The atnext/atprevious Hierarchy on the Starfree Languages

Bernd Borchert
Pascal Tesson

WSI-2004-11

Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Arbeitsbereich Theoretische Informatik/Formale Sprachen
Sand 13
D-72076 Tübingen

borchert@informatik.uni-tuebingen.de

ISSN 0946-3852

The atnext/atprevious Hierarchy on the Starfree Languages

Bernd Borchert Pascal Tesson

Universität Tübingen, Germany

{borchert,tesson}@informatik.uni-tuebingen.de

Abstract

The temporal logic operators **atnext** and **atprevious** are alternatives for the operators **until** and **since**. P **atnext** Q has the meaning: at the next position in the future where Q holds it holds P. We define an asymmetric but natural notion of depth for the expressions of this linear temporal logic. The sequence of classes at_n of languages expressible via such depth-n expressions gives a parametrization of the starfree regular languages which we call the atnext/atprevious ierarchy, or simply the at hierarchy. It turns out that the at hierarchy equals the hierarchy given by the n-fold weakly iterated block product of DA. It is shown that the at hierarchy is situated properly between the until/since and the dot-depth hierarchy.

1 Introduction

This paper continues the study of the expressiveness of fragments of linear temporal logic LTL. Kamp [Ka68] showed already 1968 that LTL and first-order expressibility conincide, especially for languages of finite words. Together with the (later) results of McNaughton & Papert [MP71] this shows that LTL expresses exactly the starfree regular languages. The natural hierarchy on the starfree regular sets is the dot-depth hiearchy. It was shown 1982 by Thomas that dot-depth corresponds to quantifier alternation depth of first-order expressions [Th82]. For the LTL logic, on the other hand, no hierarchy is known which corresponds to the dot-depth hierarchy. The two best-known LTL hierarchies, the until hierarchy [EW96, TW01] and until/since hierarchy [EW96, TW04], do exhaust the starfree languages but do this much more slowly than the dot-depth hierarchy: It was shown by Etessami & Wilke [EW96] that already the second-lowest level of the dot-depth hierarchy contains languages of every level of these two LTL hierarchies. In this paper we present an LTL hierarchy based on the two temporal operators **atnext**, introduced 1984 by Kröger [Kr84], and **atprevious** instead of the operators **until** and **since**. We define a certain notion of depth on these expression and call the resulting hierarchy the at hierarchy. We show that the at hierarchy takes a small step in the direction of the dot-depth hierarchy in the sense that the at hierarchy is situated properly between the until/since and the dot-depth hierarchy.

The standard set of LTL operators consists, in addition to the letter symbols and the Boolean operators, of the future operators **until** (2-ary), $\diamondsuit$ (eventually in the future, 1-ary), and $\oplus$ (next, 1-ary), and of the corresponding past operators **since**, $\diamondsuit$ (eventually in the past), and $\ominus$ (previous).

The 2-ary temporal logic operator **atnext**, introduced by Kröger [Kr84], is an alternative for the operator **until**. "P **atnext** Q" has the meaning as its name suggests: at the next position in the future where Q holds it holds P. As a simple example, the expression "1 **atnext** $(1 \vee 2)$" on alphabet $\{0,1,2\}$ looks whether the first non-0 is a 1, i.e. it describes the language $0^*1\{0,1,2\}^*$. The temporal logic operator **until** can be expressed via an **atnext** expression and vice versa, see Table 1. **atprevious** is defined to be the past operator corresponding to **atnext**. The LTL expressions using only the two operators **atnext** and **atprevious** will be called at expressions. We define a notion of depth for at expressions which is asymmetric in the sense in an expression only the depth of nested right subexpressions is counted, nested left subexpressions are for free. This definition will be justified by the complexity of model-checking of atprevious/atnext expressions which mainly depends on the rightmost depth. The sequence of classes $L(\mathrm{at}_n)$ of languages expressible via such depth-n expressions gives a parametrization of the starfree regular languages which we call the at hierarchy.

Our main result will be that the at hierarchy equals the hierarchy given by the n-fold weakly iterated block product of DA. In this paper DA (in normal face) is a synonym for Δ_2^L of the dot-depth hierarchy. We prove as our main lemma that the pointed languages given by at expressions of depth 1 correspond to the pointed languages of DA. Together with the block product/substitution principle from [TW04] this lemma gives the characterization of the n-th level of the at hierarchy.

We compare the at hierarchy with the until/since hierarchy and the dot-depth hierarchy. It is easy to see that the Δ-levels of the dot-depth hierarchy cap the levels of the at hierarchy which in turn cap the levels the until/since hierarchy, and for some area the three hierarchies grow at the same rate. For the other two directions of these two hierarchy comparisons we show that, besides on their bottom level which is DA for all three, the three hierarchies are incomparable: the second-lowest level of the at hierarchy contains languages from every level of the until/since hierarchy, and the second-lowest level of the dot-depth hierarchy contains languages from every level of the at hierarchy, see Figure 3. In other words: the at hierarchy is situated between the until/since hierarchy and the dot-depth hierarchy, and for all levels besides the bottom level these two inclusions are proper.

The paper is organized as follows. In Section 2 we introduce the syntax and semantics for a propositional linear temporal logic using the **atnext** and **atprevious** operators. We define the at hierarchy and show some easy facts about the levels of the hierarchy. In Section 2 the main lemma is shown, saying that the pointed languages of expressions of level 1 of the at hierarchy are the pointed languages of DA, and we conclude that the n-fold weakly iterated block product of DA equals the n-th level of the at hierachy. We use this characterization together with results from [EW96, Bo04] to completely settle the levelwise relation of the at hierarchy in comparison with the dot-depth and the until/since hierarchy. In Section 4 we add the $\oplus$ (next) and the $\ominus$ (previous) operators to the at logic, and we will see that this just adds some "generalized definite" complexity to it, like this is known for the until/since hierarchy and the dot-depth hierarchy.

2 A temporal logic with the atnext and atprevious operators

We consider the following propositional linear temporal logic on words. It uses besides the symbol b for every letter b of the alphabet Σ and besides the Boolean connectives **true**, **false**, $\neg$, $\wedge$, and $\vee$, the two 2-ary operators **atnext** and **atprevious**. The first was introduced by Kröger [Kr84] as an alternative for the operator **until**, the latter was not defined in that paper but is just the past pendant of the future operator **atnext**, like **since** is the past pendant of **until**. Intuitively,

"φ **atnext** ψ" means "for the smallest position in the future such that ψ holds it also holds φ", while "φ **atprevious** ψ" looks analogously into the past: "for the largest position in the past such that ψ holds it also holds φ'. The operators will be defined formally as follows. An at expression is any correctly built expression using the following connectives already mentioned above: b is an at expression for every letter b of the alphabet Σ, **true** and **false** are at expressions, and if φ and ψ are at expressions then also $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi$ **atnext** $\psi)$, and $(\varphi$ **atprevious** $\psi)$ are. Parentheses may be ommitted for readability. In Section 4 we will consider the case that the expressions can moreover contain the 1-ary operators $\oplus$ (next) and $\ominus$ (previous), these expressions will be called AT expressions. Note that at expressions do not contain explicit variables, like this is the case for every propositional linear temporal logic.

An at expression φ will define a language L_φ over the alphabet Σ. For this we first define by induction the following relation $(x, i) \models \varphi$ where (x, i) is a pointed word, i.e. a pair such that x is a word from Σ^* and i is a position in x, i.e. $i \in \{1, \ldots, |x|\}$.

$(x, i) \models b$
: iff the letter of x at position i is b.

$(x, i) \models \varphi \wedge \psi$
: iff $(x, i) \models \varphi$ and $(x, i) \models \psi$. Likewise for the other Boolean connectives.

$(x, i) \models \varphi$ **atnext** ψ
: iff $(x, j) \models \psi$ does not hold for any position j with $j > i$, or it holds for some position $j > i$ and $(x, j) \models \varphi$ for the smallest such j.

$(x, i) \models \varphi$ **atprevious** ψ
: iff $(x, j) \models \psi$ does not hold for any position j with $j < i$, or it holds for some position $j < i$ and $(x, j) \models \varphi$ for the largest such j.

For a word x define $x \models \varphi$ as follows:

$x \models b$
: does not hold for any letter b.

$x \models \varphi \wedge \psi$
: iff $x \models \varphi$ and $x \models \psi$. Likewise for the other Boolean connectives.

$x \models \varphi$ **atnext** ψ
: iff $(x, i) \models \psi$ does not hold for any position i, or it holds for some position i and $(x, i) \models \varphi$ for the smallest such i.

$x \models \varphi$ **atprevious** ψ
: iff $(x, i) \models \psi$ does not hold for any position i, or it holds for some position i and $(x, i) \models \varphi$ for the largest such i.

Finally, define $L_\varphi := \{x \mid x \models \varphi\}$. For a set of at expressions E let $L(E)$ be the class of languages of finite Boolean combinations of languages L_φ for $\varphi \in E$.

Variants **atnext*** and **atprevious*** of the **atnext** and the **atprevious** operators, resp., are defined as follows:

(a)	φ **until** ψ	$\equiv$	$(\psi$ **atnext** $(\neg\varphi \vee \psi)) \wedge \neg($**false atnext** $\psi)$
(b)	φ **until** ψ	$\equiv$	ψ **atnext*** $(\neg\varphi \vee \psi)$
(c)	φ **atnext** ψ	$\equiv$	$(\neg\psi$ **until** $(\varphi \wedge \psi)) \vee \neg($**true until** $\psi)$
(d)	φ **atnext** ψ	$\equiv$	$(\varphi$ **atnext*** $\psi) \vee \neg($**true atnext*** $\psi)$
(e)	φ **atnext*** ψ	$\equiv$	$\neg\psi$ **until** $(\varphi \wedge \psi)$
(f)	φ **atnext*** ψ	$\equiv$	$(\varphi$ **atnext** $\psi) \wedge \neg($**false atnext** $\psi)$

Table 1: Translating the temporal operators into each other

$(x, i) \models \varphi$ **atnext*** ψ
iff $(x, j) \models \psi$ holds for some position $j > i$ and $(x, j) \models \varphi$ for the smallest such j.

$(x, i) \models \varphi$ **atprevious*** ψ
iff $(x, j) \models \psi$ holds for some position $j < i$ and $(x, j) \models \varphi$ for the largest such j.

$x \models \varphi$ **atnext*** ψ
iff $(x, i) \models \psi$ holds for some position i and $(x, i) \models \varphi$ for the smallest such i.

$x \models \varphi$ **atprevious*** ψ
iff $(x, i) \models \psi$ holds for some position i and $(x, i) \models \varphi$ for the largest such i.

The difference is that in case ψ does not hold in the future the whole expression fails. This "stared" definition, which we prefer to use in the following, and its analogue for **atprevious** are closer to the **until** and **since** definition, resp., see Table 1. Note that in the Abstract and in the Introduction we actually used the "stared" definition, in order to state things more easily. Expressions using the stared versions will also be called at expressions. The depth of an at expression, which we define below, is independent of the choice (with or without star) of this variant.

Examples of at expressions:

φ_1 = "**true atnext*** 1"
on alphabet $\{0, 1\}$ looks whether there exists a 1, i.e. $L_{\varphi_1} = 0^*1\{0, 1\}^*$, a language in Σ_1^L and in DA (see below for the definition of the levels of the dot-depth hierarchy and the definition of DA and its block products)

φ_2 = "1 **atnext*** $(1 \vee 2)$"
on alphabet $\{0, 1, 2\}$ looks whether the first 1 or 2 is a 1, i.e. $L_{\varphi_2} = 0^*1\{0, 1, 2\}^*$, a language in Δ_2^L = DA.

φ_3 = "((**true atnext*** 2) **atprevious*** 1) **atnext*** 0"
on alphabet $\{0, 1, 2\}$ describes the turtle language " starting at the left border look for the first 0, from there look left for the first 1, and from there look right for the first 2, if any of the searches fails, reject", i.e. L_{φ_3} is also a DA language. See the next section for the definition of turtle languages.

φ_4 = "**true atnext*** $(2 \wedge (2$ **atnext*** $(1 \vee 2)))$"
on alphabet $\{0, 1, 2\}$ looks whether there exists a position i with letter 2 such that right of i

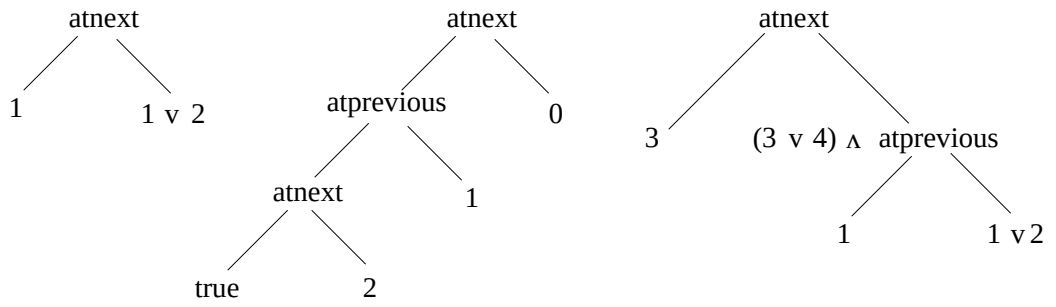

Figure 1: Depth 1 (left and middle) vs. depth 2 (right) expressions

the smallest position having letter 1 or 2 has letter 2, i.e. $L_{\varphi_4} = \Sigma^* 20^* 2\Sigma^*$, a language in Σ_2^L and in DA $\square$ DA.

φ_5 = "3 **atprevious*** $((3 \vee 4) \wedge (1$ **atnext*** $(1 \vee 2)))$"
on alphabet $\{0, 1, 2, 3, 4\}$ looks whether the first position i with a 3 or 4 such that the largest position $j < i$ with a letter 1 or 2 has letter 1, has letter 3. L_{φ_5} is a language in Δ_3^L and DA $\square$ DA.

The line of the examples φ_2 and φ_5 can be continued: Let $\{0, 1, \ldots, 2n{-}1, 2n\}$ be the alphabet for the following at expression δ_n. Define δ_2 to be φ_2 from above, and let δ_n be the expression $2n{-}1$ **atnext****atprevious** $((2n{-}1 \vee 2n) \wedge \delta_{n-1})$, where **atnext** and **atprevious** alternate with every odd/even n. The language L_{δ_n} is an element of $\Delta_n^L - \Delta_{n-1}^L$ in the dot-depth hierarchy by the results of [BLSTT04] combined with oracle results separating the polynomial and the Boolean hierarchy.

Define the *at-depth* $d_{\mathrm{at}}(\varphi)$ of an at expression φ the following way.

$d_{\mathrm{at}}(b) := 0$ for every letter b.

$d_{\mathrm{at}}(\varphi \wedge \psi) := \max(d_{\mathrm{at}}(\varphi), d_{\mathrm{at}}(\psi))$, likewise for the other Boolean connectives.

$d_{\mathrm{at}}(\varphi$ **atnext** $\psi) := d_{\mathrm{at}}(\varphi$ **atprevious** $\psi) := \max(d_{\mathrm{at}}(\varphi), 1 + d_{\mathrm{at}}(\psi))$.

Note the asymmetry in the definition of at-depth. Let at_n be the set of at expressions of at-depth at most n, and let $L(\mathrm{at}_n)$ be the set of languages expressible with them. Examples from above: φ_1, φ_2 and φ_3 have at-depth 1 while φ_4 and φ_5 have at-depth 2, see Figure 1 where φ_2, φ_3 and φ_5 are sketched. Note that by the translations (d) and (f) in Table 1 every at* expression of depth n is equivalent to an at expression of depth n, and vice versa, so at-depth is independent of the "star".

The asymmetry in the definition of at-depth it is justified the following way via the space complexity of the model checking problem for at expressions. Given an input $\langle \varphi, w \rangle$ where φ is an at expression for alphabet Σ and x is a word from Σ^* it is possible to check within alternating space

$$|\phi| + d_{\mathrm{at}}(\phi) \cdot \log(|w|)$$

whether $w \in L_\phi$, and alternation depth is bounded by $|\phi|$. The algorithm just follows the definition of $w \models \varphi$ resp. $(w,i) \models \varphi$: The recursive procedure eval$(\varphi, w, i, new, top)$ will compute whether $(w,i) \models \varphi$, the pseudo code is sketched in Table 2. The Boolean parameter *new* set to 'true' will cause the creation of a new variable in case the procedure steps into an atnext/atprevious operator, while for *new* = 'false' it will not need new space ("call-by-reference"). The Boolean parameter *top* just indicates that there is no atnext/atprevious operator above in the expression tree of φ. The main call is eval$(\varphi, x, 1, \text{true}, \text{true})$ and gives the right answer to the question whether $x \models \varphi$. The observation that the at-depth corresponds to the number of nested calls with the *new* parameter set to 'true' gives the space estimate stated above. The alternation, as a variant of parallelism, is launched by the 2-ary Boolean operators. Note that in case φ is small and/or shallow, or even fixed, the at-depth gives a reasonable upper bound for a very space-efficient model-checking algorithm, its run-time is $O(|w|^n)$ where n is the at-depth of φ. In other words: if space is limited to logarithmic space, i.e. only a fixed numer of variables (= heads) pointing into the input, then the at-depth gives a very sensitive estimation of the running time. If one allows linear space, i.e. arrays of length $|w|$, then of course the algorithm computing for every position i the value $(w,i) \models \psi$ subsequently for all sub-expressions ψ of φ, starting with the bottom ones, will do the model-checking in linear-time.

Usually, LTL logic uses the **until**, the **since**, and the two operators $\diamondsuit$ and $\diamondsuit$ (eventually in the future, eventually in the past) which are defined as follows. Like the atnext/atprevious operators they are defined the way their names indicate it. First we state the definition for pointed words, and afterwards for words. All quantifications are again over the set of positions in x.

$(x,i) \models \varphi$ **until** ψ
: if there exists a position $j > i$ such that $(x,j) \models \psi$ and for all positions k with $i < k < j$ it holds $(x,k) \models \varphi$.

$(x,i) \models \varphi$ **since** ψ
: if there exists a position $j < i$ such that $(x,j) \models \psi$ and for all positions k with $j < k < i$ it holds $(x,k) \models \varphi$.

$(x,i) \models \diamondsuit\varphi$
: if there exists a position $j > i$ such that $(x,j) \models \varphi$.

$(x,i) \models \diamondsuit\varphi$
: if there exists a position $j < i$ such that $(x,j) \models \varphi$.

$x \models \varphi$ **until** ψ
: if there exists a position j such that $(x,j) \models \psi$ and for all positions k with $k < j$ it holds $(x,k) \models \varphi$.

$x \models \varphi$ **since** ψ
: if there exists a position j such that $(x,j) \models \psi$ and for all position k with $j < k$ it holds $(x,k) \models \varphi$.

$x \models \diamondsuit\varphi \iff x \models \diamondsuit\varphi$
: if there exists a position i such that $(x,i) \models \varphi$.

The eventually operators are definable by the **until** operator through the equivalences $\diamondsuit\varphi \equiv$ **true until** φ, and $\diamondsuit\varphi \equiv$ **true since** φ. Nevertheless they are important for the definition of

```
Input: at expression φ, word w, position i in w; Boolean values new and top
Output: true if (w, i) ⊨ φ, false otherwise;

Boolean eval(φ, w, i, new, top) {
  if φ = b for a letter b return true if w_i = b, return false otherwise;
  if φ = true return true;
  if φ = false return false;
  if φ = φ1 ∧ φ2 return AND(eval(φ1, w, i, new, top), eval(φ1, w, i, new, top));
  if φ = φ1 ∨ φ2 return OR(eval(φ1, w, i, new, top), eval(φ1, w, i, new, top));
  if φ = ¬φ1 return NEG(eval(φ1, w, i, new, top));
  if new = true {
      if φ = φ1 atnext φ2 {
          integer j;
          if top = true set j := 1 otherwise set j := i + 1;
          while j ≤ |w| do {
                  if eval(φ2, w, j, true, false) return eval(φ1, w, j, false, false);
                  else set j := j + 1 };
          return 0 };
      if φ = φ1 atprevious φ2 {
          integer j;
          if top = true set j := |w| otherwise set j := i − 1;
          while j ≥ 1 do {
                  if eval(φ2, w, j, true, false) return eval(φ1, w, j, false, false);
                  else set j := j − 1; }
          return 0; } }
    otherwise {
      if φ = φ1 atnext φ2 {
          if top = true set i := 1 otherwise set i := i + 1;
          while i ≤ |w| do {
                  if eval(φ2, w, i, true, false) return eval(φ1, w, i, false, false);
                  else set j := j + 1 };
          return 0 };
      if φ = φ1 atprevious φ2 {
          if top = true set i := |w| otherwise set i := i − 1;
          while i ≥ 1 do {
                  if eval(φ2, w, i, true, false) return eval(φ1, w, i, false, false);
                  else set j := j − 1; }
          return 0; } }
```

Table 2: The model checking procedure eval

until/since depth because they are "for free": the until/since depth $d_{\mathrm{us}}(\varphi)$ of until/since expression φ is defined as the maximal number of **since** and **until** operators on a path of the expression tree of φ. $\oplus$ and $\ominus$ operators on a path are not counted. Let us_k denote the set of until/since expression of depth at most k. In Section 4 the $\oplus$ (next) and the $\ominus$ operators will be added to the logic, again like the eventually operators not counting for depth, and US_k will denote the expressions of that extended syntax which have until/since depth k [TW04]. Again, $L_\varphi := \{x \mid x \models \varphi\}$ for an until/since expression φ. For a set E of until/since expressions let $L(E)$ be the class of languages of finite Boolean combinations of languages L_φ for $\varphi \in E$.

The **until**, **atnext**, and **atnext*** operators can be translated into each other as shown in Table 1, see also [Kr84]. The analogous equivalences hold of course for the past operators, for example "φ **atprevious*** $\psi \iff \neg\psi$ **since** $(\varphi \wedge \psi)$".

It does not seem to be possible to redefine depth of until/since expressions in a way such that it corresponds to at-depth of the translated expression: this may be due to the fact that in the translations the right subexpression in the original expression (in Table 1 this is ψ) appears on both sides of the translated expression. If we would define depth for at expressions the usual way, the hierarchy we get is the same as the until/since hierarchy, this can be verified by the translations in Table 1, and was mentioned already in [TW04][p. 115].

3 The at hierarchy and iterated block products of DA

In this section we first state some simple facts about the at hierarchy. Then we characterize the pointed languages resulting from at expression of depth 1, and use this to characterize the levels of the at hierarchy via the block product/substitution lemma of [TW04]. Finally we compare the at hierarchy with the dot-depth and the until/since hierarchy.

We define the classes of the Straubing-Thérien, which a version of the dot-depth hierarchy (in Section 4 we will consider the other version of it, the Cohen-Brzozowski hierarchy). For $n \geq 1$ let Σ_n^L and Π_n^L be the class of languages definable with a first-order Σ_n or Π_n quantifier prefix over signature $[<]$, see for example the textbook [St94] for the definition of first-order logic on words. Δ_n^L is defined as the intersection of Σ_n^L and Π_n^L. The following observations state that the at hierarchy is infinite and exhausts the starfree regular languages. Part (a) gives a first level-wise comparision with the dot-depth hierarchy. Later we will obtain for $k \geq 2$ the properness of this inclusion, see Theorem 5.

Proposition 1 *Let $k \geq 1$.*
(a) $L(\mathrm{at}_k) \subseteq \Delta_{k+1}^L$,
(b) $\bigcup_{k\geq 1} L(\mathrm{at}_k) = \mathrm{STARFREE}$,
(c) There exists a language D_{k+1} which is in $\Delta_{k+1}^L \cap L(\mathrm{at}_k) \cap L(\mathrm{us}_{k-1})$ but not in Δ_k^L,
(d) $L(\mathrm{at}_k) \subset L(\mathrm{at}_{k+1})$.

Proof. (a) Consider first the case $k = 1$. Given an at_1 expression φ with n atnext/atprevious searches nested to the left, guess the set of positions $x_1, \ldots, x_n$ where the single atnext/atprevious searches are successful, check whether at position x_i the condition searched for is met and check universally whether x_{i+1} is in fact the earliest/latest position seen from x_i for which this is the case. This gives a Σ_2 expression for the languages expressed by φ. Also $\neg\varphi$ can be expressed in Σ_2^L: build

for every prefix of the atnext/atprevious searches of φ the Σ_2 expression like above but now guess with an universal quantifier that it fails exactly at this step, i.e. does not find any of the letters it is looking for. The disjunction of all these expressions says that φ fails. Moving the disjunctions behind the quantifiers and using de Morgan's law we have that L_φ can be be expressed by a Π_2 first order expression. This shows that φ is in $\Sigma_2^L \cap \Pi_2^L = \Delta_2^L$. For larger $k \geq 2$ the same procedure on the topmost path of atnext/atprevious operators leads to an Σ_2 expression with at_{k-1} subexpressions, for which the Π_{k-1} first-order expressions – which exist by induction hypothesis – are taken, resulting into a Σ_k expression because two subsequent $\forall$ levels collapse into one. Dually, a Π_k expression is obtained by plugging the Σ_{k-1} translation of an at_{k-1} subexpressions into a Π_2 expression for the topmost path of atnext/atprevious operators.

(b) Given the classical result FO = LTL = STARFREE mentioned in the Introduction, the direction $\subseteq$ follows from (a), and the direction $\supseteq$ follows from the fact that an until/since expression can be translated into an at expression, see Table 1.

(c) Let D_{k+1} be the languages $L_{\delta_{k+1}}$ defined above after the examples. D_{k+1} is not only in $L(\mathrm{at}_k)$ by the at expression δ_{k+1} which has at-depth k but can even be expressed by an until/since expression with until/since depth $k-1$.

(d) the $\subseteq$ relation holds by definition, and the properness follows from (c) together with (a).

q.e.d.

We are in this paper interested especially in Δ_2^L, i.e. the languages L for which there exist two quantifier-free expressions $\varphi_1(\vec{x},\vec{y})$ and $\varphi_2(\vec{u},\vec{v})$ over signature $[<]$ such that L equals the language expressed by $\exists\vec{x}\forall\vec{y}\varphi_1(\vec{x},\vec{y})$ and also equals the language expressed by $\forall\vec{u}\exists\vec{v}\varphi_2(\vec{u},\vec{v})$. It holds that a language is in Δ_2^L iff its syntactic monoid is in the variety **DA** [Sch76, PW97], where **DA** is defined as the class of finite monoids which fulfill the equality $(xyz)^\omega y (xyz)^\omega = (xyz)^\omega$. This equality $\Delta_2^L = L(\mathbf{DA})$ was shown by Pin & Weil [PW97], building on earlier research on **DA** and Δ_2^L, started by Schützenberger [Sch76]. By this equality we let DA (in normal face) in this paper be another name for Δ_2^L from the Straubing-Thérien hierarchy. DA has many characterizations, see the survey [TT02], for example the following one by Schwentick et al. [STV01], building on earlier notions and results by [TW98].

DA is the class of *turtle languages*, which we define here as languages expressed by certain restricted at_1 expressions: A *turtle expression* is either the expression "**true**", or an expression

$$t\ \mathbf{atnext}^*\ b$$

$$t\ \mathbf{atprevious}^*\ b$$

where t is a turtle expression and b is a letter. A *turtle language* is a finite Boolean combination of languages L_φ for a turtle expression φ. By definition, turtle languages are a subset of $L(\mathrm{at}_1)$. Schwentick et al. [STV01] showed that the class of turtle languages equals DA. This gives the direction $\supseteq$ of the following corollary, the inclusion $\subseteq$ is the case $k = 1$ from Proposition 1.

Corollary 1 $L(\mathrm{at}_1) = \mathrm{DA}$

We generalize turtle languages a little bit. A *search expression* is either the expression **true**, or a single letter b, or an expression

$$s\ \mathbf{atnext}^*\ (b_1 \vee \cdots \vee b_m)$$

$$s \ \mathbf{atprevious}^* \ (b_1 \vee \cdots \vee b_m)$$

where s is a search expression and $b_1, \ldots, b_m$ are letters. A *search language* is a finite Boolean combination of languages L_φ for a search expression φ. The examples φ_1, φ_2 and φ_3 from above are search expressions, φ_2 and φ_3 are sketched as the left and the middle expression in Figure 1. Search expressions are slightly more general than turtle expressions, ϕ_2 is an example of a search expression which is not a turtle expression. Because search languages are in between turtle languages and $L(\mathrm{at}_1)$ they also represent exactly the DA languages. Moreover, it obviously holds the equivalence

$$(\varphi_1 \wedge \varphi_2) \ \mathbf{atnext}^* \ \varphi \quad \equiv \quad (\varphi_1 \ \mathbf{atnext}^* \ \varphi) \wedge (\varphi_2 \ \mathbf{atnext}^* \ \varphi), \tag{1}$$

similarly for disjunction and negation, for **atprevious***, and also for the non-stared versions. By the iterated application of this equivalence it is possible to move all Boolean operators within an at_1 expression, besides the disjunctions of the letters at the right leaves, to the top of the expression (with at most a quadratic blow-up of the size of the expression), resulting in a finite number of search expressions combined by a Boolean expression. Therefore, search expression may serve as a kind of "normalized" form of at_1 expression, even in the context of pointed languages, see below. Search expressions, and turtle expression as a special case, may be considered as a list of searches $(s_1, \ldots, s_m)$ where s_1 corresponds to the topmost atnext/atprevious operator in the expression tree and s_m is the bottom one. Call m the length of the search expression. Each s_i is determined by (i) the set of letters searched for, (ii) the direction right/left given by the operator type **atnext** resp. **atprevious**, and (iii) – only for s_m – the information about the final check which is **true** or a letter b.

A *pointed word* over alphabet Σ is a pair (x, i) such that $x \in \Sigma^*$ and i is a position of x, i.e. $i \in \{1, \ldots, |x|\}$. A *pointed language* over an alphabet Σ is a set of pointed words over alphabet Σ. A *pointed class* is a set of pointed languages. Note that in order to define the semantics $x \models \varphi$ for expressions of a temporal logic like at expressions (likewise for until/since) we actually first had to specify the meaning of $(x, i) \models \varphi$ for pointed words before we could define the meaning of $x \models \varphi$, this seems to be typical and even unavoidable for temporal logics with both future and past operators. Let φ be a temporal logic expression, like an atnext/previous expression or an until/since expression, over alphabet Σ. The pointed language P_φ is defined to consist of the the pointed words (x, i) over alphabet Σ^* such that $(x, i) \models \varphi$, while the pointed language P^0_φ consists of the pointed words (x, i) over alphabet Σ^* such that $x \models \varphi$, independent of i. For a set F of at expressions the pointed class $P(F)$ is defined as the set of all pointed languages which are a finite Boolean combination of pointed languages P_φ or P^0_φ for expressions $\varphi \in F$.

Note that by the equivalence in equation (1) above it holds

$$P(\mathrm{S}) = P(\mathrm{at}_1) \tag{2}$$

where S is the set of search expressions. Therefore, search expressions are a kind of normal form for at_1 expressions, even in the context of pointed languages. Because it holds $L(\mathrm{T}) = L(\mathrm{S}) = L(\mathrm{at}_1) = \mathrm{DA}$, where T is the set of turtle expressions, one may expect that $P(\mathrm{T}) = P(\mathrm{S})$. This is not the case: Let Q be the pointed language over alphabet $\{a, b\}$ consisting of the pairs (x, i) such that if $x_i = a$ then $x_{i+1} = b$. This is a language in $P(S)$, as witnessed by the Boolean combination $\neg P_{s_1} \vee P_{s_2}$ of the search expressions $s_1 =$ "a" and $s_2 =$ "b **atnext*** $(a \vee b)$", but there is no turtle language recognizing Q: if its turtle expressions have maximal length k they will fail to distinguish the pointed words $((ab)^k aa(ab)^k, 2k+1)$ and $((ab)^k ab(ab)^k, 2k+1)$, simply because for both strings all turtles

starting at the pointer position $2k+1$will succeed, and the ones starting at the border will not reach position $2k+2$. This shows that $P(\mathrm{T}) \neq P(\mathrm{at}_1)$, i.e. turtle expressions are not powerful enough for a "normal form" of at_1 expressions for pointed languages.

Let two languages L_1, L_2 over the same alphabet Σ and a letter $b \in \Sigma$ be given. The *triple language* $P(L_1, b, L_2)$ is the set of pointed words (x, i) such that $x_{<i} \in L_1$, $x_i = b$, and $x_{>i} \in L_2$. For a class C its class of pointed languages $P(C)$ consists of the finite Boolean combinations of tripel languages $P(L_1, b, L_2)$ such that L_1 and L_2 are from C (and of course L_1 and L_2 are over the same alphabet of which b is also an element).

The following lemma is our main technical result. It gives a characterization of the pointed languages of DA in terms of a temporal logic – which was not yet known to exist.

Lemma 1 $P(\mathrm{at}_1) = P(\mathrm{DA})$.

Proof. $P(\mathrm{at}_1) \subseteq P(\mathrm{DA})$: Let according to equation (2) a language L from $P(\mathrm{at}_1)$ be given as a Boolean combination of pointed languages $P(\varphi_1), \ldots P(\varphi_k), P^0(\varphi_{k+1}), \ldots, P^0(\varphi_n)$ for search expressions $\varphi_1, \ldots, \varphi_k, \varphi_{k+1}, \ldots, \varphi_n$. Consider for such a search expression $\varphi = (s_1, \ldots, s_m)$ all its "factor search expressions", i.e. all search expressions of the form $(s_i, \ldots, s_j)$ such that $1 \leq i \leq j \leq s_m$. Let Q_φ be the set of pointed languages consisting of the triple languages $P(\Sigma^*, a, \Sigma^*)$, $P(L_f, a, \Sigma^*)$ and $P(\Sigma^*, a, L_f)$ such that a is a letter from Σ and f is a factor search expression of φ. Note that every language L_f is in $L(\mathrm{at}_1) = \mathrm{DA}$, therefore Q_φ consists of tripel languages from DA. Given a pointed word (x, i), these pointed languages from Q_φ can be used to find the answer to the questions whether $x_i = a?$, $x_{<i} \in L_f?$ and $x_{>i} \in L_f?$ (via $P(\Sigma^*, a, \Sigma^*)$, $P(L_f, a, \Sigma^*)$ and $P(\Sigma^*, a, L_f)$, respectively).

It is now possible to determine whether $(x, i) \in P_\varphi$ by the following "decision tree algorithm" which only asks Boolean membership queries of the above mentioned kind to the pointed languages in Q_φ: The algorithm first finds out what letter x_i is. Then it "traces" the behaviour of φ on x, started in i: if the first search s_1 is a right (= atnext) search, it is queried whether it holds $x_{>i} \in L_{(s_1)}$. If the answer is 'false' then we know that s_1 will hit the right border, and so also φ will fail on (x, i). Otherwise we ask if $x_{>i} \in L_{(s_1, s_2)}$. Again, if this is a right search and fails then we know that also φ will fail on (x, i). If it is a left search and fails then we know that the search s_2 has hit the left border of $x_{>i}$, and so in (x, i) the search (s_1, s_2) is traversing the position i. If the letter x_i is in the set of letters s_2 was searching for then we know that i is the position on which the search (s_1, s_2) is successful – in that case we continue the tracing algorithm by tracing the search $(s_3, \ldots, s_m)$ on x, starting in i. If the letter x_i is not in the set of letters s_2 was searching for then we continue the tracing algorithm by tracing the search of $(s_2, \ldots, s_m)$ on x, starting in i.

This way, we can trace the full search expression $\varphi = (s_1, \ldots, s_m)$ via queries $x_i = a?$, $x_{<i} \in L_f?$ and $x_{>i} \in L_f?$, in other words, success of φ on (x, i) can be expressed by a Boolean function on Q_φ. Therefore the pointed language P_φ is a Boolean combination of the triple languages from Q_φ. The pointed languages P^0_φ instead of P_φ can be decided by the same by the idea and the same construction of the set Q_φ, but in this case the tracing is not started at the pointer but at the border of the word. For the union Q of all $Q_{\varphi_1}, \ldots, Q_{\varphi_k}, Q_{\varphi_{k+1}} \ldots, Q_{\varphi_n}$ it holds that every set $P(\varphi_1), \ldots P(\varphi_k), P^0(\varphi_{k+1}), \ldots, P^0(\varphi_n)$ is a finite combination of pointed languages in Q. Because L is a finite Boolean combination of the former, it is also a finite Boolean combination of the pointed languages in Q, i.e. $L \in P(\mathrm{DA})$.

$P(\mathrm{DA}) \subseteq P(\mathrm{at}_1)$:

Consider a pointed language A from $P(\mathrm{DA})$ over alphabet Σ. By definition this is a finite Boolean

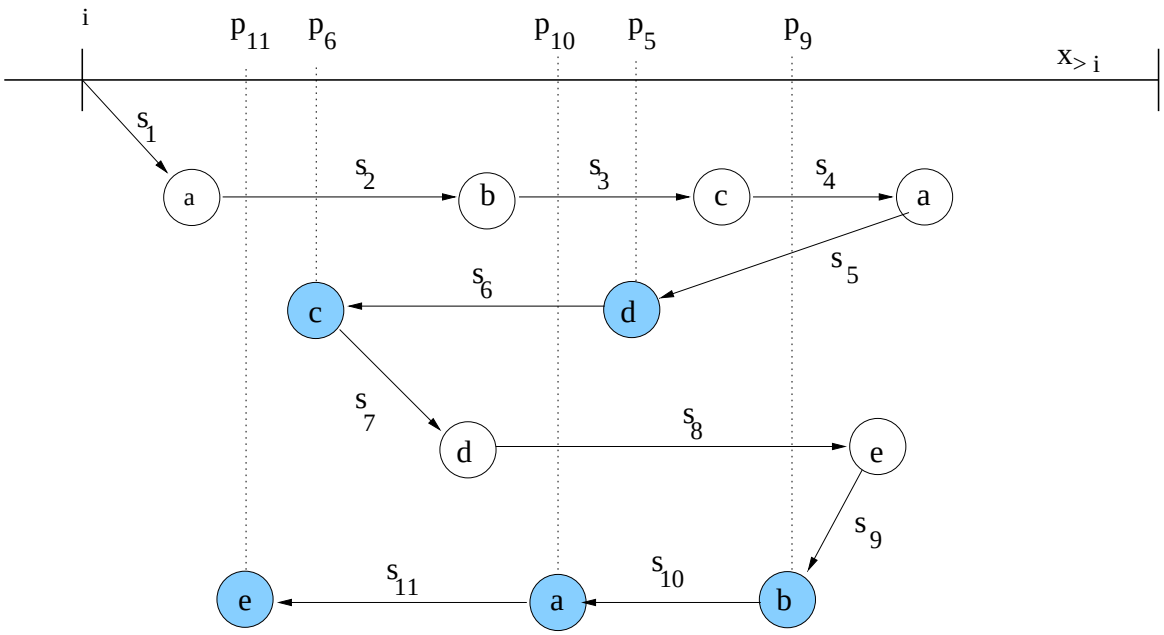

Figure 2: A turtle with its left and right searches

combination of triple languages $P(K_1, b, L_1), \ldots, P(K_n, b_n, L_n)$ such that $K_1, L_1, \ldots, K_n, L_n \in \mathrm{DA}$, all of them over the same alphabet Σ, and $b_1, \ldots b_n \in \Sigma$. Let L be one of the languages L_i. We first show as the main step of the proof that the pointed language $\{(x, i) \mid x_{>i} \in L\}$ is in $P(\mathrm{at}_1)$.

Because $L \in \mathrm{DA}$ there exists a finite set of turtle expressions $t_1, \ldots, t_c$ such that L is a finite Boolean combination of $L_{t_1}, \ldots, L_{t_n}$. Let $t = (s_1, \ldots, s_k)$ be one of these turtle expressions. Assume as the first case that s_1 is a right (= atnext) search. We show, as another sub-step of the proof, that the following pointed language can be described by an at_1 expression e: the set of pairs (x, i) such that the turtle expression t is successful on $x_{>i}$, in other words $(x, i) \in P_e$ iff $x_{>i} \in L_t$ for all pointed words (x, i).

This will be done by a careful checking of the string $x_{>i}$ according to the known moves of the turtle $t = (s_1, \ldots, s_k)$. Let $(R_1, L_1, \ldots, R_p, L_p, R_{p+1})$ be the partition of $\{0, \ldots, k\}$ such that each set R_j and L_j is a nonempty set of consecutive indices of right resp. left searches, and this partition is ordered consecutively, i.e. the indices in R_{j+1} and L_{j+1} follow the ones in L_j resp. R_j. The last set R_{p+1} may be empty – it is the last set of right searches, so it will not matter for our construction. In the example in Figure 2 this partition is (R_1, L_1, R_2, L_2) with $R_1 = \{1, 2, 3, 4\}$, $L_1 = \{5, 6\}$, $R_2 = \{7, 8\}$, and $L_2 = \{9, 10, 11\}$. Let L be the union of the L_i. In the example in Figure 2, $L = \{5, 6, 9, 10, 11\}$. We guess an order $\pi : \{1, \ldots |L|\} \to L$ of the success positions p_i of the left searches, i.e. $p_{\pi(1)} < p_{\pi(2)} < \cdots < p_{\pi(|L|)}$. This order π has to respect the reverse order of each set L_r, i.e. for $s, s' \in L_r$ it follows from $s > s'$ that $p_{\pi(s)} < p_{\pi(s')}$. Equality instead of $<$ in the guess $p_{\pi(1)} < p_{\pi(2)} < \cdots < p_{\pi(|L|)}$ is allowed given that the searched letters are the same. There are only finitely many such orders on L, including the variants with equality. In the example in Figure 2 the guessed order is $p_{11} < p_6 < p_{10} < p_5 < p_9$. Our intention is to guess the order on the actual success

positions of the left searches of the turtle, in case it is successful within $x_{>i}$. We will show that if the turtle is not successful within $x_{>i}$ then for every guess it it will be noticed by our at_1expression that the turtle fails, and if the turtle is successful then at least for the "real guess" of the success positions (and maybe some others, too), the order the at_1 expressions will tell that the turtle does not pass the position i. Let π be such a guessed order. We build the following at_1 expressions $e_1, \ldots, e_p$ which will tell whether t will pass i on a left search.

Consider $R_1 = \{1, 2, \ldots, r\}$ and $L_1 = \{r+1, r+2, \ldots, r'\}$. Let for a search s_i of the turtle t l_i be the letter searched for. We want to make sure with an at_1 expression e_1 that the two greedy searches, starting in i, for the letters $l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{r+1}$ (called it the L search) and $l_1, l_2, \ldots, l_r$ (call it the R_1 search) have success positions $q_{\pi(1)} < q_{\pi(2)} < \cdots < q_{r+1}$ and $q_1 < q_2 < \cdots < q_r$ such that $q_{r+1} < q_r$. In the example in Figure 2 this means that the greedy search for the letters e, c, a and then d in that order finishes before the greedy search for the letters a, b, c and then again a in that order, note this will guarantee that the left searches can be done safely, i.e. without passing i. Guess an order ρ representing the overlapping order of the success positions of these two greedy searches, in the example that would be the $q_1 < q_{11} < q_6 < q_2 < q_{10} < q_5 < q_3 < q_4$. Equality instead of $<$ is not only is allowed but necessary when two adjacent searches in this list search for the same and are each from a different original lists. Note that there are only finitely many such guesses of such a ρ, including the equality case. Given ρ, it can be expressed with an at_1 formula that the two greedy searches give positions in exactly this order: Let z_1 be the expression "l $\mathbf{atnext}^*$ $(l \vee l')$" where l is the letter of the first success position in ρ and l' is the letter of the first search in the other list. Note that z_1 checks if the first success position in ρ is guessed the right way. Let z_2 be the expression $(l''$ $\mathbf{atnext}^*$ $(l'' \vee l''')$ $\mathbf{atnext}^*$ $(l \vee l')$ where l'' is the letter of the second search in ρ, and l''' is the letter of the next success position in ρ of the other list. For example, z_2 in the example of Figure 2 for the guessed order is "$(e$ $\mathbf{atnext}^*$ $(e \vee c))$ $\mathbf{atnext}^*$ $(a \vee e)$". Continue building z_n this way until the whole list ρ is processed. Note that the case that both lists may at some point search for the same letter is covered by the construction. Let the final at_1 expression, call it e_ρ, be the conjunction of all the z_j. It evaluates to true if and only if the two greedy searches are succesful on $x_{>i}$ and the success positions have exactly the order guessed by ρ. Let e_1 be the disjunction of e_ρ over all possible guesses ρ. e_1 evaluates to true if the search the R_1 letters is faster than the search for the L letters. Note that in that case the turtle can do first the right search R_1 (without hitting the right border) and then can do the left search of L_1 savely: all letters it searches for will appear in that (reverse) order within $x_{>i}$, at least the success positions from the above greedy search will be a break (or a backing) point for the left searches Of course, in general the success positions of the left searches will be larger than the corresponding success positions of the greedy right searches.

Still having the guess π fixed, we build an expression e_2 telling whether the left searches of L_2 are save: this will be done basically like for L_1 but now the two greedy right searches, R_2 vs. L, start not at position i but at the success position of the last search of L_1. In the example in Figure 2 this starting position will be p_6, and the question is whether from there the letters a, d and then e can be found earlier than the letters d and then e. The expression e_2 is obtained by the translating the turtle searches $s_1, \ldots s_{r'}$ of R_1 and L_1 into the corresponding at_1 expression f. Then "f $\mathbf{atnext}^*$ g" is the desired at_1 expression telling whether the left searches of L_2 are save. Like e_2 we build for all following pairs of R_j and L_j of the partition of t the corresponding expression e_j. Define as a final e_{p+1} the at_i expression for t itself, in order to gurantee that also the last right search is successful. The conjuction of all these e_j, call it e_π will tell whether for the guess π it will be save for the turtle t to run on $x_{>i}$ without hitting the position i.

The disjunction of all e_π for all guesses of orders π, call it e, will tell whether the run for the turtle t will stay within $x_{>i}$: In case the turtle stays within $x_{>i}$ the "real" order on the success position of its left searches is such an order π that e_π will evaluate to true, note that in that case there may also exist other π' such that $e_{\pi'}$ evaluates to true, but that does not matter. And if the turtle leaves $x_{>i}$, say during left search L_j, then this will be "noticed" by e_π for every π. Therefore, the expression e has the desired property: $(x,i) \in P_e$ iff $x_{>i} \in L_t$ for all pointed words (x,i).

This shows that for a single turtle expression t (with a starting right search) one can construct an at_1 expression e such that $(x,i) \in P_e$ iff $x_{>i} \in L_t$ for all pointed words (x,i). If the first search command of the turtle expression is a left search then the same construction of e works – but now with an empty set R_1, see above, an the runs of the turtle t and its prefixes t' have to start at the right border, and exactly for this we need the "border' semantics of of pointed languages: $(x,i) \in P^0_{t'}$ iff $x_{>i} \in L_{t'}$. This still shows that $\{(x,i) \mid x_{>i} \in L_t\}$ is in $P(\mathrm{at}_1)$.

Because the above given $L \in \mathrm{DA}$ is finite Boolean combination of such $L_{t_1}, \ldots, L_{t_n}$, and each of them is in $P(\mathrm{at}_1)$, also L is in $P(\mathrm{at}_1)$.

We have shown that for a language $L \in \mathrm{DA}$ the pointed language $\{(x,i) \mid x_{>i} \in L\}$ is in $P(\mathrm{at}_1)$. By symmetry it holds that for a language $K \in \mathrm{DA}$ the pointed language $\{(x,i) \mid x_{<i} \in K\}$ is also in $P(\mathrm{at}_1)$. This means that a triple language $P(K,b,L)$ with $K, L \in \mathrm{DA}$ is in $P(\mathrm{at}_1)$ by the intersection of these two together with the pointed language P_b. Because the given pointed language A from $P(\mathrm{DA})$ is a finite Boolean combination of such triple languages $P(K,b,L)$ from DA, also A itself is in $P(\mathrm{at}_1)$. This finishes the proof of the direction $\supseteq$.

q.e.d. (Proof of Lemma 1)

The block product of classes of languages is defined as follows – purely in terms of languages. The *block product* $K \,\square\, (P_1, \ldots, P_n)$ of a language K over alphabet $\{0,1\}^n$ and an n-tuple of pointed languages $P_1, \ldots, P_n$ over alphabet Σ consists of the words $x \in \Sigma^*$ such that the word

$$(P_1(x,1), \ldots, P_n(x,1)) \;\; (P_1(x,2), \ldots, P_n(x,2)) \cdots (P_1(x,|x|), \ldots, P_n(x,|x|))) \tag{3}$$

is in K. The block product $\mathcal{K} \,\square\, \mathcal{J}$ of two classes of languages $\mathcal{K}$ and $\mathcal{J}$ is the set of block products $K \,\square\, (P_1, \ldots, P_n)$ such that $K \in \mathcal{K}$ and $P_1, \ldots, P_n \in P(\mathcal{J})$.

The block product is in general not associative, see for example [TS02]. Therefore, we have two extrem cases (and many in between) concerning the bracketing: The *strongly iterated block product of n languages* $K_1, \ldots, K_n$ is defined as

$$K_1 \,\square\, (K_2 \,\square\, (\ldots (K_{n-1} \,\square\, K_n) \ldots))$$

while the *n-fold weakly iterated block product* is defined as

$$((\ldots (K_1 \,\square\, K_2) \ldots) \,\square\, K_{n-1}) \,\square\, K_n.$$

$\mathrm{DA}^{n\square}$ and $\mathrm{DA}^{n\square_w}$ are the sets of all n-fold strongly, resp. weakly, iterated block products of DA languages. It holds $\mathrm{DA}^{n\square_w} \subseteq \mathrm{DA}^{n\square}$, see for example [TS02].

The following Theorem is an easy application of the so-called block product substition Lemma from [TW04][Th. 6] to the result above saying that the pointed languages of at_1 expressions and of DA languages coincide. Note that exactly at_{k+1} is obtained when replacing letters in at_k expressions by at_1 expressions, in the notation of [TW04] this denoted as $\mathrm{at}_{k+1} = \mathrm{at}_k \circ \mathrm{at}_1$. The prerequisites of the block product/substitution lemma are fulfilled: $L(\mathrm{at}_k) = \mathrm{DA}^{k\square_w}$ is given by induction hypothesis, $L(\mathrm{at}_1) = \mathrm{DA}$ is given by Prop. 1, and and $P(\mathrm{at}_1) = P(\mathrm{DA})$ is our main Lemma 1.

Theorem 1 *It holds for every $k \geq 1$:*

$$L(\mathrm{at}_k) = \mathrm{DA}^{k\Box_w}.$$

Compare this result with the following from Thérien & Wilke [TW04] which characterizes the levels of the until/since hierarchy. MNB is the set of languages which are unions of the following equivalence classes: two words are equivalent iff they contain the same set of letters and the order of first positions of occurrences of these letters is the same, both when seen from the left and when seen from the right side, see [TW04]. Note that MNB is a proper subset of DA.

Theorem 2 ([TW04]) *It holds for every $k \geq 0$:*

$$L(\mathrm{us}_k) = \mathrm{DA} \,\Box\, \mathrm{MNB}^{k\Box_w}$$

It was shown by Etessami & Wilke [EW96] that the Σ_2^L languages $\{a,b\}^*bbb\{a,b\}^*$, $\{a,b\}^*bbbbb\{a,b\}^*$, etc., are witnesses for the properness of the inclusions $\mathrm{us}_0 \subset \mathrm{us}_1 \subset \mathrm{us}_2 \subset \ldots$ of the levels of the until/since hierarchy. By the the easy observation that each of these language is in $\mathrm{DA} \,\Box\, \mathrm{DA}$ we have the following result.

Theorem 3 ([EW96]) $L(\mathrm{at}_2) = \mathrm{DA} \,\Box\, \mathrm{DA}$ *contains for every $n \geq 1$ languages from $L(\mathrm{us}_n) \backslash L(\mathrm{us}_{n-1})$.*

The above witnesses show that until/since depth is not the right measure for a starfree language L if one is interested in a space efficient model-checking algorithm of L: the languages $\{a,b\}^*b^{2k+3}\{a,b\}^*$ have until/since depth k but can be model-checked uniformly with a log-space algorithm running in quadratic time since they all are in at_2, see end of Section 2. The at-depth for these languages gives a better estimate of their model-checking complexity, though it is still one too high. Note that the witness languages have a practical significance: they express a certain kind of fairness for processes, see [EW96].

In [Bo04] the Σ_2^L languages $\{1,2\}^*11\{1,2\}^*$, $\{1,2,3\}^*11\{1,2\}^*11\{1,2,3\}^*$, etc., were proven to be witnesses for the properness of the inclusions $\mathrm{DA} \subset \mathrm{DA} \,\Box\, \mathrm{DA} \subset \mathrm{DA}^{\Box_3} \subset \ldots$, etc.

Theorem 4 ([Bo04]) Σ_2^L *contains for every $n \geq 1$ languages from $L(\mathrm{at}_n) \backslash L(\mathrm{at}_{n-1})$.*

The preceeding results allow to determine for every two given levels of each of the three hierarchies (at, until/since and dot-depth) whether one is included in the other or not, see also Figure 3.

Theorem 5 *Let $m, n \geq 1$.*
(a) $L(\mathrm{us}_{m-1}) \subseteq L(\mathrm{at}_n) \iff m \leq n$. $L(\mathrm{at}_n) \subseteq L(\mathrm{us}_{m-1}) \iff n = 1$.
(b) $L(\mathrm{at}_m) \subseteq \Delta_{n+1}^L \iff m \leq n$. $\Delta_{n+1}^L \subseteq L(\mathrm{at}_m) \iff n = 1$.
(d) $L(\mathrm{us}_{m-1}) \subseteq \Delta_{n+1}^L \iff m \leq n$. $\Delta_{n+1}^L \subseteq L(\mathrm{us}_{m-1}) \iff n = 1$.
(e) $L(\mathrm{us}_{m-1}) \subseteq L(\mathrm{us}_{n-1}) \iff L(\mathrm{at}_m) \subseteq L(\mathrm{at}_n) \iff \Delta_{m+1}^L \subseteq \Delta_{n+1}^L \iff m \leq n$.

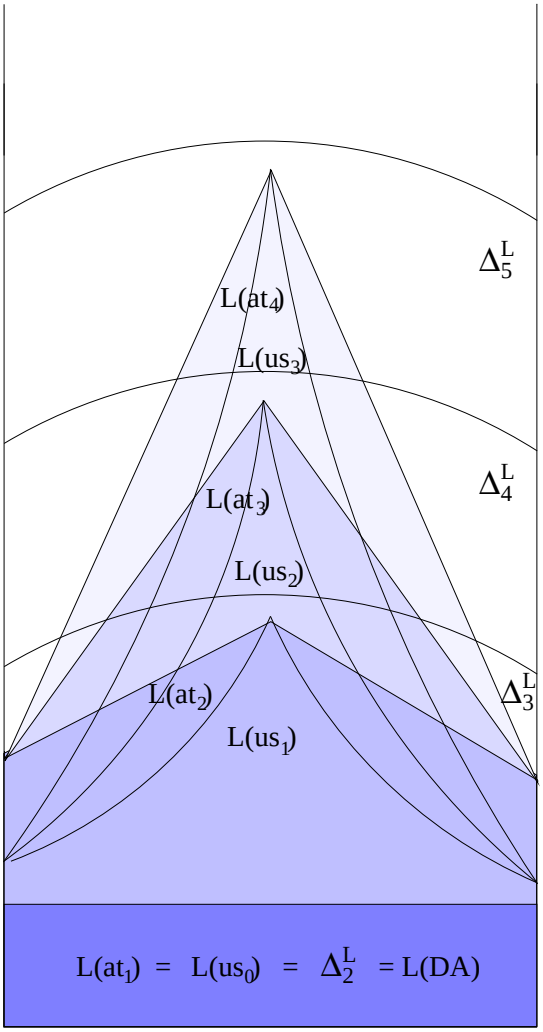

Figure 3: The at hierarchy (shaded) compared with the dot-depth and the until/since hierarchy

This theorem shows that the until/since hierarchy is properly contained in the at hierarchy which again is properly contained in the dot-depth hierarchy (as given by its Delta levels). The properness holds for every non-bottom level. It is irritating that the numbering of the levels of the three hierarchies is shifted: the three bottom level of the three hierarchy conincide (= DA) but this bottom level has number 0 in the until/since hierarchy, number 1 in the at hierarchy, and number 2 in the dot-depth hierarchy. Note that a level 0 of the at hierarchy could be defined, it consists of the trivial languages. We prefer to ignore this level and call at_1 the bottom level.

Let the at-depth, the until/since depth, and the Δ-dot-depth of a starfree regular language L be the smallest number k such that $L \in L(\mathrm{at}_k)$, $L \in L(\mathrm{us}_k)$, and $L \in \Delta_k$, respectively. Then it follows from the above theorem: For every starfree language it holds that its at-depth is greater or equal its until/since depth plus 1 and is smaller or equal its Δ-dot-depth minus 1.

4 Adding the Operators next and previous

Up to now our temporal logic did not contain the $\oplus$ (next) and the $\ominus$ (previous) operator. We will see that it only adds some "generalized definite" complexity, like the Cohen/Brzozowski hierarchy adds just some "generalized definite" complexity to the Straubing/Thérien hierarchy. This effect was also noticed for the until/since hierarchy of [TW04].

The semantics of the next and the previous operators $\oplus$ and $\ominus$, which are both 1-ary operators, is defined first for pointed words:

$(x, i) \models \oplus\psi$
iff $i + 1$ is still a position in x and $(x, i + 1) \models \psi$.

$(x, i) \models \ominus\psi$
iff $i - 1$ is still a position in x and $(x, i - 1) \models \psi$.

For a word x define $x \models \varphi$ as follows:

$x \models \oplus\psi$
iff $|x| > 0$ and $(x, 1) \models \psi$.

$x \models \ominus\psi$
iff $|x| > 0$ and $(x, |x|) \models \psi$.

Note that the next and the previous operators are quite different from the atnext and atprevious operators, though the names are similar. Besides that the former are 1-ary and the latter are 2-ary there is the crucial difference that the former only have a "local" behaviour, as we will see below, while the latter can search "globally".

Let an AT expression be an LTL expression using the atnext/atprevious operators and the next/previous operators. Extend the definition of at-depth to these expressions by

$$d_{\mathrm{at}}(\oplus\psi) = d_{\mathrm{at}}(\ominus\psi) = d_{\mathrm{at}}(\psi),$$

i.e. the next and previous operators can be used "for free". Let AT_k be the set of AT expression of at-depth $k \geq 1$.

The class of *generalized definite languages* GDEF consists of the languages L for which membership can be determined by a constant-size prefix and suffix of the word, i.e. for which there exists a constant c such that for all words x with length $\geq 2c$ it holds $x \in L$ iff $x_{<c}x_{>|x|-c} \in L$.

Let np denote the set of expressions using only the letter symbols, the Boolean operators and the next and the previous operator. It is easy to see that it holds $L(\mathrm{np}) = \mathrm{GDEF}$, and in [TW04][Lemma 10] it is shown

$$P(\mathrm{np}) = P(\mathrm{GDEF}). \tag{4}$$

As a similar characterization, GDEF is the set of languages expressible without quantifiers over the signature $[<, S, P, \min, \max]$, where S and P are the successor and predecessor function, resp. [Th82]. The classes Σ_k^B and Π_k^B of the Cohen-Brzozowski hierarchy are defined as the languages definable with a Σ_k and Π_k quantifier alternation prefix over this signature $[<, S, P, \min, \max]$ instead of the one containing only $[<]$. It holds

$$\Sigma_k^B = \Sigma_k^L \,\square\, \mathrm{GDEF}$$

for all $k \geq 0$, likewise for the Π_k^B and Δ_k^B levels. The same effect was shown for the until/since hierarchy [TW04][Th. 7]:

$$\mathrm{US}_k = \mathrm{us}_k \,\square\, \mathrm{GDEF}.$$

It comes with no surprise that the analogue also holds for the at hierarchy:

Theorem 6 *It holds for every $n \geq 1$:*

$$L(\mathrm{AT}_n) = \mathrm{at}_n \,\square\, \mathrm{GDEF} = \mathrm{DA}^{n\square_w} \,\square\, \mathrm{GDEF}$$

This theorem follows from equality 4 above together with the block product/substitution lemma from [TW04] and the the observation that $\mathrm{AT}_k = \mathrm{at}_k \circ \mathrm{np}$: The next and previous operators within for at expressions can be moved to the leaves of the expression tree, like this is also possible for until/since expressions [TW04]. This can easily be checked, as it holds for example the following equivalence:

$$\oplus(\varphi \ \mathbf{atnext} \ \psi) \quad \equiv \quad (\oplus\varphi \wedge \oplus\psi) \vee (\varphi \ \mathbf{atnext} \ \psi).$$

By the characterization 6 one can again tell for every pair of levels of the three hierarchies whether one is included in the other. The witnesses of the Theorems 3 and 4 can be extended to this case by shuffling the witness languages with a neutral letter.

Theorem 7 *Let $m, n \geq 1$.*
(a) $L(\mathrm{US}_{m-1}) \subseteq L(\mathrm{AT}_n) \iff m \leq n.\ L(\mathrm{AT}_n) \subseteq L(\mathrm{US}_{m-1}) \iff n = 1.$
(b) $L(\mathrm{AT}_m) \subseteq \Delta_{n+1}^B \iff m \leq n.\ \Delta_{n+1}^B \subseteq L(\mathrm{AT}_m) \iff n = 1.$
(d) $L(\mathrm{US}_{m-1}) \subseteq \Delta_{n+1}^B \iff m \leq n.\ \Delta_{n+1}^B \subseteq L(\mathrm{US}_{m-1}) \iff n = 1.$
(e) $L(\mathrm{US}_{m-1}) \subseteq L(\mathrm{US}_{n-1}) \iff L(\mathrm{AT}_m) \subseteq L(\mathrm{AT}_n) \iff \Delta_{m+1}^B \subseteq \Delta_{n+1}^B \iff m \leq n.$

At the end of this section it should mentioned that the authors started their investigations about the at hierarchy when they were studying the languages and classes determining as so-called leaf

languages the class Δ_k^p of the polynomial hierarchy, see [BLSTT04]. The following line of equalities follows from the fact that already the language D_k from Prop. 1(c) as well as the class Δ_{k+1}^B lead to Δ_{k+1}^p via the leaf language concept, as shown in [BLSTT04], the equalities for the intermediate classes follow from Theorems 5 and 7.

Corollary 2 ([BLSTT04]) $\forall k \geq 1$: $\mathrm{Leaf}^p(D_{k+1}) = \mathrm{Leaf}^p(L(\mathrm{us}_{k-1})) = \mathrm{Leaf}^p(L(\mathrm{at}_k)) = \mathrm{Leaf}^p(\Delta_{k+1}^L) = \mathrm{Leaf}^p(L(\mathrm{US}_{k-1})) = \mathrm{Leaf}^p(L(\mathrm{AT}_k)) = \mathrm{Leaf}^p(\Delta_{k+1}^B) = \Delta_{k+1}^p$.

5 Conclusion, Open Problems, and Acknowledgements

We introduced the atnext/atprevious hierarchy on the starfee regular languages and could show that its n-th level equals the n-fold weakly iterated block product of DA. The main Lemma 1 characterized the pointed languages of DA as the pointed languages defined by at_1 expressions.

The at depth of a starfree language may serve as another measure of its inherent complexity, like dot-depth or until/since depth.

Some open questions the authors could not answer yet:

- Do weak and strong bracketing coincide for block products of DA? For example,

  $$(\mathrm{DA} \,\square\, \mathrm{DA}) \,\square\, \mathrm{DA} = \mathrm{DA} \,\square\, (\mathrm{DA} \,\square\, \mathrm{DA})?$$

  Could this possibly be shown by the methods of the proof of Lemma 1?

- Can the characterization as at_2 help to see whether $\mathrm{DA} \,\square\, \mathrm{DA}$ is decidable or not?

- For the until/since hierarchy the eventually operators are "for free", for the at hierarchy the leftmost nesting is "for free". Is it possible to define depth of expressions of an LTL logic based on a set of operators mentioned in this paper in a way such that certain syntactic occurrences are for free (like for example certain operators, onesided nesting, or subsequent non-alternating occurrences of a pair of operators, etc.) such that the depth-n expressions describe exactly the Δ_{n+1}^L languages of the dot-depth hierarchy? In other words, is there a way of obtaining the dot-depth hierarchy in terms of an LTL hierarchy?

Thanks to Denis Thérien for hints and comments.

References

[Bo04] B. Borchert: *The dot-depth hierarchy vs. iterated block products of DA*, Report 2004-09, WSI Tübingen, 2004

[BLSTT04] B. Borchert, K.-J. Lange, F. Stephan, P. Tesson, D. Thérien, *The dot-depth and the polynomial hierarchy correspond on the Delta levels*, DLT 2004

[EW96] K. Etessami, T. Wilke: *An until hierarchy for temporal logic*, LICS 1996: 108-117

[Ka68] J. KAMP: *Tense Logic and the Theory of Linear Order*, Ph.D. thesis, University of California at Los Angeles, 1968

[Kr84] F. KRÖGER: *A generalized nexttime operator in temporal logic*, Journal of Computer and System Sciences 29(1): 80-98 (1984)

[MP71] R. MCNAUGHTON, S. PAPERT: *Counterfree Automata*, MIT Press, Cambridge MA, 1971.

[PW97] J. E. PIN, P. WEIL: *Polynomial closure and unambiguous product*, Theory Comput. Syst. 30: 383-422 (1997)

[PP86] D. PERRIN, J. E. PIN: *First-order logic and star-free sets*, Journal of Computer and System Sciences 32(3): 393-406 (1986)

[STV01] T. SCHWENTICK, D. THÉRIEN, H. VOLLMER: *Partially-ordered two-way Automata: a new characterization of DA*, DLT 2001: 239-250

[Sch76] M. P. SCHÜTZENBERGER: *Sur le produit de concatenation non ambigu*, Semigroup Forum 13: 47-75 (1976)

[St94] H. STRAUBING: *Finite Automata, Formal Logic, and Circuit Complexity*, Birkhäuser, Boston, 1994.

[TT02] P. TESSON, D. THÉRIEN: *Diamonds are forever: the variety DA*, in Semigroups, Algorithms, Automata and Languages, WSP, 2002, 475-499.

[TS02] H. STRAUBING, D. THÉRIEN: *Weakly iterated block products of finite monoids*, LATIN 2002: 91-104

[TW98] D. THÉRIEN, T. WILKE: *Over words, two variables are as powerful as one quantifier alternation: $FO^2 = \Sigma_2 \cap \Pi_2$* , STOC 1998: 41-47

[TW01] D. THÉRIEN, T. WILKE: *Temporal logic and semidirect products: an effective characterization of the until hierarchy*, SIAM Journal on Computing 31(3): 777-798 (2001)

[TW04] D. THÉRIEN, T. WILKE: *Nesting Until and Since in linear temporal logic*, Theory Comput. Syst. 37(1): 111-131 (2004)

[Th82] W. THOMAS *Classifying regular events in symbolic logic*, Journal of Computer and System Sciences 25: 360-376 (1982)