. Then, to show $r \models \phi$ it suffices to check that free variables hold as appropriate on suffixes of r and that no μ-constant need be unfolded infinitely often along r. The mechanism that keeps track of the naming of fixed point formulas is a definition list.

Definition 13.2 *(Definition list)* A *definition list* is a sequence

$$\Delta = (U_1, \phi_1) \cdots (U_n, \phi_n)$$

such that $U_i \neq U_j$ whenever $i \neq j$ and each ϕ_i only mentions constants among $\{U_1, \ldots, U_{i-1}\}$.

Let Δ be as above. Then

1. $\mathrm{dom}\Delta = \{U_1, \ldots, U_n\}$
2. $\Delta(U_i) = \phi_i$
3. For $U \notin \mathrm{dom}\Delta$

$$\Delta \cdot (U = \phi) \triangleq (U_1, \phi_1) \cdots (U_n, \phi_n)(U, \phi)$$

4. Δ^* is the extension of Δ to arbitrary formulas which replace all constants in $\mathrm{dom}\Delta$ by their definitions

First, a *sequent* is an expression of the form $r \vdash_{\Delta,V} \phi$. These are the syntactic correlates of the satisfaction relation $r \in \|\Delta^*(\phi)\| V$. As usual for tableau systems the rules are presented as upside down proof rules, as a kind of refinement rules:

$$\frac{r \vdash_{\Delta,V} \neg\neg\phi}{r \vdash_{\Delta,V} \phi}$$

$$\frac{r \vdash_{\Delta,V} \phi \wedge \psi}{r \vdash_{\Delta,V} \phi \quad r \vdash_{\Delta,V} \psi}$$

$$\frac{r \vdash_{\Delta,V} \phi \vee \psi}{r \vdash_{\Delta,V} \phi} \qquad \frac{r \vdash_{\Delta,V} \phi \vee \psi}{r \vdash_{\Delta,V} \psi}$$

$$\frac{r \vdash_{\Delta,V} a.\phi}{r^1 \vdash_{\Delta,V} \phi} \quad \lambda(r,1) = a$$

$$\frac{r \vdash_{\Delta,V} a.\phi}{r \vdash_{\Delta,V} \bot} \quad \lambda(r,1) \neq a$$

$$\frac{r \vdash_{\Delta,V} a : \phi}{r^1 \vdash_{\Delta,V} \phi} \quad \lambda(r,1) = a$$

$$\frac{r \vdash_{\Delta,V} a : \phi}{r \vdash_{\Delta,V} \top} \quad \lambda(r,1) \neq a$$

$$\frac{r \vdash_{\Delta,V} \sigma X.\phi}{r \vdash_{\Delta\cdot(U=\sigma X.\phi),V} U} \quad U \notin \mathrm{dom}\Delta$$

$$\frac{r \vdash_{\Delta,V} U}{r \vdash_{\Delta,V} \phi[U/X]} \quad \Delta(U) = \sigma X.\phi$$

Definition 13.3 *(Tableau, true sequent, terminal sequent, partial and total success)*

1. A *tableau* τ is a maximal tree obtained from a root sequent $r_0 \vdash_{\Delta_0,V_0} \phi_0$ by downwards application of the tableau rules.

2. A sequent $r \vdash_{\Delta,V} \phi$ is *true*, if $r \in \|\Delta^*(\phi)\|V$.

3. An occurrence of a sequent $r \vdash_{\Delta,V} \phi$ in a tableau τ is *terminal* if no tableau rule is applied to it.

4. The tableau τ is *partially successful* if all its terminal sequents are true.

5. The tableau τ is *totally successful* if it is partially successful, and moreover there is no infinite path

$$\begin{array}{c} r_0 \vdash_{\Delta_0,V_0} \phi_0 \\ \hline \vdots \\ \hline r_i \vdash_{\Delta_i,V_i} \phi_i \\ \hline \vdots \end{array}$$

such that for infinitely many i, ϕ_i is the same μ-constant U.

Notice that, in relation to Definition 13.3.5, once $U \in \mathrm{dom}\Delta_i$, for all $j \geq i$, $\Delta_j(U) = \Delta_i(U)$. It is instructive to see why constants are actually needed in this construction. Consider an alternative tableau system that dispenses with constants in favour of the simpler unfolding rule:

$$\frac{r \vdash_V \sigma X.\phi}{r \vdash_V \phi[X \mapsto \sigma X.\phi]}$$

and then translating the condition 13.3.5 to: There is no infinite path such that for infinitely many i, ϕ_i is the same least fixed point formula $\mu X.\phi$, say. Unfortunately, while this works well for the non-alternating fragment (c.f. [36]) in general the resulting tableau system is incomplete.

Exercise 13.4 Show that with the simplified tableau system there is no totally successful tableau for any sequent of the form $r \vdash_V \nu X.\mu Y.((Z \wedge \bigcirc X) \vee \bigcirc Y)$. Show, on the other hand, that $\nu X.\mu Y.((Z \wedge \bigcirc X) \vee \bigcirc Y) = GFZ$.

We proceed to establish soundness and completeness of the proper tableau system. We first show a Lemma to the effect that each infinite path through a tableau is in a certain sense characterised by a unique constant, and then we proceed to show soundness and completeness.

Lemma 13.5 *Let π be any infinite path*

$$\begin{array}{c} r_0 \vdash_{\Delta_0, V_0} \phi_0 \\ \hline \vdots \\ \hline r_i \vdash_{\Delta_i, V_i} \phi_i \\ \hline \vdots \end{array}$$

through a tableau τ. Then there is a unique constant U s.t. $\phi_i = U$ for infinitely many i

Proof: We roughly follow [52]. Define the *degree* of ϕ, $d(\phi)$ in the following way:

$$d(X) = d(\neg X) = d(U) = 0$$

$$d(\neg\neg\phi) = d(a.\phi) = d(a : \phi) = d(\sigma X.\phi) = 1 + d(\phi)$$

$$d(\phi \wedge \psi) = d(\phi \vee \psi) = 1 + \max(d(\phi), d(\psi))$$

and then

$$d(r \vdash_{\Delta,V} \phi) = \begin{cases} d(\phi) & \text{if } \phi \text{ is not a constant} \\ d(\Delta(\phi)) & \text{otherwise} \end{cases}$$

Let π be an infinite path as above. The subsequence

$$\pi' = r'_0 \vdash_{\Delta'_0, V'_0} U_0, r'_1 \vdash_{\Delta'_1, V'_1} U_1, \cdots$$

consisting of those sequents of π with ϕ_i a constant is infinite.
Assume that no constant occurs infinitely often among $U_0, U_1, \ldots$.
Let i_0 be maximal s.t. $U_{i_0} = U_0$. Then

$$d(r'_{i_0+1} \vdash_{\Delta'_{i_0+1}, V'_{i_0+1}} U_{i_0+1}) < d(r'_0 \vdash_{\Delta'_0, V'_0} U_0)$$

for $\Delta'_{i_0+1}(U_{i_0+1})$ is a strict subformula of $\Delta'_0(U_0)$.
Let i_1 be maximal s.t. $U_{i_1} = U_{i_0+1}$. Similarly

$$d(r'_{i_1+1} \vdash_{\Delta'_{i_1+1}, V'_{i_1+1}} U_{i_1+1}) < d(r'_{i_0+1} \vdash_{\Delta'_{i_0+1}, V'_{i_0+1}} U_{i_0+1})$$

So some U must occur infinitely often among $U_0, U_1, \ldots$ as $d(r'_0 \vdash_{\Delta'_0, V'_0} U_0)$ is finite.

Exercise 13.6 *(Easy)* Show that U is unique.

$\square$

Theorem 13.7 *(Soundness) If $r \vdash_{\Delta,V} \phi$ has a successful tableau then it is true.*

Proof: Suppose τ is a successful tableau for $r \vdash_{\Delta,V} \phi$ and that $r \notin \|\Delta^*(\phi)\|V$. We build an infinite path for which no constant is unfolded infinitely often, contradicting the uniqueness Lemma 13.5.
If a node is false then one of its children is false too. Terminals, in particular, are true.
So we can trace a path in τ between $r \vdash_{\Delta,V} \phi$ and some sequent $r_1 \vdash_{\Delta_1,V_1} U_1$ for which:

(i) All nodes (endpoints including) are false.

(ii) U_1 is a ν-constant.

(iii) Any constant introduced strictly before U_1 fails to have this property.

For if no such U_1 can be reached some μ-constant will be unfolded infinitely often, and τ was assumed to be successful.
Using ordinal approximations we can find a minimal λ_1 s.t.

$$r_1 \notin \|\Delta_1^*(\nu X_1.\phi_1)\|^{\downarrow,\lambda_1} V_1$$

where $\Delta_1(U_1) = \nu X_1.\phi_1$.
Transform the subtableau τ_1 rooted in $r_1 \vdash_{\Delta_1,V_1} U_1$ of τ into a new tableau τ_1^* by indexing U_1 by α_1 at the root of τ_1^* and minimizing (and thus strictly decreasing) every time U_1 is unfolded.
If a node in τ_1^* is false then so is one of its children. For the only rule preventing this is the introduction rule for U_1 which is not applied.
Hence we can, as before, trace a path in τ_1 using only nodes that are false in τ_1^* from $r_1 \vdash_{\Delta_1,V_1} U_1$ to some other false node $r_2 \vdash_{\Delta_2,V_2} U_2$ s.t. U_2 is introduced strictly after U_1.
$r_2 \vdash_{\Delta_2,V_2} U_2$ is false in τ_1 too. Pick this path s.t. U_2 is introduced as early as possible.
Continuing ad infinitum builds the desired path. $\square$

Theorem 13.8 *(Completeness) If $r \vdash_{\Delta,V} \phi$ is true then it has a successful tableau.*

Proof: Suppose $r \in \|\Delta^*(\phi)\|V$, and let $U_1, \ldots, U_n$ be the sequence of μ-constants of Δ in the order of declaration.
For any sequence $w = \lambda_1, \ldots, \lambda_n, 0, 0, \ldots$, Δ_w is Δ with each entry $U_i = \mu X_i.\phi_i$ changed to $\phi_i^{\downarrow,\lambda_i}$ interpreted as the expected ordinal approximation.

The sequence

$$\iota(r \vdash_{\Delta,V} \phi) = \lambda_1, \ldots, \lambda_n, 0, 0, \ldots = w$$

is the *signature* of $r \vdash_{\Delta,V} \phi$, if w is lexicographically least s.t.

$$r \in \|\Delta^*_w(\phi)\| V$$

By always selecting the choice with least signature, $r \vdash_{\Delta,V} \phi$ can be extended to a partially successful tableau.

Exercise 13.9 *(Easy)* Prove this.

To see it is totally successful suppose there is a path from $r_1 \vdash_{\Delta_1,V_1} U$ to $r_2 \vdash_{\Delta_2,V_2} U$ in τ with U a μ-constant.
Then $w_1 = \iota(r_1 \vdash_{\Delta_1,V_1} U) > \iota(r_2 \vdash_{\Delta_2,V_2} U) = w_2$.
For the initial unfolding of U strictly decreases signature. Only the introduction of new μ-constants can increase signature, but this will not affect the original decrease, as the new constant is no longer *active* once we have reached $r_2 \vdash_{\Delta_2,V_2} U$. And the length of the non-zero prefix of w_2 is identical to the corresponding length of w_1. □

14 The Sequential Calculus

Corresponding to the expressive completeness result for $\mathcal{F}(\bigcirc, U)$ is there one to be obtained for $\mathcal{F}(\bigcirc, \mu)$? The answer is yes, and the appropriate notion of meta-language for $\mathcal{F}(\bigcirc, \mu)$ is S1S, the monadic second-order theory of successor, or as it is also known, the sequential calculus. The term "second-order" here means "having quantification over predicate symbols as well as individuals", and the term "monadic" restricts quantification to unary predicate symbols, i.e. sets.

The interest in this theory originates with Büchi [4] who showed it decidable, using automata on infinite strings. The decidability of other interesting theories, such as restricted first-order theories of the reals, are reducible to this theory, for instance by viewing runs as bitstrings. Indeed such reductions were the main motivating factor for looking at theories such as S1S in the first place. We return to these issues later.

We start, as usual, by introducing the language of S1S.

Definition 14.1 *($\mathcal{F}_2$(succ)) Terms* t of $\mathcal{F}_2$(succ) are given by

$$t ::= 0 \mid x \mid \mathrm{succ}(t)$$

Formulas $\phi \in \mathcal{F}_2(\mathrm{succ})$ are given by

$$\Phi ::= X(t) \mid t_1 \doteq t_2 \mid t < t \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\Phi \mid \exists X.\Phi$$

where x ranges over individual variables and X ranges over unary predicate variables (which we do not distinguish from the propositional variables of $\mathcal{F}(\bigcirc, \mu)$).

Models for $\mathcal{F}_2(\mathrm{succ})$ are the same as models for $\mathcal{F}_1(<)$. The only difference is that we have to interpret terms and set quantification. For terms the mapping ε is extended to arbitrary terms in the obvious way by $\varepsilon(0) = 0$ and $\varepsilon(\mathrm{succ}(t)) = \varepsilon(t) + 1$. For set quantification we then define

$$(\omega, <, V) \models_\varepsilon \exists X.\Phi \text{ iff exists } A \text{ such that } (\omega, <, V[X \mapsto A]) \models_\varepsilon \Phi.$$

Notice that $\mathcal{F}_1(<)$ is indeed the first-order fragment of $\mathcal{F}_2(\mathrm{succ})$. Showing this amounts to showing that any first-order formula in $\mathcal{F}_2(\mathrm{succ})$ is equivalent to a formula the terms of which are all variables. But example 8.2 can be used to show how "composite" terms can be eliminated.

Example 14.2 *(Properties expressible in $\mathcal{F}_2(succ)$)*

1. The formula

$$\begin{aligned}\Phi_1(y) \;=\; & \exists X.X(0) \wedge \forall x.(X(x) \supset \neg X(\mathrm{succ}(x))) \wedge \\ & (\forall x.X(x) \supset X(\mathrm{succ}(\mathrm{succ}(x)))) \wedge (X(y) \supset Y(y))\end{aligned}$$

   expresses $even(Y)$ in the individual variable y.

2. Let $\Phi(X)(x)$ be a formula in the set variable X and the individual variable x. The formula

$$\begin{aligned}\Phi_2(y) \;=\; & \exists X.(\forall x.\Phi(X)(x) \supset X(x)) \wedge \\ & (\forall Y.(\forall x.\Phi(Y)(x) \supset Y(x)) \supset (\forall x.X(x) \supset Y(x))) \wedge X(y)\end{aligned}$$

   expresses that the least fixed point of the mapping that takes a set A to the set $\{n \mid (\omega, <, V[X \mapsto A]) \models_{\varepsilon[x \mapsto n]} \Phi(X)(x)\}$ exists and contains y.

We obtain:

Theorem 14.3 *For any $\mathcal{F}(\bigcirc, \mu)$ formula ϕ over the singleton label set there is an equivalent $\mathcal{F}_2(succ)$ formula ϕ^*.*

PROOF: Use example 14.2. □

Exercise 14.4 *(Easy)* Generalise 14.3 to arbitrary label sets.

The other half of expressive completeness is much more difficult, and we therefore shift attention for a while from the linear time μ-calculus to S1S.

15 Büchi Automata

The key to proving the containment S1S $\Rightarrow \mathcal{F}(\bigcirc, \mu)$ is to prove a kind of normal form Theorem for S1S. Automata serves as such normal forms. We introduce Büchi automata and prove that they define the same languages as S1S and the linear time μ-calculus.

Definition 15.1 *(Büchi automata)* A *Büchi automaton* (over a finite alphabet Σ) is a nondeterministic finite automaton:

$$\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a \in \Sigma}, F)$$

where

1. $(S, \{\xrightarrow{a}\}_{a \in \Sigma})$ is a labelled transition system
2. $s_0 \in S$ is the *initial state*
3. $F \subseteq S$ are the *accepting states*

The difference between Büchi automata and usual NFA's is that Büchi automata accept *infinite* strings.

Definition 15.2 *(Run, Successful run, Acceptance)*

1. A *run* of $\mathcal{A}$ on an ω-word $w \in \Sigma^\omega$ is a run r s.t.

   (i) $r(0) = s_0$

   (ii) for all $i \geq 0$, $r(i) \xrightarrow{w(i)} r(i+1)$

   Note that only infinite runs are considered.

2. A run r is *successful* if for infinitely many i, $r(i) \in F$.
3. The Büchi automaton $\mathcal{A}$ *accepts* w if a successful run on w exists.
4. The *language accepted by* $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } w\}$$

5. $\mathcal{L} \subseteq \Sigma^\omega$ is *Büchi recognisable* if $\mathcal{L} = \mathcal{L}(\mathcal{A})$ for $\mathcal{A}$ a Büchi automaton.

Example 15.3 The Büchi automaton on fig. 3 accepts the set of strings over $\Sigma = \{a, b\}$ that contains infinitely many occurrences of a. The automaton on fig. 4 accepts those strings that have an occurrence of a at every even transition.

Exercise 15.4 *(Easy)* Find Büchi automata accepting the set of strings over $\Sigma = \{a, b\}$ for which

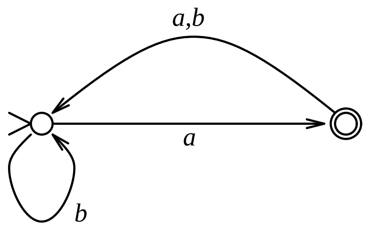

Figure 3: Example Büchi automaton

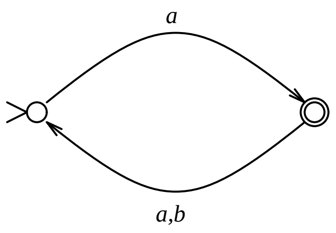

Figure 4: Example Büchi automaton

1. a and b occurs infinitely often.

2. Between any two occurrences of a there is a even number of occurrences of b.

The aim is to show the following containments where $L_1 \Rightarrow L_2$ should be interpreted to mean that any set of models/languages defined/accepted by a formula/automaton in L_1 is similarly definable in L_2:

1. $\mathcal{F}(\bigcirc, \mu) \Rightarrow \mathcal{F}_2(\text{succ})$

2. Büchi recognisable $\Rightarrow \mathcal{F}_2(\text{succ})$

3. Büchi recognisable $\Rightarrow \mathcal{F}(\bigcirc, \mu)$

4. $\mathcal{F}_2(\text{succ}) \supset$ Büchi recognisable

It follows that all of $\mathcal{F}(\bigcirc, \mu)$, $\mathcal{F}_2(\text{succ})$, and the class of Büchi recognisable languages are equivalent. It must be considered, however, exactly what is meant by "equivalent" here. Previously, in comparing $\mathcal{F}(\bigcirc, U)$ and $\mathcal{F}_1(<)$, we restricted attention to models over the singleton label set. Models for $\mathcal{F}_1(<)$ then coincided with runs for $\mathcal{F}(\bigcirc, U)$ up the renaming of states taking $r(i)$ to i. Automata, however, are typically viewed relative to a primitive alphabet Σ. We reconcile these views by, for the time being, restricting attention to those cases where each

$a \in \Sigma$ is a finite set of propositional, or set variables [4]. With respect to a run r through an underlying modal model and a finite set A of propositional variables we can then identify

1. r (for temporal logics) with
2. the model $(\omega, <, V)$ such that $i \in V(X)$ if and only if $r(i) \in V(X)$ and $X \in A$ (for classical logics) with
3. the infinite string $a_0, a_1, \ldots$ for which $a_i = \{X \mid i \in V(X), X \in A\}$ (for automata, and later, grammars).

Statement such as "$\mathcal{F}(\bigcirc, \mu) \Rightarrow$ Büchi recognisable languages" must thus be viewed with this representation in mind. That is, whenever A is the set of runs definable by a formula in $\mathcal{F}(\bigcirc, \mu)$ ($A = \{r \mid r \models \phi, \phi \in \mathcal{F}(\bigcirc, \mu)\}$) then the language of infinite strings derived from A as above is recognised by a Büchi automaton.

We have already established the containment (1) above and continue to show the containment "Büchi recognisable $\Rightarrow \mathcal{F}_2(\text{succ})$".

Theorem 15.5 *If $\mathcal{L} \subseteq \Sigma^\omega$ is Büchi recognisable then $\mathcal{L}$ is expressible by an equivalent S1S-formula Φ.*

Proof: Suppose that $\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$ accepts $\mathcal{L}$. We give a formula that holds for an ω-word w just in case w is accepted by $\mathcal{A}$. The formula expresses the existence of sets $\sigma_0, \ldots, \sigma_m$ s.t. $n \in \sigma_i$ iff at the n'th step, $\mathcal{A}$ is in the i'th state. The following formula $\Phi(\mathcal{A})$ will do:

$$\begin{array}{rcl}
\Phi(\mathcal{A}) & = & \exists X_0 \cdots X_m . \\
& & \bigwedge\{\forall x. X_i(x) \supset \neg X_j(x) \mid 0 \le i \le m, 0 \le j \le m, i \ne j\} \\
& & \text{"}\{X_i \mid 0 \le i \le m\} \text{ a partitioning"} \\
& & \wedge X_0(0) \\
& & \text{"}s_0 \text{ is the initial state"} \\
& & \wedge \forall x. \bigvee\{X_i(x) \wedge \bigwedge\{Y(x) \mid Y \in a\} \wedge X_j(\text{succ}(x)) \mid s_i \xrightarrow{a} s_j\} \\
& & \text{"Transitions adhere to the transition relation"} \\
& & \wedge \bigvee\{\forall x. \exists y. x < y \wedge X_i(y) \mid s_i \in F\} \\
& & \text{"An accepting state is visited i.o."}
\end{array}$$

□

Then for the containment "Büchi recognisable $\Rightarrow \mathcal{F}(\bigcirc, \mu)$".

Theorem 15.6 *If $\mathcal{L} \subseteq \Sigma^\omega$ is Büchi recognisable then $\mathcal{L}$ is expressible by an equivalent $\mathcal{F}(\bigcirc, \mu)$ formula.*

[4]There are other ways of reconciling these views, for instance by keeping the a's primitive and instead adding for each $a \in \Sigma$ a primitive proposition symbol q_a.

PROOF: Let $\mathcal{A}$ be a Büchi automaton with accepting states $F = \{s_1, \dots, s_n\}$ and whenever $1 \leq i \leq n$ let $\mathcal{A}_i$ be $\mathcal{A}$ with F replaced by the singleton set $\{s_i\}$.
Then $\mathcal{L}(\mathcal{A}) = \bigcup_{1 \leq i}^{n} \mathcal{L}(\mathcal{A}_i)$.
Hence it suffices to consider automata with singleton acceptance sets.
Let $\mathcal{A}$ be such an automaton, $\sigma \subseteq S$, and for each $s \in S$ let X_s be a distinguished propositional variable. For each such s and σ define the representation of s up to the set of already visited states σ:

$$C(s)\sigma = \begin{cases} X_s & \text{if } s \in \sigma \\ \mu X_s. \bigvee\{a.C(s')\sigma \cup \{s\} \mid s \xrightarrow{a} s'\} & \text{if } s \notin \sigma \text{ and } s \notin F \\ \nu X_s. \bigvee\{a.C(s')\{s\} \mid s \xrightarrow{a} s'\} & \text{otherwise} \end{cases}$$

Then we show that $r \in \|C(s)\emptyset\|$ iff $\mathcal{A}$ with initial state s accepts the ω-word $\lambda(r,1)\lambda(r,2)\cdots$.

Exercise 15.7 *(Recommended)* Complete the proof.

□

16 ω-regular Languages

To complete this batch of expressive equivalence results the hardest single step remains, namely that of showing the S1S definable languages to be Büchi recognisable. This result is due to Büchi [4]. The proof here follows Thomas [55] fairly closely. The basic steps are classical, and familiar from the theory of regular languages and finite automata: Show the Büchi recognisable languages to be closed under all the S1S connectives, show them to contain all S1S atomic formulas, and a trivial induction then gives the result.

First, however, it is convenient to introduce the notion of ω-regular language, and show the equivalence of ω-regular languages and Büchi automaton recofnisable languages.

Definition 16.1 *(ω-regular Language)* A language $\mathcal{L}$ is *ω-regular* (rational, ω-rational) if $\mathcal{L}$ is a finite union

$$\mathcal{L} = \bigcup_{i=1}^{n} L_i \cdot K_i^{\omega}$$

for L_i, K_i regular languages.

Theorem 16.2 *If $\mathcal{L}$ is recognisable by a Büchi automaton then it is ω-regular.*

PROOF: For $w = a_1 \cdots a_n \in \Sigma^*$ let $s \xrightarrow{w} s'$ if and only if there are $s_0, \ldots, s_n$ s.t. $s_0 = s$, $s_n = s'$ and

$$s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$$

For later use let also $s \xrightarrow{w}_F s'$ if a sequence $s_0, \ldots, s_n$ as above exists with $s_i \in F$ for at least one s_i.
Then

$$\begin{aligned} W_{s,s'} &\triangleq \{w \in \Sigma^* \mid s \xrightarrow{w} s'\} \\ W^F_{s,s'} &\triangleq \{w \in \Sigma^* \mid s \xrightarrow{w}_F s'\} \end{aligned}$$

Claim 16.3 *Each $W_{s,s'}$, $W^F_{s,s'}$ is a regular language.* □

Exercise 16.4 *(Easy)* Show this.

The ω-language accepted by a Büchi automaton $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \bigcup_{s \in F} W_{s_0,s} \cdot (W_{s,s})^\omega.$$

Hence, by 16.3 it is ω-regular. □

The converse containment is shown by extablishing the following closure properties of the regular and Büchi recognisable sets.

Lemma 16.5 *1. If $\mathcal{L} \subseteq \Sigma^*$ is regular then $\mathcal{L}^\omega$ is Büchi recognisable*

2. If $L \subseteq \Sigma^*$ is regular and $K \subseteq \Sigma^\omega$ is Büchi recognisable then $L \cdot K$ is Büchi recognisable

3. If $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^\omega$ are Büchi recognisable then $\mathcal{L}_1 \cup \mathcal{L}_2$ and $\mathcal{L}_1 \cap \mathcal{L}_2$ are Büchi recognisable.

PROOF: (1) Let $\mathcal{A}$ be a finite automaton recognising $\mathcal{L}$. Since $\mathcal{L}^\omega$ does not contain ε we can assume that $\mathcal{L}$ does not contain ε either. From $\mathcal{A}$ can be constructed another automaton $\mathcal{A}'$ that does not have transitions into the initial state s_0. To build the desired Büchi automaton, $\mathcal{A}''$, add for each transition $s' \xrightarrow{a} s''$ with $s'' \in F$ a transition $s' \xrightarrow{a} s_0$ in $\mathcal{A}''$, and let s_0 be the single accepting state of $\mathcal{A}''$. Then $\mathcal{A}''$ accepts $\mathcal{L}^\omega$.

Exercise 16.6 *(Easy)* Prove (2) and (3) for $\cup$.

(3) Finally for $\cap$. Let $\mathcal{A}_i = (S_i, s_{0_i}, \{\xrightarrow{a}_i\}_{a \in \Sigma}, F_i)$ recognise $\mathcal{L}_i$ for $i \in \{1, 2\}$. An automaton recognising $\mathcal{L}_1 \cap \mathcal{L}_2$ can be built by tagging pairs of states (s_1, s_2) with $s_1 \in S_1$ and and $s_2 \in S_2$ with either 0, 1, or 2. 0 indicates that no accepting state has been met, 1 that an accepting state in $\mathcal{A}_1$ has been met and 2 that an

accepting state in both $\mathcal{A}_1$ and $\mathcal{A}_2$ has been met. The initial state will thus be the triple $(s_{0_1}, s_{0_2}, 0)$. The transition relation will be determined by the rules

$$\frac{s_1 \xrightarrow{a}_1 s_1' \qquad s_2 \xrightarrow{a}_2 s_2'}{(s_1, s_2, i) \xrightarrow{a} (s_1', s_2', i)}$$

$$\frac{s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2' \quad s_1' \in F_1}{(s_1, s_2, 0) \xrightarrow{a} (s_1', s_2', 1)}$$

$$\frac{s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2' \quad s_2' \in F_2}{(s_1, s_2, 1) \xrightarrow{a} (s_1', s_2', 2)}$$

$$\frac{s_1 \xrightarrow{a}_1 s_1' \qquad s_2 \xrightarrow{a}_2 s_2'}{(s_1, s_2, 2) \xrightarrow{a} (s_1', s_2', 0)}$$

The accepting states of the combined automaton will be all states of the form $(s_1, s_2, 2)$.

Exercise 16.7 *(Easy)* Verify that the combined automaton does recognise $\mathcal{L}_1 \cap \mathcal{L}_2$.

□

We have thus proved:

Theorem 16.8 *An ω-language $\mathcal{L}$ is Büchi recognisable iff it is ω-regular.*

PROOF: By the closure properties just established. □

17 Closure Under Complement

The standard approach to proving closure underr complement by reducing to (easy to complement) deterministic automata will not work.

Let $\mathcal{A}$ be a *deterministic* finite automaton accepting the regular language $\mathcal{L} \subseteq \Sigma^*$. As a Büchi automaton, $\mathcal{A}$ accepts $w \in \Sigma^\omega$ just in case infinitely many prefixes of w belong to $\mathcal{L}$ (and thus lead to an accepting final state). That is,

$$\begin{aligned} \mathcal{L}(\mathcal{A}) &= \lim \mathcal{L} \\ &\triangleq \{w \in \Sigma^\omega \mid \text{for infinitely many } i.w(0,i) \in \mathcal{L}\} \end{aligned}$$

where $w(0,i)$ is the substring $w(0) \cdots w(i)$ of w.

Not all ω-regular languages have this form. Let

$$\begin{aligned} \mathcal{L} &= \{w \in \{a,b\}^\omega \mid w \text{ contains only a finite} \\ &\qquad\qquad \text{number of occurrences of } a\} \\ &= (a^* b^*) b^\omega \end{aligned}$$

Assume that $\mathcal{L} = \lim K$. We obtain $n_1, n_2, \ldots$ s.t.

$$\begin{array}{ll} b^{n_1} \in K & \text{as } b^\omega \in \mathcal{L} = \lim K \\ b^{n_1}ab^{n_2} \in K & \text{as } b^{n_1}ab^\omega \in \mathcal{L} = \lim K \\ \vdots & \qquad \vdots \end{array}$$

and then $b^{n_1}ab^{n_2}a \cdots \in \lim K = \mathcal{L}$. But this is a contradiction.

Moreover, this example shows that the deterministic Büchi recognisable language are not closed under complementation. For note that

$$\mathcal{L} = \{a, b\}^\omega \perp \lim(b^*a)^*$$

One alternative is to show closure under complement directly. There are other alternatives, for instance to use automata with modified acceptance conditions that does allow determinisation. We return to this issue later.

Theorem 17.1 *If $\mathcal{L} \subseteq \Sigma^\omega$ is Büchi recognisable then so is $\Sigma^\omega \perp \mathcal{L}$.*

PROOF: For the proof both $\mathcal{L}$ and $\Sigma^\omega \perp \mathcal{L}$ will be represented as finite unions $\mathcal{L}_1 \cdot \mathcal{L}_2^\omega$ where $\mathcal{L}_1$ and $\mathcal{L}_2$ are regular languages. The result then follows.
The languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are constructed as congruence classes. A relation $\sim \subseteq \Sigma^* \times \Sigma^*$ is a *congruence*, if $\sim$ is an equivalence relation (ie. reflexive, symmetric and transitive) such that whenever $w_1 \sim w_1'$ and $w_2 \sim w_2'$ then $w_1 \cdot w_2 \sim w_1' \cdot w_2'$.
Let now $\mathcal{L} = \mathcal{L}(\mathcal{A})$. Recall that

$$W_{s,s'}^{(F)} = \{w \in \Sigma^* \mid s \xrightarrow{w}_{(F)} s'\}$$

Let then

$$\begin{array}{lll} w_1 \sim w_2 & \text{iff} & \forall s, s'.(w_1 \in W_{s,s'} \text{ iff } w_2 \in W_{s,s'}) \\ & & \quad \text{and } (w_1 \in W_{s,s'}^F \text{ iff } w_2 \in W_{s,s'}^F) \end{array}$$

or equivalently:

$$\begin{array}{rcl} [w] & = & \bigcap_{s,s' \in S} \{W_{s,s'} \mid w \in W_{s,s'}\} \cap \\ & & \bigcap_{s,s' \in S} \{W_{s,s'}^F \mid w \in W_{s,s'}^F\} \cap \\ & & \bigcap_{s,s' \in S} \{\Sigma^* \perp W_{s,s'} \mid w \notin W_{s,s'}\} \cap \\ & & \bigcap_{s,s' \in S} \{\Sigma^* \perp W_{s,s'}^F \mid w \notin W_{s,s'}^F\} \end{array}$$

As each $W_{s,s'}^{(F)}$ is regular, so is each equivalence class $[w]$.
The induced partition is finite.
Moreover, $\sim$ is a congruence.
Say that $\sim$ *saturates* $\mathcal{L}' \subseteq \Sigma^\omega$, if for all $w_1, w_2 \in \Sigma^*$,

$$[w_1] \cdot [w_2]^\omega \cap \mathcal{L}' \neq \emptyset \text{ implies } [w_1] \cdot [w_2]^\omega \subseteq \mathcal{L}'$$

Lemma 17.2 $\sim$ *saturates* $\mathcal{L}$.

Exercise 17.3 *(Recommended)* Show this.

Notice that it follows that $\sim$ saturates $\Sigma^\omega \perp \mathcal{L}$ as well. For if

$$[w_1] \cdot [w_2]^\omega \cap (\Sigma^\omega \perp \mathcal{L}) \neq \emptyset$$

then

$$[w_1] \cdot [w_2]^\omega \not\subseteq \mathcal{L},$$

so by saturation,

$$[w_1] \cdot [w_2]^\omega \cap \mathcal{L} = \emptyset,$$

whence

$$[w_1] \cdot [w_2]^\omega \subseteq (\Sigma^\omega \perp \mathcal{L})$$

so $\sim$ does indeed saturate $\Sigma^\omega \perp \mathcal{L}$.
Suppose now that whenever $\sim$ saturates $\mathcal{L}'$ then

$$\mathcal{L}' = \bigcup \{[w_1] \cdot [w_2]^\omega \mid [w_1] \cdot [w_2]^\omega \cap \mathcal{L}' \neq \emptyset\} \tag{3}$$

This equation holds, in particular, for $\Sigma^\omega \perp \mathcal{L}$. But then $\Sigma^\omega \perp \mathcal{L}$ is ω-regular, and the proof is completed. It thus remains to prove equation 3.
By saturation,

$$\bigcup \{[w_1] \cdot [w_2]^\omega \mid [w_1] \cdot [w_2]^\omega \cap \mathcal{L}' \neq \emptyset\} \subseteq \mathcal{L}'$$

To prove the converse containment let $w \in \mathcal{L}'$. We define an equivalence relation $\equiv$ on pairs (i, j) with $i < j$ by letting

$$(i, j) \equiv (i', j') \text{ iff } w(i, j) \sim w(i', j')$$

As the partition of Σ^* induced by $\sim$ is finite, then so is the one on such pairs (i, j) induced by $\equiv$.
Now *Ramsey's theorem* states that there is an infinite set $H = \{i_0, i_1, \ldots\}$ which is *homogeneous*, meaning that

> there is a pair (i, i') such that whenever $i_k, i_l \in H$ and $k < l$ then $(i, i') \equiv (i_k, i_l)$.

In particular all pairs (i_k, i_{k+1}) are in $[(i, i')]_\equiv$.
Then

$$w \in [w(0, i_0)]_\sim \cdot ([w(i, i')]_\sim)^\omega.$$

The proof of (3), and consequently the Complementation Theorem, is thus complete. $\square$

18 $\mathcal{F}_2(\mathbf{succ}) \Rightarrow$ Büchi recognisable

We are now in a position to finish the proof of expressive completeness.

Theorem 18.1 *If $\mathcal{L} \subseteq \Sigma^\omega$ is definable in S1S then it is ω-regular.*

Proof: First we reduce S1S to a simpler logic $\mathrm{S1S}_0$ with formulas generated by

$$\Phi ::= X \subseteq Y \mid \mathrm{succ}(X) = Y \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \exists X.\Phi$$

where $\mathrm{succ}(X) = Y$ means "X and Y are singletons $\{x\}$ and $\{y\}$, and $\mathrm{succ}(x) = y$".

Let Φ be any S1S formula. We have already seen how 0 and nested occurrences of succ can be eliminated. To eliminate $<$ rewrite for instance $x < y$ as

$$\forall X.(X(\mathrm{succ}(x)) \wedge \forall z.(X(z) \supset X(\mathrm{succ}(z)))) \supset X(y)$$

Thus Φ can be rewritten into a form where only atomic formulas of the forms $x = y$, $\mathrm{succ}(x) = y$ and $X(x)$ can occur. To eliminate individual variables let now

$$\begin{aligned}
X = Y &\stackrel{\Delta}{=} X \subseteq Y \wedge Y \subseteq X \\
X \neq Y &\stackrel{\Delta}{=} \neg(X = Y) \\
\mathrm{Sing}(X) &\stackrel{\Delta}{=} \exists Y.Y \subseteq X \wedge X \neq Y \wedge \\
&\qquad\qquad \forall Z.Z \subseteq X \supset X = Z \vee Y = Z
\end{aligned}$$

Then we associate to each individual variable a 2nd order variable and rewrite for instance

$$\begin{aligned}
\mathrm{succ}(x) = y &\quad\text{as}\quad \mathrm{Sing}(X) \wedge \mathrm{Sing}(Y) \wedge \mathrm{succ}(X) = Y \\
\exists x.X(x) &\quad\text{as}\quad \exists Y.\mathrm{Sing}(Y) \wedge Y \subseteq X
\end{aligned}$$

Completing this procedure an equivalent $\mathrm{S1S}_0$ formula is obtained.

Exercise 18.2 Check out the details.

The second, and final, step is to show all $\mathrm{S1S}_0$ definable languages to be Büchi recognisable. We first recall what we mean by $\mathrm{S1S}_{(0)}$ definable:
An $\mathrm{S1S}_0$ formula $\Phi(X_1, \ldots, X_n)$ is represented as a Büchi automaton $\mathcal{A}$ over the language $2^{\{X_1,\ldots,X_n\}}$ which is *equivalent* in the sense that

> Φ is true of the model $(\omega, <, V)$ if and only if $\mathcal{A}$ accepts the ω-word $a_0, a_1, \ldots$ where for all $i \in \omega$, $a_i = \{X_j \mid 1 \leq j \leq n \wedge i \in V(X_j)\}$

The automaton $\mathcal{A}$ is constructed by induction on the structure of Φ.

Exercise 18.3 *(Easy)* Show that the languages defined by atomic $S1S_0$ formulas are Büchi recognisable.

We have already shown the Büchi recognisable languages closed under $\neg$ and $\vee$. For $\exists$ we need to show closure under *projection*. Let $\mathcal{A}$ accept the language defined by $\Phi(X_1, \ldots, X_m, X)$. The projection of $\mathcal{A}$ with respect to X is obtained by replacing each transition $s \xrightarrow{a} s'$ by the transition $s \xrightarrow{a'} s'$ with $a' = a \cap \{X_1, \ldots, X_m\}$.

Exercise 18.4 *(Easy)* Show that the projection of $\mathcal{A}$ with respect to X recognises the language defined by $\exists X.\Phi$.

The proof is thus complete. $\square$

To summarise we have now shown that languages defined by the linear time μ-calculus, S1S, Büchi automata, and ω-regular languages are the same. The most difficult part of this is an induction in the structure of S1S formulas showing that all S1S definable languages are recognised by Büchi automata. This is a kind of normal form Theorem. Indeed, Siefkes [50] used a construction like this to obtain a completeness result for S1S. Of the inductive cases, complementation turned out to be by far the most involved. This need not necessarily be the case. By working with formulas in positive form, complementation may be avoided altogether (or at least: restricted to propositional variables)—at the expense of having to deal with other connectives such as conjunction and universal quantification. Indeed, Dam [11] used such an approach applied to the linear time μ-calculus to give a direct proof of the equivalence between Büchi automata and $\mathcal{F}(\bigcirc, \mu)$.

19 Quantified $\mathcal{F}(\bigcirc, G)$ and ETL

Another couple of equivalent theories deserves mentioning at this stage. Both were introduced by Wolper in [60]. First, the language $\mathcal{F}(\bigcirc, G, \forall)$ is linear time temporal logic with $\bigcirc$ and 2nd order propositional quantification. An example formula is

$$\exists X.X \wedge G((Y \vee \bigcirc X) \supset X) \wedge \forall Z.(G((Y \vee \bigcirc Z) \supset Z)) \supset (Z \supset X).$$

It is not hard to see that this formula expresses FY.

Theorem 19.1 *$\mathcal{F}(\bigcirc, G, \forall)$ and $\mathcal{F}(\bigcirc, \mu)$ are expressively equivalent.* $\square$

Exercise 19.2 Show this.

A second equivalent theory is Wolper's *extended temporal logic*, ETL. We started out from Wolper's observation that $even(q)$ is not expressible in $\mathcal{F}(\bigcirc, U)$. This observation led Wolper to an extension of temporal logic which has, in addition to the boolean connectives, an n-ary operator $\Omega(\mathcal{G})(Y_1, \ldots, Y_n)$ for every right-linear grammar $\mathcal{G} = (N, T, P, X_0)$ with terminals $T = \{Y_1, \ldots, Y_n\}$.

Definition 19.3 *(Right-linear grammars)* A context-free grammar $\mathcal{G} = (N, T, P, X_0)$ is *right-linear* if every production in P has one of the forms

$$\begin{aligned} X &\to Y \\ X &\to YX \end{aligned}$$

for $X \in N$ and $Y \in T$.

For the semantics we define

$$r \models \Omega(\mathcal{G})(\phi_1, \ldots, \phi_n)$$

if and only if there is an infinite word w generated by $\mathcal{G}$ such that for all $i \in \omega$, whenever $w(i) = Y_j$ then $r^i \models \phi_j$.

Instead of right-linear grammars we could use ω-regular languages or Büchi automata.

Example 19.4

1. The grammar $\mathcal{G}$ with productions

   $$X_0 \to YX_0$$

   expresses GY.

2. Consider the grammar $\mathcal{G}$ with productions

   $$\begin{aligned} X_0 &\to Y_1X_1 \\ X_1 &\to Y_2X_2 \\ X_2 &\to Y_3X_2 \end{aligned}$$

   Then $\Omega(\mathcal{G})(\top, \phi, \top)$ expresses $\bigcirc\phi$.

Exercise 19.5 *(Easy)* Express U and *even* in ETL.

The following result which is stated without proof is due to Wolper et al [61].

Theorem 19.6 *ETL and S1S are equally expressive.* □

20 Applications 1: The Linear Time Fixed Point Hierarchy

The expressiveness results obtained so far has a number of interesting consequences. We here consider two:

1. the linear time fixed point hierarchy, and
2. decidability

Since alternation of fixed points has so drastic impact on the legibility of formulas, it is of interest to consider formulas up to their alternation depth. Let Σ_n^μ, Π_n^μ have the properties that

1. Σ_n^μ and Π_n^μ are closed under all connectives except μ and ν
2. $\Sigma_n^\mu \subseteq \Pi_{n+1}^\mu$
3. $\Pi_n^\mu \subseteq \Sigma_{n+1}^\mu$
4. $\phi \in \Sigma_n^\mu$ implies $\mu X.\phi \in \Sigma_n^\mu$
5. $\phi \in \Pi_n^\mu$ implies $\nu X.\phi \in \Pi_n^\mu$

The following corollary should be compared with the strictness result for Niwinski's μ-calculus [43].

Corollary 20.1 *Every $\mathcal{F}(\bigcirc, \mu)$-formula is equivalent to a formula in Σ_2^ν.*

Proof: The representation used in the proof of Theorem 15.6 stays within Σ_2^ν.
□

21 Applications 2: Decidability

A major motivation for originally studying Büchi automata in particular was to provide decision procedures for theories such as S1S.

Theorem 21.1 *The emptiness problem for Büchi automata is decidable.* □

Exercise 21.2 *(Easy)* Prove this.

As all the equivalence results (with respect to S1S) proved above involve effective translations this entails:

Corollary 21.3 1. *The equivalence problem for Büchi automata is decidable.*

2. *The satisfiability and validity problems for S1S, $\mathcal{F}(\bigcirc, \mu)$ and $\mathcal{F}(\bigcirc, G, \forall)$ are decidable.*

Proof: (1) $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$ iff

$$(\mathcal{L}(\mathcal{A}_1) \perp \mathcal{L}(\mathcal{A}_2)) \cup (\mathcal{L}(\mathcal{A}_2) \perp \mathcal{L}(\mathcal{A}_1)) = \emptyset.$$

(2) Rewrite to Büchi automata and complement, if necessary. □

Of course decidable does not necessarily mean *manageable*. We return to this issue later. A couple of other corollaries of the expressive completeness Theorem are worth mentioning:

WS1S: The *weak monadic theory of successor* has 2nd order quantification over finite sets only. Finite sets may be expressed in S1S by replacing $\Phi(X)$ by "X is bounded and $\Phi(X)$":

$$(\exists y.\forall x.X(x) \supset x < y) \wedge \Phi(X)$$

Presburger arithmetic: This is the first order language of ω under addition. The idea is to represent numbers as finite bitstrings. Then $n + m = k$ is defined in WS1S by the formula

$$\begin{aligned}\exists c.\neg(c(0)) \wedge \forall x. \quad & (\neg n(x) \wedge \neg m(x) \wedge \neg c(x) \supset \\ & \quad \neg k(x) \wedge \neg c(\mathrm{succ}(x))) \wedge \\ & (n(x) \wedge \neg m(x) \wedge \neg c(x) \supset \\ & \quad k(x) \wedge \neg x(\mathrm{succ}(x))) \wedge \\ & \ldots\end{aligned}$$

Then the decision problem for Presburger arithmetic reduces to that of WS1S. By considering instead infinite bitstrings and reducing to S1S it can be seen that also the 1st order theory of the reals is decidable.

22 Complexity

Even though (for instance) S1S and $\mathcal{F}(\bigcirc, \mu)$ are expressively equivalent their complexities differ quite dramatically. Let the function $t_k(n)$ be defined inductively by

$$t_0(n) = n$$

$$t_{k+1}(n) = 2^{t_k(n)}$$

A language $\mathcal{L} \subseteq \Sigma^*$ is *elementary-recursive* if there is a Turing machine recognising $\mathcal{L}$ in space bounded by $t_k(n)$ for some fixed k. We state the following result without proof.

Theorem 22.1 *(Meyer [41]) Satisfiability of formulas in WS1S is not elementary-recursive.* □

So even though a decision procedure for WS1S exists, it is in practice utterly useless (for general formulas, at least). This result extends to S1S and $\mathcal{F}(\bigcirc, G, \forall)$ and contrasts with the following:

Theorem 22.2 *Satisfiability of formulas in $\mathcal{F}(\bigcirc, \mu)$, emptiness of Büchi automata, emptiness of ω-regular expressions, are all elementary-recursive.* □

In fact much sharper upper bounds are known. A couple of examples:

1. Emptiness of Büchi automata is logspace complete for NLOGSPACE [?].

2. Satisfiability for ETL is PSPACE-complete [60].

3. Satisfiability for $\mathcal{F}(\bigcirc, \mu)$ is decidable in deterministic exponential time (this follows from a corresponding result for the modal μ-calculus [58].

23 Determinisation

We have already seen that deterministic Büchi automata are strictly weaker than their nondeterministic counterpart. Nevertheless determinisation is a very important and useful property. It is used in decision procedures for various program logics such as

1. the modal μ-calculus [53, 58].

2. CTL* [18].

and not least the monadic 2nd order theory of the binary tree—which we will return to later.

So it is of interest to find alternative notions of automata for which determinisation is in fact possible. Three well-known such notions exist, called Muller, Rabin, and Streett automata. In the next few sections we introduce these types of automata and characterise their descriptive power.

24 Muller Automata

Definition 24.1 *(Muller automata)* A *Muller automaton* is a finite automaton

$$\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$$

for which

1. each $\xrightarrow{a}$ is *deterministic* (ie. for every s and a there is a unique s' for which $s \xrightarrow{a} s'$)

2. $F \subseteq 2^S$ is a collection of accepting state-sets.

An ω-word w is *accepted* by the Muller automaton $\mathcal{A}$ as above, if the set of states occurring infinitely often in the unique run r of $\mathcal{A}$ on w is a member of F. Then an ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is *Muller recognisable* if it is the set of all ω-words accepted by some Muller automaton.

Nondeterministic Muller automata have the same power as deterministic ones. We do not consider them further.

Theorem 24.2 *Every ω-language recognised by a deterministic Büchi automaton is Muller recognisable.*

PROOF: Let $\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$ be a deterministic Büchi automaton. Let $F_M = \{A \subseteq S \mid A \cap F \neq \emptyset\}$. Then the Muller automaton $\mathcal{A}_M = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F_M)$ recognises $\mathcal{L}(\mathcal{A})$. $\square$

Theorem 24.3 *The Muller recognisable language are closed under Boolean operations.*

PROOF: If $\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$ recognises $\mathcal{L}$ then $(S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, 2^S \perp F)$ recognises $\Sigma^\omega \perp \mathcal{L}$. If $\mathcal{A}_i = (S_i, s_{0_i}, \{\xrightarrow{a}_i\}_{a\in\Sigma}, F_i)$ recognises $\mathcal{L}_i$, $i \in \{1,2\}$, then the product automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$ accepts $\mathcal{L}_1 \cup \mathcal{L}_2$ where the acceptance set F contains all sets A such that either $\{s_1 \mid (s_1, s_2) \in F\}$ or $\{s_2 \mid (s_1, s_2) \in F\}$. $\square$

Theorem 24.4 *Any Muller recognisable language is a Boolean combination of sets $limW$ for $W \subseteq \Sigma^*$ regular.*

PROOF: (Easy—this one is from [55]). Any set lim$W$ is recognised by a deterministic Büchi automaton, as we have seen, and hence Muller recognisable by Theorem 24.2. So any boolean combination of such languages is also Muller recognisable, by Theorem 24.3.

Conversely, let $\mathcal{A}$ be a Muller automaton recognising $\mathcal{L}$. For each $s \in S_{\mathcal{A}}$ let $W_s \subseteq \Sigma^*$ be the regular language accepted by the DFA $\mathcal{A}$ modified to having the single accepting state $\{s\}$. Write $\mathcal{L}$ as a Boolean combination of the sets limW_s.
$\square$

The theorem, due to McNaughton [40], that Büchi and Muller automata recognises the same languages, is the fundamental theorem of the theory of automata on infinite objects. We have almost shown one direction:

Corollary 24.5 *Any Muller recognisable language is Büchi recognisable.*

PROOF: If $\mathcal{L}$ is Muller recognisable, $\mathcal{L}$ can be written as a Boolean combination of sets lim$W$. Each lim$W$ is Büchi recognisable, and the Büchi recognisable languages are closed under Boolean operations. $\square$

In order to show the converse inclusion we take a detour through some variants of Muller automata.

25 Rabin and Streett Automata

Definition 25.1 *(Rabin automata)* A *Rabin automaton* is a finite automaton

$$\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, \Omega)$$

for which

1. each $\xrightarrow{a}$ is *deterministic*

2. Ω is a finite set $\{(RED_1, GREEN_1), \ldots, (RED_n, GREEN_n)\}$ such that for each $i : 1 \leq i \leq n$, $RED_i, GREEN_i \subseteq S$.

A run r of such a Rabin automaton is *accepting*, if for some i, $1 \leq i \leq n$, all states in RED_i occur only a finite number of times in r and some state in $GREEN_i$ occur infinitely often in r.

Streett automata are similar to Rabin automata, but with the dual acceptance condition (for all i, $1 \leq i \leq n$, if some state in $GREEN_i$ occurs infinitely often in r then so does some state in RED_i).

Again there are nondeterministic versions of Rabin and Streett automata, no more powerful than their deterministic counterparts. No proofs of this will be given.

Theorem 25.2 *Any Rabin recognisable language is Muller recognisable.*

PROOF: Suppose that $\mathcal{L}$ is recognised by the Rabin automaton $(S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, \Omega)$ and that $\Omega = \{(RED_1, GREEN_1), \ldots, (RED_n, GREEN_n)\}$. Let W_i be the regular language recognised by the finite automaton $(S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, GREEN_i)$ and W_i' be the regular language recognised by the finite automaton $(S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, RED_i)$. Then

$$\mathcal{L} = \bigcup_{1\leq i}^{n} (\lim W_i \cap (\Sigma^\omega \perp \lim W_i'))$$

Each set $\lim W_i$ ($\lim W_i'$) is recognised by a deterministic Büchi automaton, and hence by a Muller automaton, by Theorem 24.2. By Theorem 24.3 then so is $\mathcal{L}$. □

26 The Subset Construction

McNaughton's Theorem is consequently proved once we show that any Büchi recognisable language is recognised by a Rabin automaton. The proof given here is due to Safra.

First it is instructive to consider why the usual subset construction fails. Consider the Büchi automaton $\mathcal{A}_1$ of fig. 26. Replacing states by sets of states and the transition relation by its determinised version we obtain $\mathcal{A}_2$ of fig. 26. Clearly we cannot take as accepting those states of $\mathcal{A}_2$ containing an accepting state in $\mathcal{A}_1$, as for the finite case. For then $\{s_0, s_1\}$ becomes erroneously accepting. An alternative strategy tries to mark states, as they are encountered. When accepting states are encountered, they are marked. Also successors of marked states are marked. When all members of a state set are marked that state set is deemed to be accepting, and the marking of its members reset. This is fine for $\mathcal{A}_1$, but consider $\mathcal{A}_3$ of fig. 26. Applying the subset construction to $\mathcal{A}_3$ gives $\mathcal{A}_2$ again,

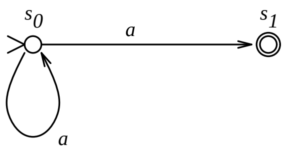

Figure 5: Büchi automaton $\mathcal{A}_1$

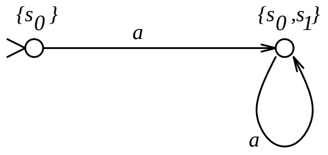

Figure 6: Büchi automaton $\mathcal{A}_2$

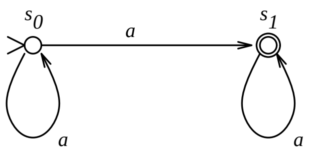

Figure 7: Büchi automaton $\mathcal{A}_3$

but according to the marking strategy no state-sets will be marked.

What we really should do is after looking at $\{s_0, s_1\}$ some finite amount of time, discard s_0 and go on with the remainder—which would in this case be trivially accepting.

27 Safra's Construction

The way Safra's construction does this is by maintaining each state in the determinised version as a tree of subsets, such that when looking at $\{s_0, s_1\}$ a child is created labelled $\{s_1\}$, and then the state labelled by this tree is accepting when either all members of $\{s_0, s_1\}$ has been marked, or the same holds for $\{s_1\}$.

Let $\mathcal{A} = (S, s_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$ be the given Büchi automaton. We construct the Rabin automaton

$$\mathcal{A}_R = (S_R, s_{R_0}, \{\xrightarrow{a}_R\}_{a\in\Sigma}, \Omega)$$

States of $\mathcal{A}_R$ are ordered trees labelled with subsets of S, and satisfying the conditions:

1. The union of labels of the children of a node v is a proper subset of the label of v,
2. Two nodes that are not ancestral have disjoint labels.

Furthermore each node is *coloured* either *white* or *green* to code up acceptance.

Note that the size of each such tree is bounded by $n = |(S)|$. For the set labelling each node v contains a state $s \in S$ which belongs only to ancestors of v.

The *initial state* s_{R_0} is the tree containing a single node labelled by $\{s_0\}$, coloured white.

The deterministic *transition relation* $\xrightarrow{a}_R$ transforms a tree by the following sequence:

1. Set the colour of all nodes to white
2. Whenever σ labels v and $\sigma \cap F \neq \emptyset$ create a new rightmost child v' of v labelled $\sigma \cap F$ and colour it white
3. If v is labelled by σ, replace σ by $\cup_{s\in\sigma}\{s' \mid s \xrightarrow{a} s'\}$ and keep the colouring unchanged
4. If both v and v' contain s and v' is a left sibling of v, remove s from the set labelling v
5. Remove all nodes labelled by $\emptyset$
6. If the union of labellings of the children of v is equal to the labelling of v itself, remove all descendants of v and colour v green

The application of the deterministic transition relation thus involve both creation and deletion of nodes.

Naming, first policy: To define acceptance we first identify nodes by a unique name drawn, say, from ω. Then a state in $\mathcal{A}_R$, consisting of a labelled, coloured, and named tree as above, is accepting if one of its nodes are green. This is the Büchi acceptance condition.

Naming, second policy: However, this will not bound the number of states, and thus a finite representation is needed. Instead we draw names only from the interval 1 to $2n$. All states in $\mathcal{A}_R$ uses at most n names, as we saw. When creating new nodes any number in this interval not already used can be taken. At most n new numbers are needed at every transition. The removal of nodes cuts down the number of nodes in the new state to at most n. For acceptance it must be the case that

1. Node i exists and is green infinitely often
2. Node i exists almost always

This gives a Rabin acceptance set Ω of size linear in n:

$$\Omega = \{(RED_i, GREEN_i) \mid 1 \leq i \leq 2n\}$$

for which RED_i is the set of trees for which node i does not exist, and $GREEN_i$ the set of trees for which node i is green

Exercise 27.1 Verify that the two naming strategies with their acceptance conditions are equivalent.

In view of this it suffices to consider the simpler first policy when proving correctness.

Theorem 27.2 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_R)$

PROOF: We first show that if $\mathcal{A}_R$ accepts $w \in \Sigma^\omega$ then so does $\mathcal{A}$. Assume then that a (uniquely named) node v is green infinitely often in a run of $\mathcal{A}_R$. Let $i_1 < i_2 < \cdots$ be the positions at which v is green, and let $i_0 = 0 < i_1$. Let σ_j be the label of v at position i_j.

Exercise 27.3 *(Recommended)* Show that whenever $s' \in \sigma_{j+1}$ then there is some $s \in \sigma_j$ and a path of $\mathcal{A}$ on $w(i_j, i_{j+1})$ from s to s' which visits an accepting state.

Construct now a tree of pairs (s, j) with $s \in \sigma_j$ with $(s', j+1)$ a child of (s, j) just in case some path as above exists. This is a finitely-branching, infinite tree. Hence, by König's lemma, it has an infinite branch. This branch represents an accepting run of $\mathcal{A}$.

Assume conversely that r is an accepting run of $\mathcal{A}$ on w. At every stage the root of $\mathcal{A}_2$ contains the corresponding state of $\mathcal{A}$. It is consequently never removed.

If the root is green infinitely often we are done. So assume not. Let then i be least such that the root from stage i onwards is never green, and $r(i)$ is accepting.

At this point $r(i)$ is placed in a child of the root. The state of r from i onwards can be moved to a left sibling, but this can only happen a finite number of times. So from $j \geq i$ onwards the state of r is a member of the same child of the root. Unless, of course, a child containing the state of r is after i some time removed. But that's impossible. If this child is infinitely often green then we're done. Otherwise an accepting member of r is eventually placed in the next level, and the argument can be repeated.

As the depth of the tree is finite, some node must eventually become green infinitely often, concluding the correctness proof. $\square$

We have consequently proved:

Corollary 27.4 *(McNaughton [40]) A language is ω-regular iff it is Muller recognisable (iff it is Büchi recognisable iff it is Rabin recognisable).*

The reason why this proof is particularly interesting among the many proofs of this theorem is that the complexity of the construction is essentially optimal: The number of states in the Rabin automaton is bounded both above (by the construction, see [48]) and below by $2^{\mathcal{O}(2\log 2)}$. We do not go into details.

28 An Application: WS1S

McNaughton's theorem can be used to show the expressive power of S1S and WS1S to be equivalent. We have already seen that the finite set quantifiers are definable within S1S, so only the converse inclusion must be shown. The proof of this relies on a finitary version of Büchi's theorem.

Theorem 28.1 *(Büchi 1960,Elgot 1961) A language $\mathcal{L} \subseteq \Sigma^*$ is regular iff it is definable in S1S interpreted over finite words.* $\square$

The proof of this is very similar to the proof of the infinite case. It involves first redefining the interpretation of S1S formulas. Variables are limited to range over existing positions, the successor operation must be modified, and the empty word handled.

We must then show that this finite interpretation of S1S encodes finite runs of NFA's. The result then follows by a bit of rewriting, and then using the well-known closure properties of regular languages. But then every regular language is also definable in WS1S, similarly modified.

Theorem 28.2 *S1S and WS1S define the same ω-languages.*

PROOF: By McNaughton's theorem every ω-regular language is equivalent to a Boolean combination of sets lim$W$. We find a WS1S formula Φ defining W when interpretations are over finite words.

By relativising each quantifier in Φ to x (i.e. replacing $\forall y.\Phi'$ by $\forall y.y < x \supset \Phi'$ etc) we obtain a WS1S formula $\Phi'(x)$ defining the language of ω-words w for which $w(0,x) \in W$. But then lim$W$ is defined in WS1S by $\forall x.\exists y.x < y \wedge \Phi'(y)$. □

29 Star-free Regular Expressions

An expressive completeness proof similar to the one we have been giving for $\mathcal{F}(\bigcirc, \mu)$ with respect to $\mathcal{F}_2(\mathrm{succ})$ is quite feasible also for $\mathcal{F}(\bigcirc, U)$ with respect to $\mathcal{F}_1(<)$, even though we have preferred here the proof of Gabbay et al [20]. Here we merely state the results for reference.

Definition 29.1 *(Star-free regular expressions, languages)* A *star-free regular expression* is an expression built up using $\emptyset$, alphabet symbols $a \in \Sigma$, concatenation, union, and complementation with respect to Σ^*. A *star-free language* is a language generated by a star-free regular expression.

We obtain the following result (c.f. Thomas [55]) extending the expressive completeness result for $\mathcal{F}(\bigcirc, U)$ which we have already proved:

Theorem 29.2 *The following conditions are equivalent for an ω-language $\mathcal{L} \subseteq \Sigma^*$:*

1. *$\mathcal{L}$ is definable in $\mathcal{F}(\bigcirc, U)$.*

2. *$\mathcal{L}$ is definable in $\mathcal{F}_1(<)$.*

3. *$\mathcal{L}$ is a finite union of sets $\mathcal{L}_1 \cdot \mathcal{L}_2^\omega$ where $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$ are star-free.*

4. *$\mathcal{L}$ is a finite union of sets lim$\mathcal{L}_1 \cap (\Sigma^\omega \perp$ lim$\mathcal{L}_2)$ where $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$ are star-free.*

5. *$\mathcal{L}$ is obtained from Σ^ω by nested applications of boolean operations and concatenation with star-free languages to the left.* □

30 Linear Time Logics: Extensions and Variations

We have almost completed our round of linear time temporal logics and given ample evidence that the expressive power of $\mathcal{F}(\bigcirc, \mu)$ is indeed natural and stable, and in some sense also necessary. Whether at the end of the day it is sufficient is much less clear.

Sticking to the linear time structure $(\omega, <)$ there are trivially extensions of S1S that are strictly stronger. The line is traditionally drawn where decidability stops. Some predicates such as "is a power of k", "is a factorial" can be added. The doubling function can not.

We have already considered some linear time structures other than $(\omega, <)$ such as the first order theory of the reals under addition. Within linear time logic past time operators have been considered extensively.

1. It is the setting generally considered in Tense Logic (c.f. [7]).

2. Expressive completeness results for $\mathcal{F}(\bigcirc, U)$ extend to the language with also the past time versions "previous" and "since" with respect to the first-order theory of the integers [32].

3. Vardi [57] considers the linear time μ-calculus with "previous", but interpreted over ω. This language is no more expressive than $\mathcal{F}(\bigcirc, \mu)$ (**Exercise:** Show this) but the addition of past time operators can be argued to be better suited for *compositional* reasoning, in particular as the procedure for eliminating past time connectives may be prohibitively costly.

4. Results similar to the above for the second order monadic theory of the integers $(\mathrm{Z}, <)$ can be obtained using automata-theoretic techniques.

The technique we used for proving expressive completeness for $\mathcal{F}(\bigcirc, U)$ has its roots with Shelah [49] (see also [22]). The methods introduced by Shelah have been used to prove decidability and undecidability for a variety of orderings and, most notably:

Theorem 30.1 *(Shelah 1975) The monadic theory of* $(\mathrm{R}, <)$ *is undecidable.* □

31 Branching Time Logics

A new dimension is added when formulas are allowed to talk not only of properties of positions in a single run, but also to examine the branching structure of models, for instance by quantifying over runs passing through a given state, or, as in modal logic, quantifying over possible next states. This adds very considerable richness, since there is much more variability in the fundamental notions such as what are the "right" underlying notions of model, and what are the meta-languages with which to compare expressiveness. Partly for this reason answers to questions of expressiveness have tended to be much less definitive than for linear time logics, and also comparatively more emphasis has been put on the interrelationship between different branching time logics, than on the relationship with e.g. first or second-order classical theories. This change of emphasis is reflected in the remaining parts of the note.

A general notion of model appropriate to a branching time setting consists of a modal model (TS/LTS/labelled modal model) together with a set of infinite runs through it.

Definition 31.1 *(Branching time structure)* A *branching time structure* is a structure

$$\mathcal{T} = (S, \{\xrightarrow{a}\}_{a \in \Sigma}, V, \mathcal{R})$$

where

1. $\mathcal{T}' = (S, \{\xrightarrow{a}\}_{a \in \Sigma}, V)$ is a labelled modal model, and
2. $\mathcal{R}$ is a set of runs through $\mathcal{T}'$

Notice that the proviso concerning totality of model remains in force so that attention can be restricted to infinite runs. The set $\mathcal{R}$ of runs serves to delineate the sets of runs over which formulas can quantify: These may for instance be all runs satisfying a given *fairness constraint*, given by a suitable criterion such as a linear time temporal formula, or an automaton-like acceptance condition.

A particularly important class of branching time structures are obtained when $\mathcal{R}$ is the set of all runs through its underlying transition system. If that is the case $\mathcal{T}$ is said to be *R-generable*. The following characterisation of the R-generable structures is due to Emerson [14].

Let $\rho = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$ be a finite, and $r = s_n \xrightarrow{a_{n+1}} \cdots$ an infinite run. Then the *concatenation* of ρ and r is the (infinite) run

$$\rho \cdot r = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a_{n+1}} \cdots$$

Definition 31.2 *(Suffix, fusion, limit closure)* The set $\mathcal{R}$ is said to be

1. *Suffix closed*, if $r \in \mathcal{R}$ implies $r^1 \in \mathcal{R}$,
2. *Fusion closed*, if $r_1, r_2 \in \mathcal{R}$ and $r_1(i) = r_2(j)$ implies $r_1(0,i) \cdot r_2^j \in \mathcal{R}$,
3. *Limit closed*, if whenever there is for any i an (infinite) run ρ_i s.t. $r(0,i) \cdot \rho_i$ is defined and a member of $\mathcal{R}$, then $r \in \mathcal{R}$.

Theorem 31.3 *$\mathcal{R}$ is suffix, fusion, and limit closed iff there is an R-generable branching time structure $\mathcal{T}$ with set of runs $\mathcal{R}$.*

Exercise 31.4 Prove this.

32 CTL*

The first branching time logic we consider, called CTL*, is obtained by adding path (run) quantifiers to $\mathcal{F}(\bigcirc, U)$. We deal (until further notice) only with branching time structures that are suffix-closed. This restriction can easily be lifted.

Formulas of CTL* are generated, consequently, by the following grammar:

$$\phi ::= q \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid a.\phi \mid \phi_1 U \phi_2 \mid E\phi$$

The usual definitions apply, and moreover we define $A\phi \triangleq \neg E \neg \phi$.

For the semantics, given a fixed branching time structure, we add to the satisfaction clauses for $\mathcal{F}(\bigcirc, U)$ the condition

$$r \models E\phi \text{ iff } \exists r_1 \in \mathcal{R}.\ r(0) = r_1(0) \text{ and } r_1 \models \phi$$

We see that, like propositional variables, the holding of a quantified formula such as $A\phi$ depends only on the current *state*, and not the on the run in its entirety. Reflecting this many presentations of CTL* maintain a distinction between state- and path-formulas: $\bigcirc$ and U builds path-formulas, propositional variables and the path quantifiers build state-formulas, and both path- and state-formulas are closed under boolean connectives. For state-formulas ϕ we can define a derived satisfaction condition by

$$s \models \phi \text{ if and only if for all } r \text{ with } r(0) = s,\ r \models \phi$$

and then for ϕ a state-formula, $r \models \phi$ if and only if $r(0) \models \phi$.

CTL* was introduced originally by Emerson and Halpern [16] to provide a uniform framework in which the relationship between linear time and branching time logic could be investigated. This issue was brought to the forefront by Lamport in his [34] who showed a simple linear-time and a simple branching time logic to be expressively incomparable, prompting the invention of CTL* to investigate the status of Lamports observations.

Comparing with linear time logic the most important feature of CTL* from a practical point of view is its capacity to express alternating path quantification such as $AGEFq$ meaning that "at all future points it is possible to reach a state where q holds". To get a deeper feel for the expressive power of CTL* it is useful to compare it first with some of its less expressive sublanguages. To compare linear and branching time formulas in terms of their capacity to express properties of *states* we view linear time formulas as universally quantified over paths.

Definition 32.1 *(LTL)* Let LTL be the sublanguage of CTL* consisting of all formulas of the form $A\phi$ where $\phi \in \mathcal{F}(\bigcirc, U)$.

On the other hand we can define a sublanguage of socalled *pure branching time formulas* by disallowing nesting of linear time connectives so that the basic modalities take the form e.g. AF.

Definition 32.2 *(CTL)* Let CTL be the sublanguage of CTL* consisting of all formulas of the form ϕ with the property that any linear time subformula (formula of the form $\phi_1 U \phi_2$ or $a.\phi$) is immediately preceded by a path quantifier.

CTL was introduced by Clarke and Emerson [9]. A number of other sublanguages of CTL* have been introduced over time (c.f. [15]). Here we mention only one other variant which we refer to as CTL($\wedge, \neg$) which relaxes CTL by allowing boolean combinations of linear time formulas before path quantifying, but not nesting. Trivially all three sublanguages, LTL, CTL, and CTL($\wedge, \neg$) are contained in CTL*. We show that in all cases the containment is strict, that CTL and LTL are incomparable in expressive power, and that CTL($\wedge, \neg$) equal to CTL in expressive power.

33 Limits of LTL

We prove Lamports result using a simple proof of Clarke and Draghiescu [8]. We first characterise those CTL* properties that are expressible in LTL.

For simplicity let until further notice $|\Sigma| = 1$.

For a branching time structure $\mathcal{T}$ and $r \in \mathcal{R}_\mathcal{T}$ let $\mathcal{T}(r)$ be the single-path structure

$$\mathcal{T}(r) = (S_r, \rightarrow_r, V_r, \mathcal{R}_r)$$

where

$$S_r = \{r^i \mid i \in \omega\},$$

$$\rightarrow_r = \{(r^i, r^{i+1}) \mid i \in \omega\},$$

$$V_r(q) = \{r^i \mid r^i(0) \in V(q)\},$$

$$\mathcal{R}_r = \{(r^i, r^{i+1}, \ldots) \mid i \in \omega\}$$

Let ϕ be a CTL* formula. Then ϕ^d is the $\mathcal{F}(\bigcirc, U)$-formula obtained by deleting all path quantifiers from ϕ.

Exercise 33.1 *(Easy)* Show that $r \models_{\mathcal{T}(r)} \phi$ iff $r \models_{\mathcal{T}(r)} \phi^d$.

Lemma 33.2 *Let ϕ be a* CTL* *formula. Then ϕ is expressible in* LTL *iff ϕ and $A\phi^d$ are equivalent.*

Proof: Let ϕ and $A\psi$, $\psi \in \mathcal{F}(\bigcirc, U)$ be equivalent. We have

$$\begin{aligned} & r \models_\mathcal{T} \phi \\ & \quad \text{iff} \quad \text{for all paths } r' \text{ s.t. } r(0) = r'(0),\ r' \models_\mathcal{T} \psi \\ & \quad \text{iff} \quad \text{for all paths } r' \text{ s.t.} \ldots, r' \models_{\mathcal{T}(r')} \psi \\ & \quad \text{iff} \quad \text{for all paths } r' \text{ s.t.} \ldots, r' \models_{\mathcal{T}(r')} A\psi \end{aligned}$$

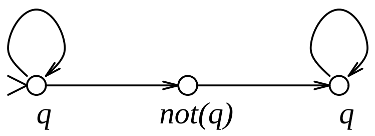

Figure 8: Model $\mathcal{T}$ refuting CLT $\Rightarrow$ LTL

iff for all paths r' s.t. $\ldots, r' \models_{\mathcal{T}(r')} \phi$
iff for all paths r' s.t. $\ldots, r' \models_{\mathcal{T}(r')} \phi^d$
iff for all paths r' s.t. $\ldots, r' \models_{\mathcal{T}} \phi^d$
iff $r \models_{\mathcal{T}} A\phi^d$

□

Theorem 33.3 *The CTL-formula $AFAGq$ is not expressible in LTL.*

PROOF: Consider the R-generable model $\mathcal{T}$ given in fig. 8. Let r be any run through $\mathcal{T}$. Then $r \not\models AFAGq$ but $r \models A(AFAGq)^d$. □

34 Limits of CTL

We give a proof due to Emerson [15]. Inductively construct the sequences of models $M_1, M_2, \ldots$ and $N_1, N_2, \ldots$ as in fig 9. Clearly in all N_i, $s_i \models AFGq$ while in M_i, $s_i \not\models AFGq$. On the other hand we can easily show

Lemma 34.1 *Whenever i is greater than or equal to the length of ϕ then in N_i, $s_i \models \phi$ if and only if $s_{i+1} \models \phi$, and $t_i \models \phi$ if and only if $t_{i+1} \models \phi$.*

PROOF: A simple induction in the structure of ϕ □

With this Lemma in place we can then show

Lemma 34.2 *Whenever i is greater than or equal to the length of ϕ then $s_i \models \phi$ in M_i if and only if $s_i \models \phi$ in N_i, and $t_i \models \phi$ in M_i if and only if $t_i \models \phi$ in N_i.*

PROOF: Another little induction in the structure of ϕ. □

We have thus proved:

Theorem 34.3 *$AFGq$ is not expressible in* CTL. □

We have thus proved that both LTL and CTL are strictly contained in CTL*, and that LTL and CTL are of incomparable expressive power. We conclude the section by showing that no expressive power is gained by allowing boolean combinations of linear time connectives, as long as they are not nested.

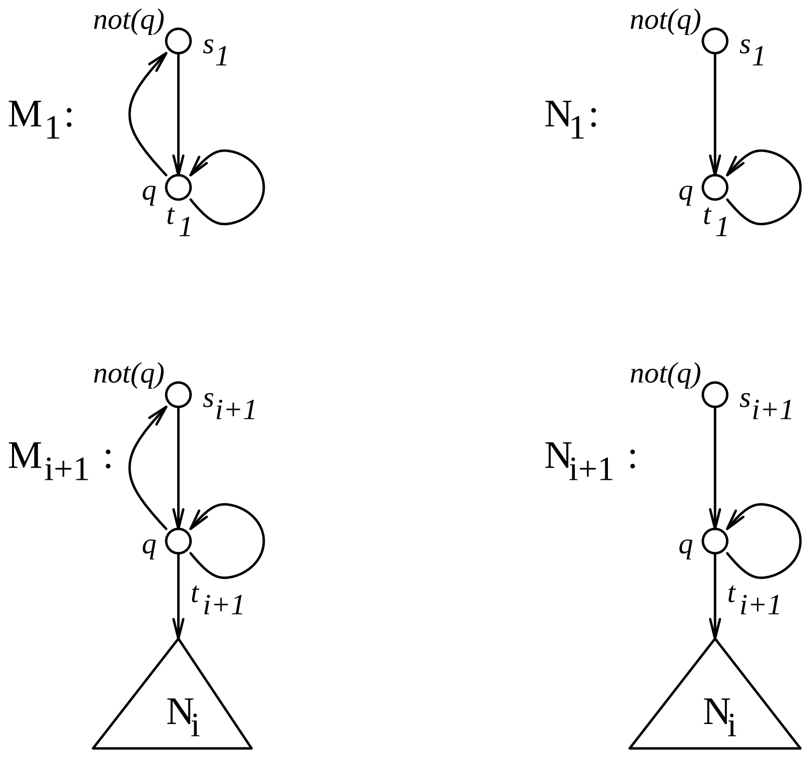

Figure 9: Model $\mathcal{T}$ refuting CLT $\Rightarrow$ LTL

Theorem 34.4 CTL *and* CTL$(\wedge, \vee)$ *are of equal expressive power.*

Proof: We give rules to rewrite each formula in CTL$(\wedge, \vee)$ into a form where all occurrences of the existential path quantifier are applied to subformulas which are conjunctions $\phi_1 \wedge \cdots \wedge \phi_n$ and all ϕ_i have a linear time outermost connective. The rules are the following:

$$E\neg\neg\phi \rightarrow E\phi$$

$$E\neg(\phi \wedge \psi) \rightarrow (E\neg\phi) \vee (E\neg\psi)$$

$$E\neg \bigcirc \phi \rightarrow E \bigcirc \neg\phi$$

$$E\neg(\phi U \psi) \rightarrow (EG\neg\psi) \vee E(\neg\psi U(\neg\phi \wedge \neg\psi))$$

$E\neg\phi \rightarrow \neg\phi$ whenever ϕ is a state-formula

$E(\phi \wedge \psi) \rightarrow \phi \wedge E\psi$ whenever ϕ is a state-formula

$E(\phi \wedge \psi) \rightarrow E\phi \wedge \psi$ whenever ψ is a state-formula

$$E(\neg\neg\phi \wedge \psi) \rightarrow E(\phi \wedge \psi)$$

$$E(\neg(\phi \wedge \psi) \wedge \gamma) \rightarrow E(\neg\phi \wedge \gamma) \vee E(\neg\psi \wedge \gamma)$$

$$E(\neg \bigcirc \phi \wedge \psi) \rightarrow E(\bigcirc\neg\phi \wedge \psi)$$

$$E(\neg(\phi U \psi) \wedge \gamma) \rightarrow (E((G\neg\psi) \wedge \gamma) \vee E((\neg\psi U(\neg\phi \wedge \neg\psi)) \wedge \gamma)$$

The last four rules have symmetric versions too. Then the rewrite procedure is completed by applying the following rules to first push out all occurrences of $\bigcirc$ and then iterating the rewrite procedure until, finally, conjunctions of until-formulas can be removed:

$$E((\phi U \psi) \wedge \gamma) \rightarrow E((\psi \vee (\phi \wedge \bigcirc(\phi U \psi))) \wedge \gamma)$$

$$E(\bigcirc\phi \wedge \bigcirc\psi) \rightarrow E \bigcirc E(\phi \wedge \psi)$$

$$E((\phi_1 U \psi_1) \wedge (\phi_2 U \psi_2)) \rightarrow$$
$$E((\phi_1 \wedge \phi_2) U(\psi_1 \wedge E(\phi_2 U \psi_2))) \vee E((\phi_1 \wedge \phi_2) U(\psi_2 \wedge E(\phi_1 U \psi_1)))$$

The last two rules are given in binary form only. However, suitable generalisations are easily obtained. □

35 Extensions of CTL*

Just as $\mathcal{F}(\bigcirc, U)$ could be argued to be too weak in the linear time case, so CTL* can be argued to be too weak in the branching time case. For instance, CTL* is incapable of expressing a property such as $Aeven(q)$. Any of the solutions adopted in the linear time case can conceivably also be adopted in the case of CTL*. A first solution to spring to mind may be a fixed point extension of CTL*. Exactly how this is to be done may be less obvious. Adding extremal fixed points without any restrictions (other than the familiar syntactic monotonicity condition) results in a logic which is at this date hardly understood at all, and we leave this as a difficult but also exciting issue for future work. A less ambitious route is to restrict fixed points such as to allow the formation of fixed points $\sigma X.\phi$ only when no occurrence of X is within the scope of a path quantifier in ϕ. Using automata this logic can be given an expressively equivalent but more compact formulation. Such an extended version of CTL*, called ECTL*, can be presented as follows:

Formulas of ECTL* are generated by

$$\phi ::= q \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid E(\mathcal{A}, \phi_1, \ldots, \phi_m)$$

where $\mathcal{A}$ is a Büchi automaton over alphabet $\{0,1\}^m$.

For the *semantics*, intuitively $r \models E(\mathcal{A}, \phi_1, \ldots, \phi_m)$ if and only if there is a run r' which corresponds through the assignment $\phi_i \mapsto i$ to an ω-word accepted by $\mathcal{A}$. This correspondence is made precise in the following way: w is determined by for all $i \in \omega$ letting $w(i) = (i_1, \ldots, i_m)$ iff $r'^i \models \phi_{i_j}$ for each j, $1 \leq j \leq m$.

In somewhat different forms ECTL* was introduced by Vardi and Wolper [59] and Clarke et al [10]. The version given here follows Thomas [54].

Theorem 35.1 *ECTL* is strictly more expressive than CTL*.* $\square$

Exercise 35.2 *(Recommended)* Show this.

36 The Modal μ-calculus

An on the face of it quite different way of entering branching time is by adding extremal fixed points to modal logic just as we did for linear time logic. The rationale remains the same.

Example 36.1 The CTL formula AGq can be characterised as the greatest fixed point of the equation

$$X = q \wedge \Box X$$

and the CTL formula $E(q_1 U q_2)$ can be characterised as the least solution of the equation

$$X = q_2 \vee (q_1 \wedge \Diamond X).$$

Syntactically, formulas of the modal μ-calculus, L_μ, are given by the grammar

$$\phi ::= X \,\big|\, \neg\phi \,\big|\, \phi \wedge \phi \,\big|\, [a]\phi \,\big|\, \mu X.\phi$$

The semantics is determined as in def. 12.4 with the only differences that formulas denote sets of states rather than runs, and that the "box" is interpreted as in modal logic, by

$$\|[a]\phi\|V = \{s \mid \forall s'.\text{if } s \xrightarrow{a} s' \text{ then } s' \in \|\phi\|V\}.$$

Most properties of $\mathcal{F}(\bigcirc, U)$ described in sections 12 and 13 transfer to L_μ with minimal changes. In particular, L_μ has a sound and complete tableau system with sequents of the form $s \vdash_{\Delta,V} \phi$ which is obtained from the tableau system for $\mathcal{F}(\bigcirc, \mu)$ by just replacing the rules for the next-time operator with the rules:

$$\frac{s \vdash_{\Delta,V} \,<a>\phi}{s' \vdash_{\Delta,V} \,\bot} \quad s \xrightarrow{a} s'$$

$$\frac{s \vdash_{\Delta,V} [a]\phi}{\{s' \vdash_{\Delta,V} \phi \mid s \xrightarrow{a} s'\}}$$

Notice that in general the rule for $[a]$ is infinitary.

With example 36.1 in mind it is clear that L_μ qualifies as a branching time temporal logic. What may be less clear is that it is in fact highly expressive, strictly including both CTL* and ECTL*. In [12] direct translations of both CTL* and ECTL* are given. However, just to establish the expressiveness result it is, in view of Theorem 35.1, sufficient to consider ECTL* the translation of which is actually the simpler of the two.

Since variables can be translated as variables, negations as negations, and conjunctions as conjunctions, it suffices to consider ECTL* formulas of the form $E(\mathcal{A}, \phi_1, \ldots, \phi_m)$. Moreover, each ϕ_i can be assumed to be a propositional variable X_i. Additionally, if the acceptance set of $\mathcal{A}$ is $\{s_0, \ldots, s_k\}$ and each $\mathcal{A}_i$, $0 \leq i \leq k$, is obtained from $\mathcal{A}$ by replacing F by $\{s_i\}$ then the translation of $E(\mathcal{A}, X_1, \ldots, X_m)$ can start by rewriting into the disjunction

$$E(\mathcal{A}_0, X_1, \ldots, X_m) \vee \cdots \vee E(\mathcal{A}_k, X_1, \ldots, X_m).$$

It thus suffices to consider formulas of the form $E(\mathcal{A}, X_1, \ldots, X_m)$ where $\mathcal{A}$ has exactly one accepting state. For such formulas we build a tree in a rather straightforward fashion, reflecting the state transition structure of $\mathcal{A}$, and the construction translating such formulas into L_μ is then essentially the same as in the proof of Theorem 15.5. We thus obtain:

Theorem 36.2 *For R-generable models, L_μ is at least as expressive as ECTL*.*$\Box$

Exercise 36.3 Complete the proof of this statement.

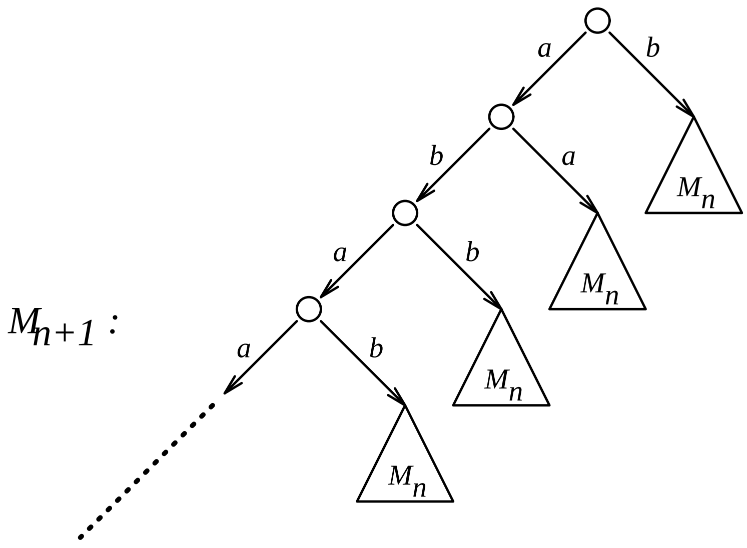

Figure 10: Constructing M_{n+1} from M_n

The construction shows, moreover, that even the fragment of L_μ restricted to alternation depth 2 is at least as expressive as ECTL*. To show that the containment is strict in fact the fragment with alternation depth 1 will do.

Theorem 36.4 *The class of models expressed by the L_μ formula $\nu X.{<}a{>}X \wedge {<}b{>}X$ is not expressible in ECTL*.*

PROOF: We construct an inductive sequence of tree-like models, M_i, $i \in \omega$. M_0 is the model with an infinite path for each $w \in \{a,b\}^\omega$ and branching at the root node only. That is, for each $w \in \{a,b\}^\omega$ and each $i \in \omega$, M_0 has a state (w,i), and in addition M_i has a distinguished root state, s_0. The transition relation is determined by

$s_0 \overset{w(0)}{\rightarrow} (w,1)$ for each $w \in \{a,b\}^\omega$, and

$(w,i) \overset{w(i)}{\rightarrow} (w,i+1)$ whenever $w \in \{a,b\}^\omega$ and $i > 0$.

Then M_{n+1} is built from M_n by adding a new initial state $s_{0_{n+1}}$ and for each $w \in \{a,b\}^\omega$ a transition $s_{0_{n+1}} \overset{w(0)}{\rightarrow} s_{w^1}$ where s_{w^1} is the root state of the tree shown on fig. 10 where w^1 is the example infinite word $abaa\cdots$ labelling the "spine" of the tree. Now an easy inductive proof shows that an ECTL* formula with at most n nestings of the existential path quantifier holds of M_n if and only if it holds of the infinite $\{a,b\}$-labelled binary tree. However, the latter tree satisfies $\nu X.{<}a{>}X \wedge {<}b{>}X$ while this is not the case for any M_n. □

37 S2S - the Monadic Second Order Theory of the Binary Tree

We now turn to expressiveness results with respect to classical theories and automata in the style of S1S and Büchi automata. In contrast to the situation for linear time logic, for branching time logic largely only incomplete results are currently known.

One difficulty is identifying a suitable candidate meta-language with which to compare. The classical choice is Rabin's extension of Büchi's sequential calculus to the infinite binary tree, S2S. Rabin, in a landmark paper [47], proved this theory decidable using automata on infinite binary trees. Many branching time temporal logics (such as CTL* or L_μ) can be reduced to S2S while preserving satisfiability. Thus Rabin's decidability result can be used to show decidability of these logics. However, as S1S, S2S is non-elementary. A computationally more efficient approach is to reduce to the corresponding tree-automata, c.f. [17, 58].

We conclude the tutorial with a look at S2S and tree automata and survey some of the expressiveness results currently known.

The language of S2S is the monadic second-order language of two successor functions. For the syntax, *terms*, ranged over by t, are generated by

$$t ::= x \mid \varepsilon \mid t0 \mid t1$$

where x ranges over individual variables. Here ε is intended to be the empty word, $t0$ the first, or left, successor of t, and $t1$ the second, or right. *Formulas* Φ are generated by

$$\Phi \quad ::= \quad t \in X \mid t_1 = t_2 \mid t_1 < t_2 \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \exists x.\Phi \mid \exists X.\Phi$$

where the usual abbreviations apply.

For the semantics, a *model* for S2S is a labelled binary tree:

$$(\{0,1\}^*, \varepsilon, \mathrm{succ}_0, \mathrm{succ}_1, <, q_1, \ldots, q_n)$$

where

1. ε is the empty word,
2. $\mathrm{succ}_i(w) = wi$, $i \in \{0,1\}$ and $w \in \{0,1\}^*$,
3. $<$ is the prefix relation on $\{0,1\}^*$,
4. $q_1, \ldots, q_n \subseteq \{0,1\}^*$

Exercise 37.1 *(Recommended)* Express in S2S:

1. ε is the empty word,

2. $\text{succ}_i(w) = wi$, $i \in \{0,1\}$ and $w \in \{0,1\}^*$,
3. $<$ is the prefix relation on $\{0,1\}^*$,
4. $q_1, \ldots, q_n \subseteq \{0,1\}^*$

Theorem 37.2 *(Rabin 1969)* S2S *is decidable.*

We later outline a proof of this powerful and difficult result. Decidability of a number of other theories is obtained by reducing to S2S. To name but a few:

1. The second-order monadic theory of n successors, SnS, for any $n \in \omega$ is decidable, as well as SωS,
2. The second-order monadic theory of countable linear orders is decidable,
3. The theory of countable boolean algebras with quantification over ideals is decidable.

38 Some Expressiveness Results

We here mention some of the expressiveness results relating branching time temporal logics to S2S.

- Hafer and Thomas [24] shows that when restricted to the infinite binary tree, CTL* is exactly as expressive as S2S with quantification restricted to paths, and that ECTL* is exactly as expressive as S2S with quantification restricted to chains. For $n > 2$, however, SnS with quantification restricted to chains is strictly more expressive than ECTL*.
- Niwinski [44] shows that a μ-calculus based on powerset algebras of trees is exactly as expressive as a type of infinite tree grammar which in turn is exactly as expressive as SnS.
- Hüttel [26] shows that a μ-calculus generalising $\mathcal{F}(\bigcirc, \mu)$ by having n successor operations is exactly as expressive as SnS up to a kind of observational equivalence of trees.
- Gabbay [19] shows the negative result that general partially ordered time structures gives rise to an infinity of independent temporal connectives.
- Amir [1] obtains expressive completeness for some particular partially ordered time structures in the sense that a finite basis exists for the temporal connectives definable in the first-order language of the structure.

39 Tree Automata

We identify an *infinite Σ-labelled binary tree* with a mapping $\tau : \{0,1\}^* \to \Sigma$. If $\Sigma = 2^{\{q_1,\ldots,q_n\}}$ we may identify such trees with model structures for S2S. Automata on trees generalise automata on words in a fairly straightforward manner.

Definition 39.1 *(Tree automata)* A *tree automaton* is a structure

$$\mathcal{A} = (S, S_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F)$$

where

1. S is the finite set of states,
2. $S_0 \subseteq S$ is the set of initial states,
3. for each $a \in \Sigma$, $\xrightarrow{a} \subseteq S \times S \times S$. Rather than $\xrightarrow{a} (s_1, s_2, s_3)$ we write $s_1 \xrightarrow{a} (s_2, s_3)$.
4. F is an acceptance set to be clarified later.

A *run* of $\mathcal{A}$ on a tree τ is an S-labelled tree $r : \{0,1\}^* \to S$ such that

1. $r(\varepsilon) \in S_0$, and
2. for each $w \in \{0,1\}^*$, $r(w) \xrightarrow{\tau(w)} (r(w0), r(w1))$

Automata on infinite trees come in variants similar to those for automata on infinite word:

Büchi automata: Here $F \subseteq S$ and a successful run is a run r for which for each path through r some $s \in F$ occurs infinitely often.

Rabin automata: Here F is a finite set

$$F = \{(RED_1, GREEN_1), \ldots, (RED_n, GREEN_n)\}$$

of pairs $(RED_i, GREEN_i)$ with $RED_i, GREEN_i \subseteq S$. A run r is successful if for all paths through r there is some i, $1 \leq i \leq n$, such that

1. no $s \in RED_i$ occurs infinitely often, and
2. some $s' \in GREEN_i$ occurs infinitely often

Similarly one can define *Muller* and *Streett* tree automata.

Exercise 39.2 1. Show that

$$\begin{aligned}\mathcal{L}_{T,0} \quad = \quad & \{\tau \mid a \text{ occurs infinitely often} \\ & \qquad \text{along some path of } \tau\}\end{aligned}$$

is Büchi (and hence Rabin) recognisable.

2. Show that
$$\begin{aligned}\mathcal{L}_{T,1} &= \overline{\mathcal{L}_{T,0}}\\ &= \{\tau \mid a \text{ occurs only finitely often}\\ &\qquad\qquad \text{along all paths of } \tau\}\end{aligned}$$
is Rabin recognisable.

3. (Harder) Show that $\mathcal{L}_{T,1}$ is not Büchi recognisable.

4. Show that the Rabin recognisable languages are closed under union and intersection.

5. Show that the Rabin recognisable languages are closed under (Σ_1-) *projection:* Let $\mathcal{A}$ be a Rabin automaton over alphabet $\Sigma_1 \times \Sigma_2$. Construct a Rabin automaton $\mathcal{A}_1$ over alphabet Σ_1 such that $\mathcal{A}_1$ accepts the Σ_1-labelled tree τ_1 iff a Σ_2-labelled tree τ_2 exists s.t. $\mathcal{A}$ accepts the $\Sigma_1 \times \Sigma_2$-labelled tree τ given by
$$\tau(w) = (\tau_1(w), \tau_2(w))$$
for all $w \in \{0,1\}^*$.

40 Tree Automata as Games

We outline (part of) a proof of the expressive equivalence of S2S and the Rabin recognisable languages based on games. This approach is due to Büchi [5, 6] and Gurevich and Harrington [23]. We follow here Gurevich [22].

Definition 40.1 *(Game automata)* A *game automaton* (a tree automaton in [22], but we have already used that term) is a structure
$$\mathcal{A}_g = (S, \{\overset{(i,a)}{\rightarrow}\}_{i\in\{0,1\},a\in\Sigma}, \{\overset{a}{\rightarrow}_{\text{in}}\}_{a\in\Sigma}, F)$$
where

1. S is a finite set of states,
2. for each $i \in \{0,1\}$, $a \in \Sigma$, $\overset{(i,a)}{\rightarrow} \subseteq S \times S$ is the transition relation,
3. for each $a \in \Sigma$, $\overset{a}{\rightarrow}_{\text{in}} \subseteq S$ is the *initial state table,*
4. $F \subseteq 2^S$ is the acceptance set.

Given a tree τ the game $?(\mathcal{A}, \tau)$ is played between $\mathcal{A}$ and another player called *pathfinder.* The game proceeds as follows:

Initially $\mathcal{A}$ chooses a state s_0 such that $\overset{\tau(\varepsilon)}{\rightarrow}_{\text{in}}\ s$.

The game progresses by stages. At each stage pathfinder first chooses an $i \in \{0,1\}$ and then $\mathcal{A}$ answers by choosing a state. This choice is subject to the following restriction. Let $n \geq 1$, and assume $?(\mathcal{A}, \tau)$ has reached stage n. Assume:

1. At stage $n \perp 1$, $\mathcal{A}$ chose s_{n-1}.
2. $d_1, \ldots, d_{n-1}$ is the sequence of $d_j \in \{0,1\}$ chosen by pathfinder at stage j.
3. At stage n pathfinder chooses d_n.

Then the choice s_n of $\mathcal{A}$ must satisfy:

$$s_{n-1} \overset{d_n,\tau(d_1,\ldots,d_n)}{\longrightarrow} s_n$$

To ensure that $\mathcal{A}$ is always able to make a move we introduce a special state FAILURE to which a transition is always possible but from which only FAILURE itself is reachable.

A *position* in $?(\mathcal{A},\tau)$ is a finite prefix p of some play $s_0, d_1, s_1, \ldots$ of $?(\mathcal{A},\tau)$. A *strategy* for $\mathcal{A}$ in the game $?(\mathcal{A},\tau)$ is a map f assigning to every position of even length a legal state. Similarly a strategy for pathfinder assigns to every position of odd length a member of $\{0,1\}$.

The automaton $\mathcal{A}$ *wins* a play $s_0, d_1, s_1, \ldots$ of $?(\mathcal{A},\tau)$ if

$$\{s \mid \exists^\infty j. s = s_j\} \in F,$$

and otherwise pathfinder wins. This is the Muller acceptance condition. Then $\mathcal{A}$ *accepts* τ just in case $\mathcal{A}$ has a winning strategy in $?(\mathcal{A},\tau)$.

Exercise 40.2 1. Show that the languages $\mathcal{L}_{T,0}$ and $\mathcal{L}_{T,1}$ above are game automaton recognisable.

2. Show that a language is Muller tree automaton recognisable (and hence Rabin recognisable) iff it is game automaton recognisable.

3. Show that a language is recognised by a deterministic Muller tree automaton if and only if for some game automaton $\mathcal{A}$, $\mathcal{A}$ has a winning strategy which depends only on the last two elements of every position.

We have already seen (exercise 39.2.4 and 5) that the Rabin recognisable languages are closed under $\cup$, $\cap$, and existential quantification. In view of exercise 40.2.2 the two most important steps left to prove Rabins Theorem are to prove that emptiness of game automaton recognisable languages is decidable, and that the game automaton recognisable languages are closed under complementation. The fundamental tool we use to prove both results is Gurevich and Harrington's socalled Forgetful Determinacy Theorem.

41 Forgetful Determinacy

The complementation of game automaton recognisable languages involves the transformation of the negation of an existential sentence "no strategy for $\mathcal{A}$ in ? $(\mathcal{A}, \tau)$ is winning"

to an existential sentence

"there is a winning strategy for the complemented automaton $\overline{\mathcal{A}}$ in ? $(\overline{\mathcal{A}}, \tau)$"

Determinacy as in exercise 40.2 allows this transformation to be done, but (as we should suspect by now) it is too crude. Rather we show that in order to establish whether or not a player has a winning strategy only a bounded amount of information of the history of the game is needed. The information needed is coded as "last appearance records".

Given a position p, the *last appearance record*, LAR(p) is p with all but the final occurrences of states deleted. So for instance

$$\text{LAR}(s_0, 0, s_1, 0, s_2, 1, s_1) = (s_0, s_2, s_1)$$

Also let node(p) be p with all states deleted, so that e.g.

$$\text{node}(s_0, 0, s_1, 0, s_2, 1, s_1) = (0, 0, 1)$$

Now a *forgetful strategy* for a player is a strategy f with the property that $f(p) = f(q)$ whenever

1. p and q are positions for which it is that players turn to make a move,
2. node(p_1) = node(p_2), and
3. LAR(p_1) = LAR(p_2).

We omit the proof of the following

Theorem 41.1 *(Forgetful determinacy [23]) In any game ? $(\mathcal{A}, \tau)$ either $\mathcal{A}$ or pathfinder has a winning forgetful strategy.* □

42 Complementation

We first use forgetful determinacy to prove the closure under complementation.

Theorem 42.1 *The game automaton recognisable languages are closed under complementation.*

Proof: Assume a given game automaton $\mathcal{A}$. Forgetful strategies can be encoded as trees: Let RECORDS be the set of all finite words u over the alphabet S with the property that each state appears at most once in u. Let Σ' be the set of all mappings

$$g : \text{RECORDS} \to \{0, 1\}.$$

Clearly Σ' is finite. A Σ'-tree τ' *yields* the following strategy for pathfinder:

$$\text{if } g = \tau'(\text{node}(p)) \text{ then make the move } g(\text{LAR}(p)).$$

Lemma 42.2 *$\mathcal{A}$ rejects τ iff there is a Σ'-tree τ' that yields a winning strategy for pathfinder in ?$(\mathcal{A}, \tau)$.* □

Exercise 42.3 Prove this.

We then rewrite the property

"τ' yields a winning strategy for pathfinder"

in a suitable fashion.

Lemma 42.4 *The property*

"τ' yields a winning strategy for pathfinder"

holds iff every path $\varepsilon, d_1, d_1 d_2, \ldots$ through the infinite binary tree satisfies the property (*):
For each n let $w_n = d_1, \ldots, d_n$, $a_n = \tau(w_n)$, and $g_n = \tau'(w_n)$. Then:

(*) *For every run $r = s_0, s_1, \ldots$ of states and every sequence $u_0, u_1, \ldots$ of records, if*

1. $u_0 = s_0$,
2. $\xrightarrow{a_0}_{\text{in}} s_0$,
3. $d_{n+1} = g_n(u_n)$,
4. $s_n \overset{d_{n+1}, a_{n+1}}{\longrightarrow} s_{n+1}$, *and*
5. $u_{n+1} = \mathrm{LAR}(u_n s_{n+1})$,

then the set of all s occurring infinitely often in r is not a member of F. □

Exercise 42.5 Prove this.

The property (*) is expressible as an S1S formula Φ over the alphabet $\{\varepsilon, 0, 1\} \times \Sigma \times \Sigma'$, in the sense that Φ holds for the infinite word

$$(\varepsilon, a_0, g_0), (d_1, a_1, g_1), \ldots$$

if and only if (*) does. With the aid of Büchi's and McNaughton's theorems we can construct a (deterministic) Muller automaton

$$\mathcal{A}' = (S', s_0, \{\overset{(d,a,g)'}{\longrightarrow}\}_{d \in \{\varepsilon,0,1\}, a \in \Sigma, g \in \Sigma'}, F')$$

that recognises the language expressed by Φ.

Let $\mathcal{A}''$ be the game $\Sigma \times \Sigma'$-automaton

$$\mathcal{A}'' = (S', \{\overset{(a,g)''}{\longrightarrow}\}_{a \in \Sigma, g \in \Sigma'}, \{\overset{(a,g)''}{\longrightarrow}_{\text{in}}\}_{a \in \Sigma, g \in \Sigma'}, F')$$

where

- $\overset{(a,g)''}{\rightarrow}_{\text{in}}(s)$ iff $s_0 \overset{(\varepsilon,a,g)}{\rightarrow} s$, and
- $s \overset{(d,(a,g))''}{\rightarrow} s'$ iff $s \overset{(d,a,g)'}{\rightarrow} s'$

Lemma 42.6 *τ' yields a winning strategy for pathfinder if and only if $\mathcal{A}''$ accepts the $\Sigma \times \Sigma'$-tree given by τ and τ'.* □

Exercise 42.7 Prove this.

Let then $\mathcal{A}'''$ be the Σ-projection of $\mathcal{A}''$. Then

$$\begin{aligned} &\mathcal{A}''' \text{ accepts } \tau \\ &\quad \text{iff} \quad \text{some tree } \tau' \text{ yields a winning strategy} \\ &\quad \phantom{\text{iff}} \quad \text{for pathfinder in } ?(\mathcal{A}, \tau) \\ &\quad \text{iff} \quad \mathcal{A} \text{ rejects } \tau \end{aligned}$$

and the proof is complete. □

43 Decidability

We finally prove the decidability of the emptiness problem for game automata and, as a consequence, of S2S.

Theorem 43.1 *It is decidable whether or not the language of trees accepted by a game automaton $\mathcal{A}$ is empty.*

PROOF: First the general problem is reduced to that of a singleton alphabet Σ using projection. Thus the question is reduced to whether or not $\mathcal{A}$ accepts the unique Σ-tree τ. List all forgetful strategies for $\mathcal{A}$ and all forgetful strategies for pathfinder. There is a finite number of each, and one of them will be winning. Each play of a forgetful strategy against another will eventually become periodic, and it can thus be decided who will win. □

The proof of the following Theorem is very much in the spirit of the proof of the similar statement for Büchi automata and S1S.

Theorem 43.2 *Each game automaton recognisable tree language over $\Sigma = 2^{\{X_1,\ldots,X_n\}}$ is expressible in S2S.* □

Again in quite a similar manner to the corresponding proof for S1S we can then show:

Theorem 43.3 *Every S2S expressible tree language is game automaton recognisable.*

PROOF: We have proved all necessary cases except the ground ones, which are left as exercises. □

Thus:

Corollary 43.4 *(Rabin [47])* S2S *is decidable.* □

References

[1] A. Amir. Separation in nonlinear time models. *Information and Control*, 66:177–203, 1985.

[2] K. R. Apt. Ten years of hoare's logic: A survey–part 1. *ACM Transactions on Programming Languages and Systems*, 3:724–767, 1981.

[3] W. J. Blok. The lattice of modal logics: An algebraic investigation. *The Journal of Symbolic Logic*, 45(2):221–236, 1980.

[4] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. International Congress on Logic, Methodology and Philosophy of Science*, 1962.

[5] J. R. Büchi. Using determinacy to eliminate quantifiers. *Lecture Notes in Computer Science*, 56, 1977.

[6] J. R. Büchi. State-strategies for games in $\mathrm{f}_{\sigma\delta} \cap \mathrm{g}_{\sigma\delta}$. *Journal of Symbolic Logic*, 48, 1983.

[7] J. P. Burgess. Basic tense logic. In *Handbook of Philosophical Logic, Vol. II* (D. Gabbay and F. Guenthner (eds.)), pages 89–134, 1984.

[8] E. M. Clarke and Dragiescu. Expressibility results for linear-time and branching-time logics. *Lecture Notes in Computer Science*, 398, 1989.

[9] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. *Lecture Notes in Computer Science*, 131, 1981.

[10] E. M. Clarke, O. Grümberg, and R. P. Kurshan. A synthesis of two approaches for verifying finite state concurrent systems. Manuscript, Carnegie-Mellon University, 1987.

[11] M. Dam. Fixed points of Büchi automata. In *Proc. 12th Conf. Foundations of Software Technology and Theoretical Computer Science,* Lecture Notes in Computer Science, 652:39–50, 1992.

[12] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[13] S. Eilenberg. *Automata, Languages, and Machines Vol. A and B.* Academic Press, 1974 and 1976.

[14] E. A. Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26, 1983.

[15] E. A. Emerson. Modal and temporal logic. In *Handbook of Theoretical Computer Science* (J. van Leeuwen (ed.)), pages 996–1072, 1990. Elsevier.

[16] E. A. Emerson and J. Halpern. “sometimes” and “not never” revisited: On branching versus linear time. *Journal of the ACM*, **33**:151–178, 1986.

[17] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th Symp. Foundations of Computer Science*, pages 328–337, 1988.

[18] E. A. Emerson and A.P. Sistla. Deciding full branching time logic. *Information and Control*, **61**:175–201, 1984.

[19] D. Gabbay. Expressive functional completeness in tense logic. In *Aspects of Philosophical Logic* (U. Monnich, ed.), pages 91–117, 1981. Reidel.

[20] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symp. Principles of Programming Languages*, pages 163–173, 1980.

[21] R. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes Number 7. Center for the Study of Language and Information, 1987.

[22] Y. Gurevich. Monadic second-order theories. In *Model-Theoretic Logics,* J. Barwise and S. Feferman (eds), Springer-Verlag, pages 479–506, 1985.

[23] Y. Gurevich and L. A. Harrington. Automata, trees, and games. In *Proc. STOC*, 1982.

[24] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th ICALP,* Lecture Notes in Computer Science, pages 269–279, 1987.

[25] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, **32**:137–162, 1985.

[26] H. Hüttel. Sns can be modally characterised. *Theoretical Computer Science*, 74:239–248, 1990.

[27] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[28] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[29] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[30] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen, 1968.

[31] N. Immerman and D. Kozen. Definability with a bounded number of bound variables. In *Proc. LICS'87*, 1987.

[32] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD Thesis, University of California, 1968.

[33] D. Kozen and J. Tiuryn. Logics of programs. In *Handbook of Theoretical Computer Science* (J. vsan Leeuwen (ed.), pages 791–840, 1990. Elsevier.

[34] L. Lamport. "sometimes" is sometimes "not never". In *Proc. POPL*, 1980.

[35] L. Lamport. The temporal logic of actions. DEC-SRC technical report no. 79, 1991.

[36] K. G. Larsen. Proof systems for Hennessy-Milner logic with recursion. in *Proc. 13th CAAP* Lecture Notes in Computer Science, 299, 1988.

[37] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall International, 1981.

[38] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. *Lecture Notes in Computer Science*, **193**:97–107, 1985.

[39] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1990 (?).

[40] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[41] A. R. Meyer. Weak monadic second order theory of successor is not elementary recursive. *Lecture Notes in Mathematics*, 453:132–154, 1975.

[42] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[43] D. Niwiński. On fixed point clones. In *Proc. 13th International Coll. Automata, Languages and Programming*, pages 464–473, 1986.

[44] D. Niwiński. Fixed points vs. infinite generation. In *Proc. 3rd Ann. Symp. Logic in Computer Science*, pages 402–409, 1988.

[45] G. D. Plotkin. A structural approach to operational semantics. Aarhus University report DAIMI FN-19, 1981.

[46] V. Pratt. Semantical considerations on floyd-hoare logic. In *Proc. 17th IEEE Symposium on Foundations of Computer Science*, pages 109–121, 1976.

[47] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, **141**:1–35, 1969.

[48] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. Foundations of Computer Science*, pages 319–327, 1988.

[49] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102, 1975.

[50] D. Siefkes. *Büchi's Monadic Second Order Successor Arithmetic.* Lecture Notes in Mathematics, 120. Springer-Verlag, 1970.

[51] C. Stirling. Modal and temporal logics. In *Handbook of Logic in Computer Science* Vol. 2, Oxford University Press, 1992.

[52] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89:161–177, 1991.

[53] R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, **81**:249–264, 1989.

[54] W. Thomas. Computation tree logic and regular ω-languages. *Lecture Notes in Computer Science*, 354:690–713, 1988.

[55] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North-Holland, 1989.

[56] J. F. A. K. van Benthem. *The Logic of Time.* Reidel, 1983.

[57] M. Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th Ann. ACM Symp. Principles of Programming Languages*, pages 250–259, 1988.

[58] M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th ACM Symp. Theory of Computing*, 1985.

[59] M. Y. Vardi and P. Wolper. Yet another process logic. *Lecture Notes in Computer Science*, 164, 1984.

[60] P. Wolper. Temporal logic can be more expressive. *Information and Control*, **56**:72–99, 1983.

[61] P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In *Proc. 24'th FOCS*, 1983.

[62] D. Wood. *Theory of Computation.* John Wiley and Sons, 1987.