SPE/IADC-223782-MS

Automatic Online Classification of Drilling Activities

E. Cayeux, NORCE Norwegian Research Centre, Stavanger, Norway; J. Macpherson, Baker Hughes, Houston, Texas, USA; H. U. Brackel, Baker Hughes, Celle, Germany; J. K. Igland and S. Schaefer, Exebenus, Stavanger, Norway

Abstract

Recognition of which drilling activity is occurring at the wellsite is important for calculating key performance indicators (KPIs), diagnosing potential issues during a drilling operation, and providing advice to the automated drilling control system (ADCS). The drilling team is aware of the current drilling activity at any given time, and daily drilling reporting systems categorize drilling operations into activities and sub-activities. Recognizing which drilling activity is being executed in real-time by a computer system is, however, a challenging task. A method has been proposed to uniquely characterize the drilling process state at any instant. This method relies on the decomposition of the possible boundary values of the drilling process into so-called micro-states. The combination of the values of all the micro-states constitutes the drilling process state. The process of estimating the value of each micro-state can be computerized using inputs from measurements and estimations from digital twins. The succession of certain patterns in the flow of drilling process states is characteristic of a specific drilling activity. By recognizing these patterns, it is possible to categorize the current drilling activity. The drilling process is a hierarchical construct, and layers of activities with drilling equipment yield layers of drilling operations. Since the construct is hierarchical it is possible to formulate rules about the relationships between drilling process states and how they characterize drilling activities. In the digital space, a similar technique is Backus-Naur Form (BNF), a standard technique used to describe valid syntax of computer programs in a programming language. BNF rules can define what constitutes a low-level drilling activity, and also specify that only certain combinations of low-level drilling activities make sense. This allows defining drilling activities and sub-activities in a computer-interpretable manner. The use of BNF rules simplifies the implementation of a parser that categorizes drilling activities in real-time using a classical pushdown automaton. Any future adjustments to these rules to accommodate new drilling methods and techniques do not require recoding the drilling activity parser. If there are insufficient states to uniquely identify the drilling activity then the parser can also generate the set of possible drilling activities matching the set of identified states, and it can be used to pro-actively trigger a desired activity based on an entry, for example, in the rig action plan. This work is part of the Drilling and Wells Interoperability Standards (D-WIS) sub-committee of the Drilling System Automation Technical

Section (DSATS). The D-WIS infrastructure provides shared functionalities to facilitate the collaboration of multiple applications during drilling operations. Adding a shared and clearly defined classification of drilling activities allows for high-level synchronization of independent multiple agents working together to enhance drilling process efficiency.

Introduction

Automating the drilling process is a complex task due to the need to manage multiple aspects simultaneously, such as adhering to the general well program, executing detailed rig action plans (RAP), maximizing performance, minimizing risks, and responding to unexpected situations, all while following company-specific standard practices. This interconnectedness in managing a drilling operation calls for a 'system of systems' approach (Manthorpe 1996) to developing drilling automation systems. This approach applies the standard divide-and-conquer strategy to tackle complex problems. However, breaking down a complex problem into smaller, simpler ones requires synchronizing the various systems and subsystems with the evolving timeline of the drilling operation. Each part must recognize the current drilling activity and understand how it relates to other activities needed to achieve the overall drilling program's goals. Traditionally, the concepts of drilling activity and sub-activity have been used by drilling crews to report progress during well construction, but there are multiple conventions, none of which are defined in a way that can be interpreted by computer programs.

It is important to start with a definition of terms, in particular microstates, drilling process states and activities and drilling key performance indicators (KPI). The last three of these terms are used somewhat interchangeably within the well construction industry, which leads to some confusion. The definitions used here are from the Drilling and Wells Interoperability Standards workgroup (Kuilenburg et al. 2024).

- A *microstate* is a boundary condition. It is a constraint or limit of a component or system of the drilling operation. Boundary values, measured or modeled, define the complete set of conditions of the microstates (Cayeux et al. 2024). Taken together, the microstates define drilling process states.
- *Drilling Process States* indicate the condition of the rig, rig equipment, downhole equipment, and the borehole. A change in drilling process state may indicate a change in well drilling activities.
- *Drilling Activities* are drilling processes applied either manually or automatically during well construction. Examples are tripping, making a connection, placing the bit on-bottom, starting pumps, drilling a hard stringer, etc. Some of these are described by entries in the daily drilling report (DDR), by codes such as the IADC DDR Plus codes (Shackleton et al. 2020).
- *Drilling KPIs* are employed to measure the quality of drilling activities. One way to measure the quality is based on time. For instance, a KPI may use an estimate of the drilling process state to measure the time an activity is active. The granularity of the estimate is typically coarser than that required for real-time control of the drilling process (Iversen et al. 2016).

In summary as shown in Fig. 1, microstates define the boundary conditions of the drilling process and in aggregate help define drilling process states, which may indicate a change in activities that form the drilling process. Activities may be controlled both manually and automatically, and themselves define the state of the drilling process. Finally, KPIs measure the duration of an activity (and sub-activities) to evaluate the efficiency of the drilling process.

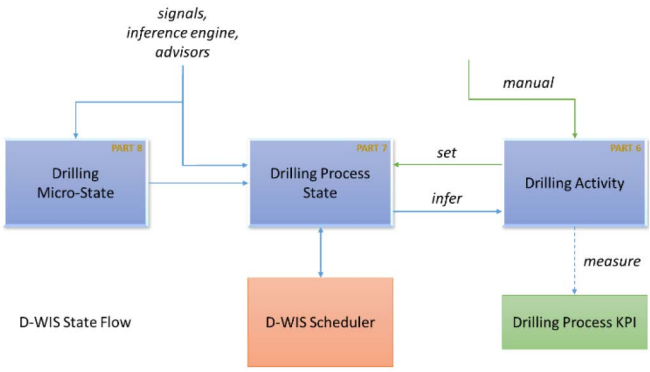

Figure 1—Relationship of microstates, drilling process states, drilling activities and KPI's as developed by the Drilling and Wells Interoperability Standards group.

There are several drilling activity coding systems used in the drilling industry at this time, primarily for daily drilling reports (DDR). Among these are codes from the International Association of Drilling Contractors (IADC), such as the DDR+ codes (Shackleton et al. 2020), codes from the Canadian Association of Oilwell Drilling Contractors (CAODC), WITSML and WITS drilling codes (Jantzen et al. 1989), and each individual operator may have its own codes either based on these or developed individually (Baumgartner et al. 2022).

These codes have some hierarchical structure, such as operations, activities, sub-activities and so on, but there is little formal definition making it difficult to interpret them automatically. The codes were developed primary as a human interpretable classification system, for the daily drilling report. With automation of the drilling system, the codes are not fit-for-purpose, and there is a need to formally define a system for classifying drilling operations that is both interpretable by humans and computer systems.

To enable the automatic generation of KPIs, activities and sub-activities have been classified automatically. For example, the automatic classification of activities using real-time surface logs is achieved through pattern recognition and a hierarchical decomposition of the drilling process into four levels: strategic, tactical, detailed, and machine control (Spoerker 2013). With the availability of digital rig action plans (often referred to as eRAP), efforts have been made to recognize drilling activities by comparing top-side drilling signal patterns with the expected scenarios outlined in the eRAP (Farmanbar et al. 2020). More recently, machine learning techniques have been applied to recognize the drilling process state in order to quantify drilling performance (Yin et al. 2020, Krikor et al. 2022). Often, published works do not clearly differentiate between a process state and an activity, and to the best of our knowledge, there are no precise definitions of standard drilling activities and sub-activities that can be used by a computer program to recognize online drilling activities.

In contrast to drilling micro-states, which can be recognized at any moment by observing the boundary conditions of the drilling system, classifying a drilling activity often requires the observation of a series of these micro-states. This distinction is analogous to a similar dichotomy in programming language parsing: while a single letter can be recognized immediately, a computer program is a stream of letters, and specific sequences of these letters correspond to classifiable and interpretable computer instructions. To facilitate the development of computer program compilers, computer scientists have created methods for recognizing valid programs and converting high-level programming languages into computer processing unit (CPU) instructions (Aho et al. 1985).

The earliest high-level programming languages either had extremely simple syntax, such as LISP (Reilly 2003), or did not adhere to well-defined grammars, like FORTRAN (Backus 1978). It was with the advent of programming languages like ALGOL (Backus et al. 1960) that formal syntax definitions were introduced in the form of Backus-Naur Form (McCracken and Reilly 2003). In fact, it took many years for FORTRAN to adopt a more rigorously defined syntax that aligns with BNF principles (FORTRAN-90)(Adams et al. 1992).

Drawing a parallel with the evolution of computer programming parsing through formal grammars, it is intriguing to explore whether these principles could be applied to the problem of recognizing drilling activities from sequences of drilling micro-states. However, some drilling activities are not clearly observed through measurements, such as moving a rig into position or skidding the drilling module over a slot in a template. As a result, it is difficult to define micro-states in these cases, as they do not directly relate to the boundary conditions of the drilling process itself. Therefore, this paper will limit the scope of recognizing drilling activities to those that can be associated with variations in the boundary conditions of the drilling process.

In this paper, the concepts of lexical and syntactic grammars will be presented, followed by a description of a formal grammar that enables the classification of drilling activities during the execution of a drilling operation. Finally, several examples will be explained.

Lexical and Syntactic Grammar

John Backus and Peter Naur developed the concept of formal grammar for computer languages while working on the ALGOL programming language. Since then, the use of formal grammars in programming languages has become standard practice. The purpose of formal grammar is twofold: first, to define keywords, operators, and identifiers, and second, to specify how these lexical elements can be combined into valid constructs within the language, i.e., define syntactic rules.

The rules defined in a BNF grammar consist of a left-hand side and a right-hand side. The left-hand side represents a syntactic category that can be constructed using the syntactical elements on the right-hand side. If a syntactical element on the right-hand side does not have a corresponding grammar rule, it is referred to as a terminal element. Non-terminal elements can be expanded into sequences of other symbols using grammar rules.

In ANTLR (Another Tool for Language Recognition) (Parr and Quong 1995), the left side is separated from the right side by a colon (:), and a grammar rule is terminated with a semicolon (;). ANTLR is one of the most widely used BNF parsers, supporting various programming languages and allowing the attachment of computer instructions to each grammar rule that is fulfilled while reading a stream of characters. Fig. 2 shows a simple example based on two non-terminals.

```
parserRule : expression ;
```

Figure 2—Example of a grammar rule expressed in ANTLR.

A rule in this context is also referred to as a production. The right-hand side of a production rule can consist of multiple alternatives, which are separated by the | character. Additionally, non-terminal elements on the right-hand side can be the same as the element on the left-hand side, indicating that the production rule is recursively defined. Fig. 3 shows a simple grammar that can be used to parse basic arithmetic expressions.

```
expression: term | expression '+'' term | expression '-' term; // rule 1
term: factor | term '*'' factor | term '/' factor; // rule 2
factor: NUMBER |'(' expression ')'; // rule 3
NUMBER: DIGIT | NUMBER DIGIT; // rule 4
DIGIT: '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'| '8'| '9'; // rule 5
```

Figure 3—Example of a simple grammar to parse basic arithmetic expressions.

In this example, the terminals are ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘(‘, ‘)’, ‘+’, ‘–’ ‘*’ and ‘/’ and there are five production rules. Note that anything following ‘//’ is considered a comment and has no significance in the grammar. Another convention is to enclose terminal characters in single quotes.

Just like in human languages, it is helpful to distinguish between words and phrases. In computer science terminology, a word is called a token, while a phrase is made up of tokens and individual characters. From a linguistic perspective, words or tokens are part of lexical analysis, whereas the parsing of phrases belongs to syntactic analysis. To differentiate between lexical and syntactic rules, the non-terminals in lexical rules are capitalized, while those in syntactic rules begin with a lowercase letter. In the example above, rules 1, 2, and 3 pertain to syntactic production rules, while rules 4 and 5 are used to recognize tokens.

The following stream of characters "2*(3+4)" is then interpreted as follow:

- 2 → rule 5 (digit) and rule 4 (number)
- 2 * → start of rule 2 (term)
- 2 * (→ start of rule 3 (factor)
- 2 * (3 → rule 5 (digit) and rule 4 (number)
- 2 * (3 + → start of rule 1 (expression)
- 2 * (3 + 4 → start rule 5 (digit) and rule 4 (number)
- 2 * (3 + 4) → end of rule 1 (expression) and end of rule 3 (factor) and end of rule 2 (term)

As seen in this example, some rules are initiated but remain unfinished after several additional characters are processed. In programming terms, this means the rules must be stacked while processing the character stream. Since the production rules define relationships between non-terminals, the grammar can be viewed as an automaton. This overall process corresponds to a construct known as a stack automaton (Ginsburg et al. 1967).

For convenience, BNF has been extended with additional concepts, commonly referred to extended BNF. For example, an element followed by '?' indicate s that the element may appear either once or not at all, while an element followed by '*' can appear zero or more times, and an element followed by '+' must appear at least once, similar to the use in regular expressions. These modifiers are often used in conjunction with parentheses to group terms. Additionally, the keyword 'EOS' refers to the end of the stream, and 'NIL' is used to represent an empty token.

This paper loosely adopts the grammar syntax of ANTLR. ANTLR uses an extended BNF syntax. The intention is not to show a complete compilable grammar rather than to convey the idea of using such 'language' to describe drilling operations.

Formal Grammar Applied to Drilling Operations

Now, by considering drilling microstates as an alphabet, a drilling operation can be viewed as a stream of these microstates. Similar to the interpretation of programming languages, it is possible to define production rules that map sequences of drilling microstates to specific drilling activities. In practice, it is not the entire drilling microstate that is treated as a terminal, but rather a pattern within the drilling microstate.

However, drilling microstates alone are not sufficient to distinguish every drilling activity. For instance, the sequence of microstates corresponding to running in hole with a drill string or a plug will likely appear identical. From a human perspective, though, these two operations have different meanings, and it would be advantageous to label them accordingly. To achieve this, additional terminals are introduced in the form of equipment types. One terminal could be "drill string", another "plug", and a third "casing string". With these additions, different run-in-hole activities can now be labeled correctly. Yet, some activities would still be difficult to distinguish from others without an additional context description. An example context is for instance "pressure testing". With the grammar rules defined in Fig. 4, it is then possible to describe precisely a specific activity.

```
EQUIPMENT: 'DrillString' | 'CasingString' | 'LinerString' | 'Plug' | ... ;
CONTEXT: 'PressureTesting' | ...;
activity : runInHole | ... ;
specficActivity: activity 'with' EQUIPMENT 'in' CONTEXT | activity 'with' EQUIPMENT | activity 'in' CONTEXT;
```

Figure 4—Example of grammar rules to define a specific activity, considering the equipment and the context.

Neither BNF nor ANTLR have built-in syntax elements to express sequential or concurrent operations. But this grammar can easily be built (see Fig. 5).

```
activity : ... ;
sequentialActivity : 'SEQ' '['  ACTIVITY (',' ACTIVITY )* ']' ;
concurrentActivity : 'PAR' '[' ACTIVITY (',' ACTIVITY)* ']' ;
flowStrategyActivity : sequentialActivity | concurrentActivity | activity
```

Figure 5—Grammar rules to define sequential or concurrent operations.

Additionally, the concept of equipment extends to fluids. As a result, it is possible to differentiate between circulating a drilling fluid and pumping fluids during a cementing operation, or to distinguish between drilling fluid circulation and drilling fluid displacement. Fig. 6 shows an example concerning the latter case.

```
EQUIPMENT: ... | 'FluidInPlace' | 'NewFluid' | 'SpacerFluid' | 'CementSlurry' | 'DisplacementFluid' | ... ;
activity: ... | mudPumpRunning | cementPumpRunning | ... ;
circulation: pumpingSameFluid 'with' 'FluidInPlace';
pumpingSameFluid: mudPumpRunning 'with' 'FluidInPlace';
displacement: pumpingNewFluid 'with' 'FluidInPlace';
pumpingNewFluid:  mudPumpRunning 'with' 'NewFluid';
mudPumpRunning: ...;
primaryCementation: 'SEQ' '[' pumpingSpacerFluid ',' pumpingCementSlurry ',' pumpingDisplacementFluid ']' 'with' 'FluidInPlace';
pumpingSpacerFluid: cementPumpRunning 'with' 'SpacerFluid';
pumpingCementSlurry: cementPumpRunning 'with' 'CementSlurry';
pumpingDisplacementFluid: cementPumpRunning 'with' 'DisplacementFluid';
cementPumpRunning: ...;
```

Figure 6—Example of grammar rules referencing fluids.

The International Association of Automation (ISA) standard, ISA-88 batch Control (ISA 1995), is a construct in which an activity is delivered by a procedure in combination with optional equipment. A recipe defines the combination, and the construct has a hierarchical structure. The Drilling and Wells Interoperability Standards (D-WIS) group has adopted ISA-88 for the drilling operation.

The activity model is that a well plan is composed of phases, which consists of actions, which themselves contain tasks. The BNF rules are hierarchically organized to follow this decomposition, allowing the interpretation of drilling operations at different levels. It is then possible to associate the notion of phases, actions and tasks to corresponding levels in the BNF grammar (see Fig. 7). To facilitate this mapping, the production rules are numbered using a system similar to Dewey Decimal Classification (Scott and SCOTT 1998). This system relies on hierarchical levels of classification and allows for uniquely identifying nodes in a tree structure. The first level in the tree has only one number: the index of the branch in the tree structure. Additional levels are added, separated by a decimal point ".". The number after the decimal point represents the index of the branch below the previous level. For example, the Dewey code "1.2.3" corresponds to the third branch of the second branch of the first branch in the tree structure. By convention, branches are numbered starting from index one.

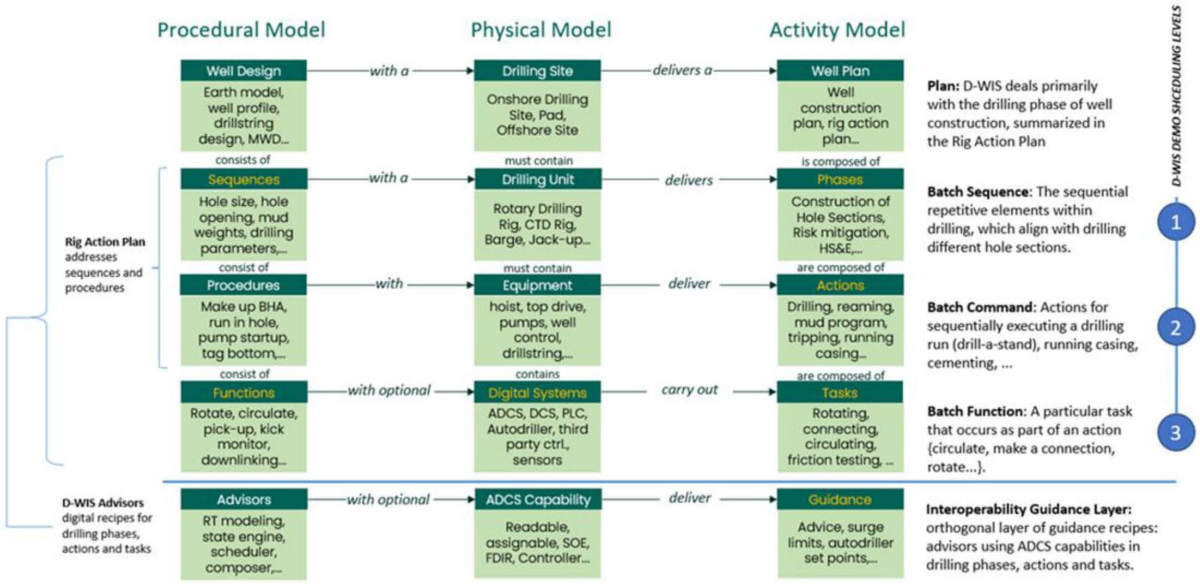

Figure 7—DWIS construct for drilling automation, based on ISA 88, in which an activity is delivered by a procedure with equipment. See text for a description. The lowermost level is a digital layer "the D-WIS Interoperability layer" in which automation is delivered advisors combined with an Automated Drilling Control Systems (ADCS).

The microstate representation of the drilling process is expressed as a tuple with 80 positions. Each position can have either 2 or 3 states, but the state can also be undefined, denoted by the value 0. The index of each dimension is described in the supplementary material of (Cayeux et al. 2024). Microstate representations are considered terminals of the grammar. However, usually only a pattern of the microstate is used to define the terminal. To specify which parts of the microstate pattern are defined, the following notation is used:

- A microstate representation is enclosed in squared brackets "[" and "]",
- A list of position numbers and state values is provided, separated by commas ",",
- The position number starts with the letter "p" followed by an integer corresponding to the position index, starting from 0.
- The value is defined after an equal sign "=" and can take the values 0, 1, 2, or 3.

Fig. 8 shows an example of a microstate representation.

```
[p4=2,p12=2,p5=1,p13=1]
```

Figure 8—Example of a microstate representation.

The signification of the example of Fig. 8 is the following:

- "$p4 = 2$" means that the flowrate at the top of string, Q_{tos}, shall be strictly positive, i.e., $Q_{tos} > \varepsilon Q_{TOS}$ where εQ_{TOS} is a minimum threshold value.
- "$p12 = 2$" means that the flowrate at the outlet, Q_{out}, shall be strictly positive, i.e., $Q_{out} > \varepsilon Q_{out}$.
- "$p5 = 1$" means that the variation of the flowrate at the top of string, characterized by a standard deviation, σQ_{tos}, shall be smaller than a threshold value σQ_{tos_0},
- "$p13 = 1$" means that the variation of the flowrate out, σQ_{out}, shall also be smaller than a threshold value, σQ_{out_0}.

This pattern corresponds to a condition in which the flow in the well is in steady-state conditions. Fig. 9 shows an example of production rules that use microstate patterns to recognize a pump startup.

```
pumpStartup: (fillPipe | accelerateFromNoFlow) (accelerateToNextFlowrate waitForSteadyStateFlow)+;
fillPipe: [p4=1,p8=1] [p4=2,p5=2,p8=1] [p4=2,p5=1,p8=1]? [p4=2,p5=2,p8=2];
accelerateFromNoFlow: [p4=1,p8=1] [p4=2,p5=2,p8=1]? [p4=2,p5=2,p8=2];
accelerateToNextFlowRate: [p4=2,p5=2,p8=2,p12=1]? [p4=2,p5=2,p8=2,p12=2,p13=2];
waitForSteadyStateFlow: ([p4=2,p5=1,p8=2,p12=2,p13=2] [p4=2,p5=1,p8=2,p12=2,p13=1])+;
```

Figure 9—Example of production rules using microstate patterns to recognize a pump startup.

The "pumpStartup" rule indicates that during a pump startup, there may optionally be a "fillPipe". Afterward, there must be at least one sequence of "accelerateToNextFlowrate" followed by "waitForSteadyStateFlow."

It should be noted that $p8 = 1$ means that the pressure at the top of string is marginal, i.e., $p_{tos} \leq p_{atm} + \varepsilon p_{tos}$ where εp_{tos} is a threshold value. On the other hand, $p8 = 2$ means that the pressure at the top string is sufficient to indicate that flow has started in the string, i.e., $p_{tos} > p_{atm} + \varepsilon p_{tos}$. The "fillPipe" rule stipulates that initially, there is no flow and no pressure at the top of the string ($p4 = 1, p8 = 1$). Next, the flow at the top of the string is accelerated, but there is still no pressure at the top of the string ($p4 = 1$, $p5 = 2$, $p8 = 1$). It is possible that the flow at the top of the string may become stable, but still without any significant pressure ($p4 = 2$, $p5 = 1$, $p8 = 1$). Finally, the flow at the top of the string changes when the pressure becomes substantial ($p4 = 1$, $p5 = 2$, $p8 = 2$).

The "accelerateToNextFlowRate" rule is broken down as follows:

- Optionally, the flow at the top of the string is accelerated, meaning it is strictly positive ($p4 = 2$) but unstable ($p5 = 2$), while the flow at the outlet has not yet started ($p12 = 1$). This may correspond to the beginning of breaking the gel, during which fluid compressibility and gel strength prevent the movement of fluid at the outlet.
- This is followed by a phase where the flow at the outlet becomes strictly positive ($p12 = 2$) but unstable ($p13 = 2$), while the flow at the top of the string continues to accelerate ($p4 = 2$, $p5 = 2$).

Note that the pattern [$p4 = 2$,$p5 = 1$,$p8 = 2$,$p12 = 2$,$p13 = 2$] means that the flow rate at the top of the string and at the outlet are strictly positive ($p4 = 2$,$p12 = 2$), the flow rate at the top of the string is stable ($p5 = 1$), the flow rate at the outlet is strictly positive ($p12 = 2$) but not yet stable ($p13 = 2$). Therefore, the rule "waitForSteadyStateFlow" triggers when, first, the flow at the top of the string is stable while the flow at the outlet is unstable, followed by a condition where both the flow at the top of the string and at the outlet become stable, while remaining strictly positive for both conditions. This cycle can repeat several times as flow oscillations may have been induced by the previous flow acceleration.

Using the example of fig. 11 in (Cayeux et al. 2024), it is possible to apply the above defined grammar rules to recognize a pump startup. Fig. 10 illustrates how the rules apply:

- First, an "accelerateFromNoFlow" is recognized through the sequence of microstate patterns [$p4 = 1, p8 = 1$], [$p4 = 2, p5 = 2, p8 = 1$], [$p4 = 2, p5 = 2, p8 = 2$].
- Then, a "waitForSteadyStateFlow" is parsed because of the sequence of microstate patterns [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 2$], [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 1$], [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 2$], [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 1$].
- Next, the rule "accelerateToNextFlowRate" is triggered because of the microstate pattern [$p4 = 2, p5 = 2, p8 = 2, p12 = 2$].
- Finally, another "waitForSteadyStateFlow" is recognized because of the sequence of microstate patterns [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 2$], [$p4 = 2, p5 = 1, p8 = 2, p12 = 2, p13 = 1$].

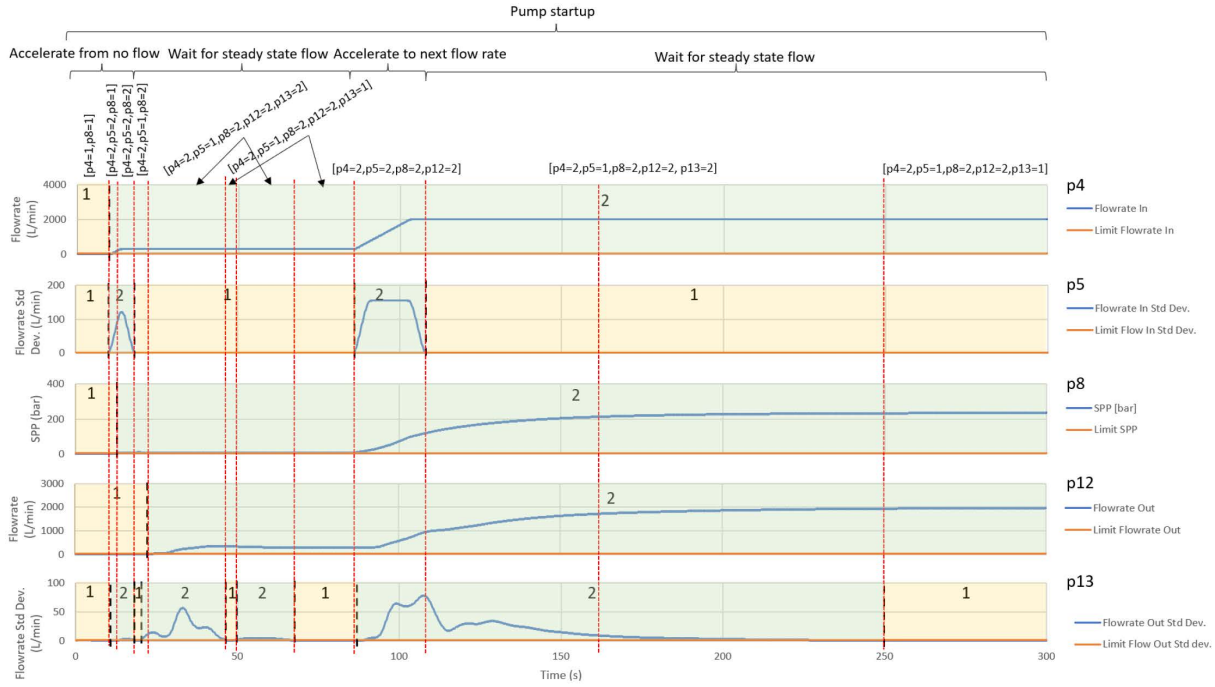

Figure 10—The sequence of microstate patterns matches the grammar rules for a "pumpStartup" operation.

```
startSurfaceRotation:  'SEQ' '[' accelerateFromNoRotation ',' (accelerateFromRotation ',')* constantRotation ']';
accelerateFromNoRotation: [p2=1] [p2=2,p3=2];
accelerateFromRotation: [p2=2,p3=1] [p2=2,p3=2];
constantRotation: [p2=2,p3=1];
```

Figure 11—Production rules for identifying a startup of surface rotation.

The sequence "accelerateFromNoFlow", "waitForSteadyStateFlow", "accelerateToNextFlowRate", "waitForSteadyStateFlow" is compatible with the rule "pumpStartup" and therefore the operation can be labelled accordingly.

Examples

In the previous section, the process of recognizing a pump startup using formal grammar rules was discussed. Here, additional examples will be provided. These examples incrementally build from recognizing standard tasks, which can subsequently be used to interpret actions and ultimately label phases.

In a typical drilling context, after establishing circulation, the drill-string rotation is initiated—referred to as the "startSurfaceRotation" task. The production rules for identifying this task are provided in Fig. 11.

The microstate $p2$ is defined such that $p2 = 1$ if $|\dot{\theta}_{tos}| \leq \varepsilon_{\dot{\theta}_{tos}}$, and $p2 = 2$ otherwise, where $\dot{\theta}_{tos}$ represents the angular rotational speed of the top of the string and $\varepsilon_{\dot{\theta}_{tos}}$ is a threshold value. The microstate $p3$ is based on the standard deviation of this angular rotational speed over a time window ($\sigma_{\dot{\theta}_{tos}}$), i.e., $p3 = 1$ if $\sigma_{\dot{\theta}_{tos}} > \sigma_{\dot{\theta}_{tos_0}}$ and $p3 = 2$ otherwise where $\sigma_{\dot{\theta}_{tos_0}}$ is a set threshold.

The production rules for the "startSurfaceRotation" task essentially specify that the task initiates from a standstill and accelerates to a constant rotational speed, either in one continuous increase or through stepwise increments.

An example is shown on Fig. 12 where the top-of-string rotation occurs in two steps.

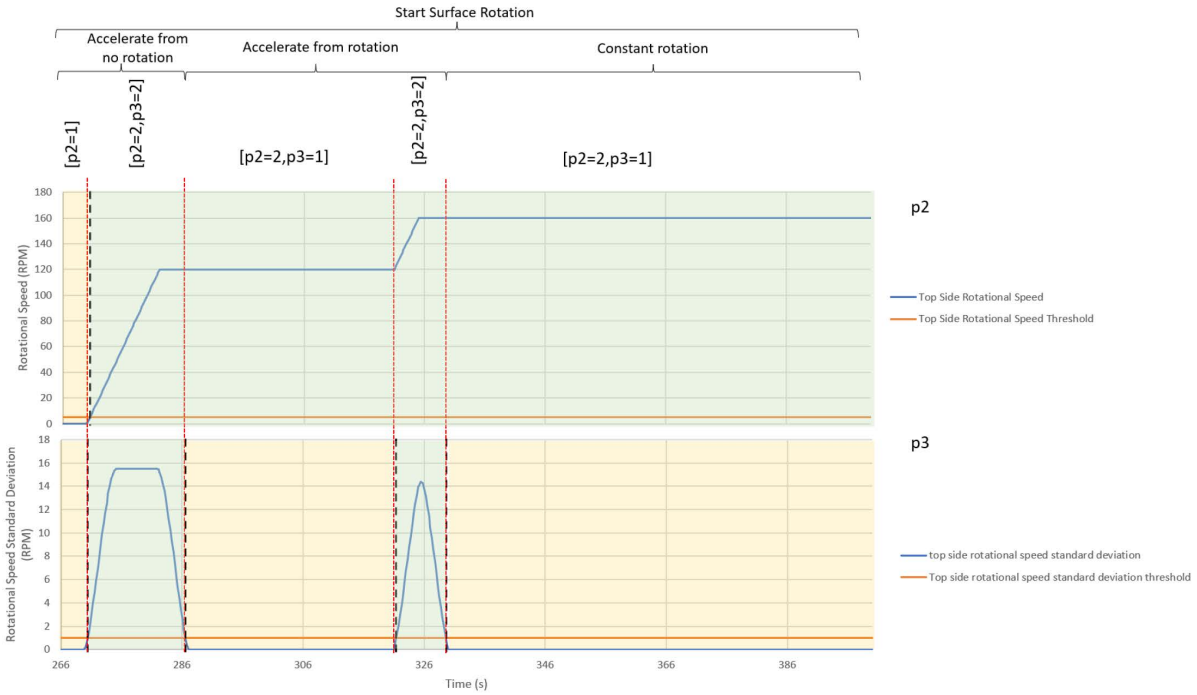

Figure 12—The top of string rotation is initiated from a standstill to a constant rotational speed in two steps.

The "drillFormation" task can be executed by rotating the bit and applying force to the formation, or by using an under-reamer or hole opener to do the same. The microstate $p22$ defines the axial movement of the bottom of the string, where $p22 = 1$ when $|v_{bos}| \leq \varepsilon v_{bos}$, $p22 = 1$ when $v_{bos} > \varepsilon v_{bos}$ and $p22 = 3$ when $v_{bos} < -\varepsilon v_{bos}$, with v_{bos} representing the bottom-of-string axial velocity and εv_{bos} as an axial velocity threshold.

The microstate $p18$ is related to the bottom-of-string rotational velocity, defined as $p18 = 1$ if $|\dot{\theta}_{bos}| \leq \varepsilon \dot{\theta}_{bos}$, otherwise $p18 = 2$. The bottom-of-string-on-bottom microstate, $p15$, is defined by $p15 = 1$ if $|F_{bos\_rock}| \leq \varepsilon F_{s\_rock}$ and $p15 = 2$ otherwise, where F_{bos_rock} is the force between the bottom of the string and the formation, and εF_{s_rock} is a threshold value. Similarly, the microstate $p17$ applies the same criteria but for the force between an under-reamer or hole opener and the formation. Thus, $p17 = 1$ if $|F_{ur\_rock}| \leq \varepsilon F_{S\_rock}$ and $p17 = 2$ otherwise, where F_{ur_rock} denotes the force between the under-reamer or hole opener and the formation.

The grammar rules for the "drillFormation" task are defined in Fig. 13.

```
drillFormation: 'SEQ' '[' tagBottom ',' drill ',' pickOffBottom ']':
tagBottom: [p22=1,p18=2,p15=1] ([p22=2,p18=2,p15=1] [p22=1,p18=2,p15=1]?);
drill: ([p22=2,p18=2,p15=2] | [p22=1,p18=2,p15=2] | [p22=2,p18=2,p17=2] | [p22=1,p18=2,p17=2])+;
pickOffBottom: ([p22=3,p18=2,p15=2] | [p22=3,p18=2,p17=2]) [p22=3,p18=2,p15=1,p17=1];
```

Figure 13—Grammar rules for the drill formation task.

Fig. 14 provides an example of the "drillFormation" task in a configuration that does not include a hole opener or under-reamer. It is notable that during the "tagBottom" and "drill" tasks, $p22$ transitions from state 2 to 1. The key point is that $p22$ does not enter state 3, which is characteristic of the "pickOffBottom" task.

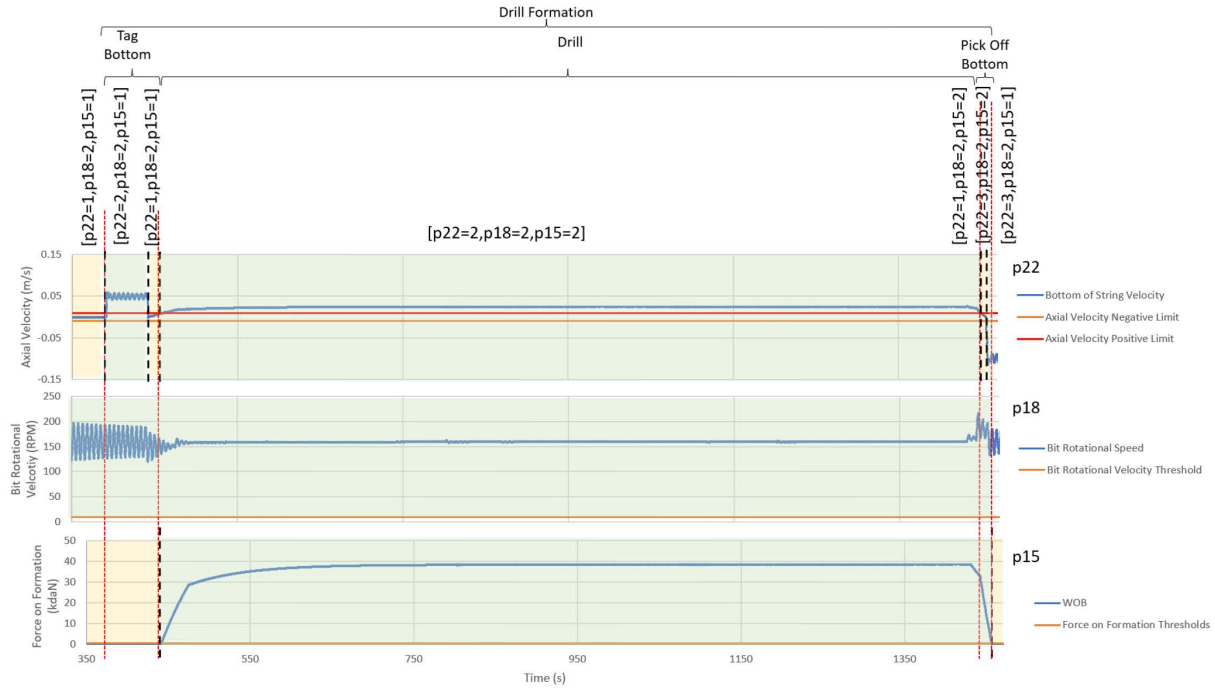

Figure 14—Example of a "drillFormation" task during which the formation is tagged, then drilled before the drill-string is picked off bottom.

After drilling a formation, to displace cuttings above the bottom hole assembly (BHA) before stopping circulation, it is common practice to reciprocate the drill-string. Reciprocation involves multiple sequences of reaming up and down the drill-string with fluid flow. During this process, there must be flow at the top of the string, and the BHA must be off bottom. Reaming up refers to upward axial movement with drill-string rotation, while reaming down corresponds to a downward axial movement. The grammar rule defining the "reciprocation" task is shown on Fig. 15.

```
reciprocation: ([p0=2,p2=2,p4=2,p15=1] [p0=1,p2=2,p4=2,p15=1]? [p0=3,p2=2,p4=2,p15=1])+;
```

Figure 15—Grammar rule defining a reciprocation task.

The grammar rule utilizes the microstate, which is determined based on the top-of-string axial velocity, v_{tos}. If $|v_{tos}| \leq \varepsilon_{v_{tos}}$ then $p0 = 1$, if $v_{tos} > \varepsilon_{v_{tos}}$ then $p0 = 2$ else $p0 = 3$. Here, v_{tos} represents the axial velocity at the top of the string, and $\varepsilon_{v_{tos}}$ is a threshold value.

Fig. 16 illustrates an example of a "reciprocation" task. The "ream-up" task begins once the bit is off bottom, with rotation, upward axial movement, and circulation. During this task, there may be periods with zero axial movement. The grammar rule defines this state as optional; however, here, it is used to adjust flow rate and rotational speed before initiating the "ream-down" task. The "ream-down" task occurs when the axial velocity is directed downward while rotation and circulation continue. The termination of top-of-string rotation signals the end of the "reciprocation" task, as this condition does not align with the grammar rule.

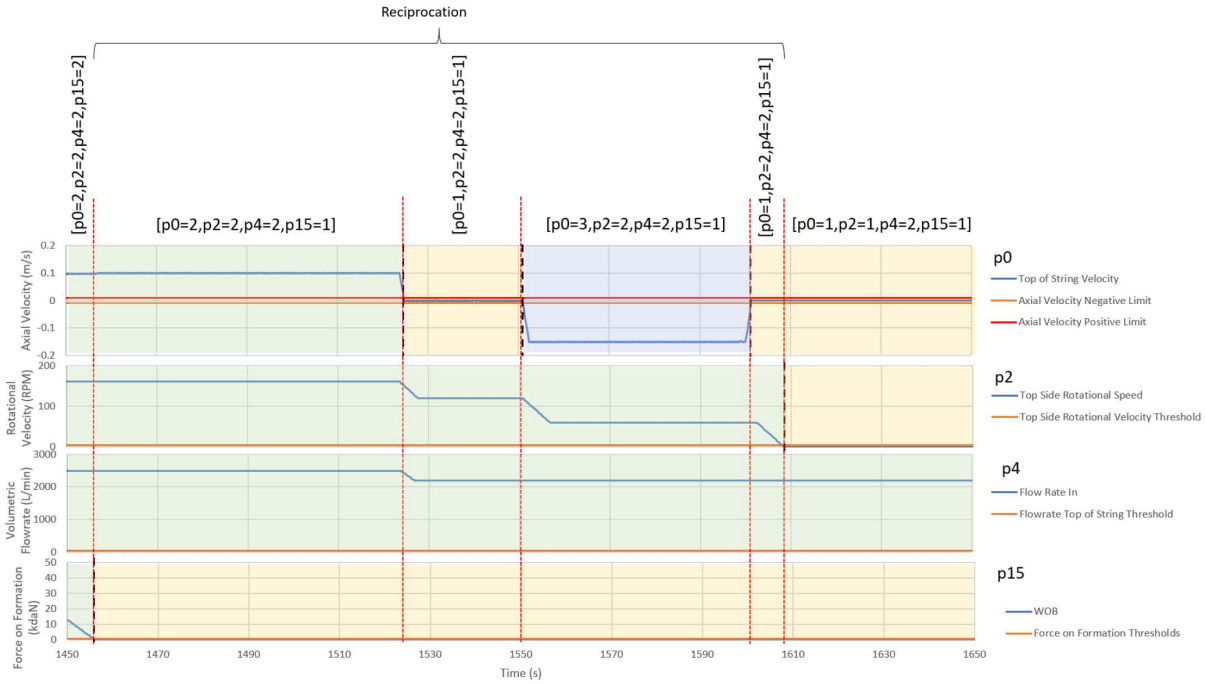

Figure 16—Example of a "reciprocation" task that starts with a ream-up and finishes with a ream-down.

Before making a connection, the top-of-string rotation must be stopped. However, in deviated wells, after halting surface rotation, residual torque may remain due to static friction along the drill-string. The drilling control system typically includes a mechanism to zero out this torque at the top of the string. Only then can the top-drive be disconnected from the top of the string. The grammar rule describing the "stopRotation" task is shown on Fig. 17:

```
stopRotation: [p2=2,p10=2] [p2=1,p10=2]? [p2=1,p10=1];
```

Figure 17—Grammar rule corresponding to stopping the surface rotation.

The rule employs the microstate $p10$, which is calculated using the top-of-string torque. Specifically, if $\tau_{tos} \leq \varepsilon\tau_{tos}$ then $p10 = 1$ else $p10 = 2$, where τ_{tos} is the torque at the top of the string and $\varepsilon\tau_{tos}$ is a threshold value. Fig. 18 illustrates a "stopRotation" task. Starting from conditions with positive surface rotation and torque, the top-of-string rotation is first stopped, followed by the zeroing of the residual top-of-string torque.

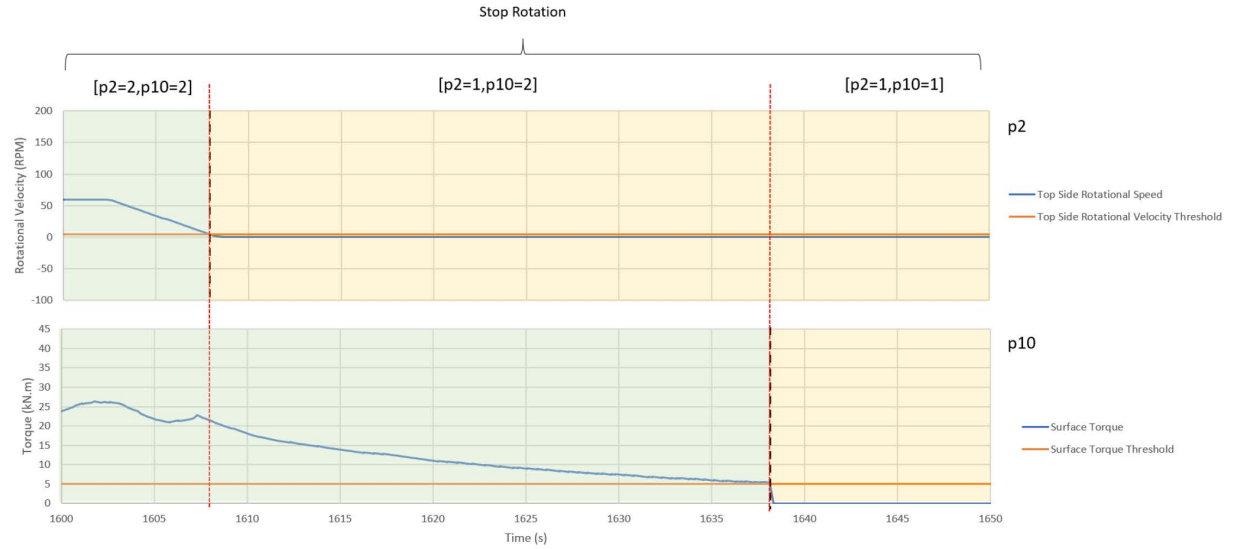

Figure 18—Example of a "stopRotation" task during which the rotational speed at the top of string is reduced to zero and then the remanent top of string torque is zeroed.

It is also necessary to stop circulation before making a connection. Initially, the flow rate at the top of the string is reduced to zero. However, due to the compressibility of the drilling fluid and the inertia of the fluid mass in motion, it takes time for the pressure at the top of the string to reach a sufficiently low value, allowing the top-drive or kelly to be disconnected. Fig. 19 shows the grammar rule to interpret a "stopCirculation" task.

```
stopCirculation: [p4=2,p8=2] [p4=1,p8=2] [p4=1,p8=1];
```

Figure 19—Grammar rule to recognize a stop circulation task.

Fig. 20 illustrates the sequence of microstates corresponding to a "stopCirculation" task.

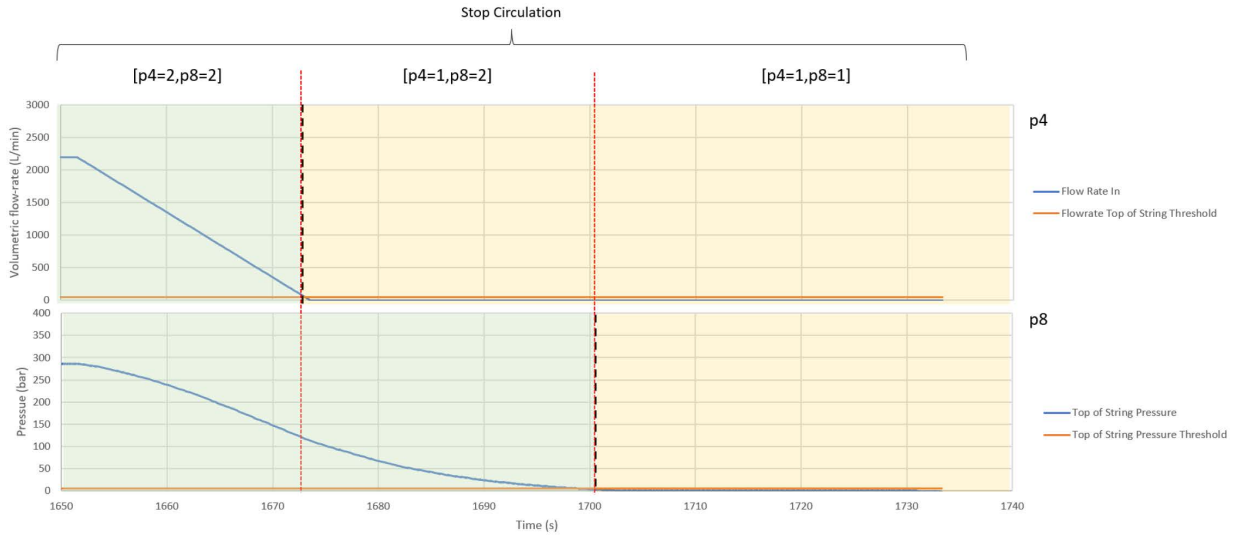

Figure 20—Example of a "stopCirculation" task during which the flowrate at the top of string is stopped. Yet it takes some time before the pressure at the top of string gets negligeable.

Having described some of the typical tasks involved in a "drill-a-stand" action, we can now define the grammar rules for this action (see Fig. 21).

```
drillAStand: 'SEQ' '[' startDrillProcedure ',' drillProcedure ',' endDrillProcedure ']';
startDrillProcedure: 'SEQ' '[' pumpStartup ',' (rotationStartup | orientate) ']' ;
drillProcedure: ('SEQ' '[' drillFormation ',' intermediateDrillProcedure? (',' drillFormation ',' intermediateDrillProcedure?)*']';
endDrillProcedure: 'SEQ' '[' (stopRotation ',')? stopPump ']' ;
intermediateDrillProcedure: slideProcedure | rotateProcedure| holeManagementProcedure;
slideProcedure: 'SEQ' '[' pickOffBottom ',' (stopRotation ',')? orientate ']';
rotateProcedure: 'SEQ' '[' pickOffBottom ',' rotationStartup ']';
holeManagementProcedure: 'SEQ' '[' pickOffBottom ',' 'SEQ' '[' (reciprocation (',' frictionTest)?)+ ']' ']';
```

Figure 21—Grammar rules to interpret a drill-a-stand action.

Note that, for conciseness, the "pickOffBottom" and "orientate" rules are not described in this paper. An example of a drill-a-stand procedure and its interpretation using the grammar rules defined above is shown in Fig. 22. In this example, the operation begins with a "pumpStartup" followed by a "rotationStartup". These tasks are categorized as part of a "startDrillProcedure" task. Next, the sequence of tasks "tagBottom", "drill", "pickOffBottom" and "reciprocate" are parsed and labeled as a "drillProcedure" task. It should be noted that "tagBottom", "drill" and "pickOffBottom" are sub-tasks of the "drillFormation" task and that "reciprocate" is also classified as an "intermediateDrillProcedure" task. Finally, the tasks "stopRotation" and "stopPump" are identified as part of the "endDrillProcedure" task.

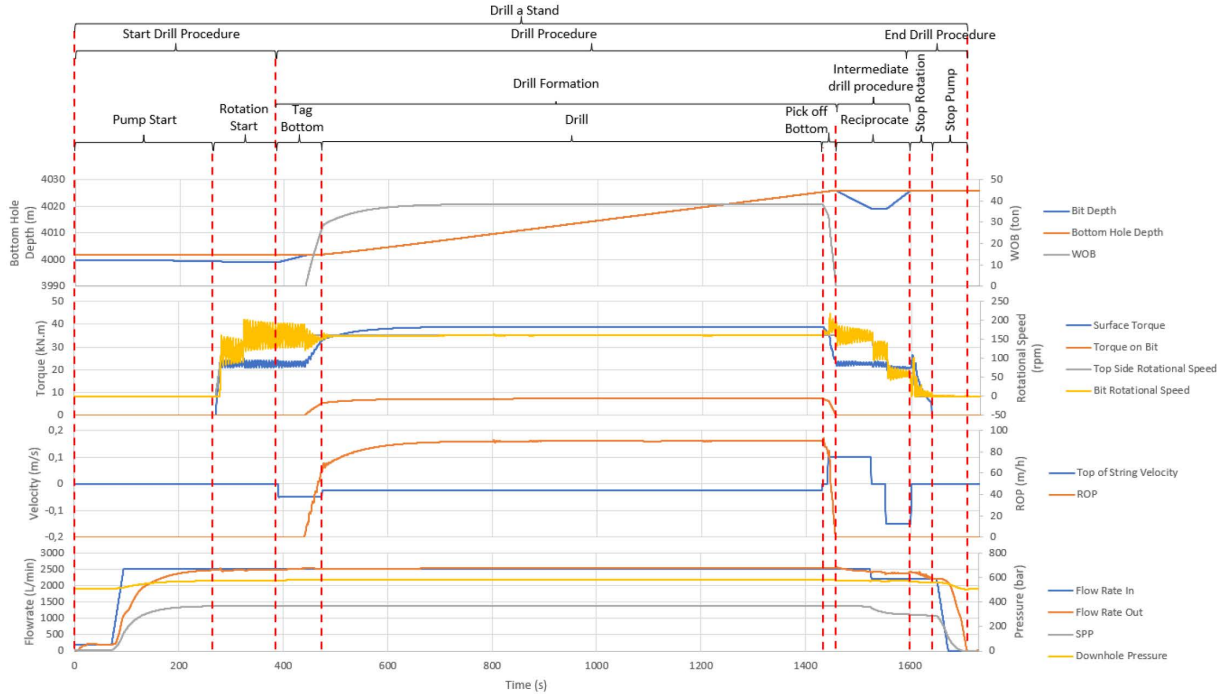

Figure 22—Example of a drill-a-stand procedure that is decomposed in "start drill procedure", "drill procedure" and "end drill procedure", which in turn are decomposed in "pump start", "rotation start", "tag bottom", "drill", "reciprocation", "stop rotation", "stop pump".

Similarly, phase descriptions can reference the grammar rules for actions and be structured hierarchically using production rules.

Fig. 23 exemplifies this principle, illustrating a "drillASection" phase from an actual North Sea drilling operation, broken down into more detailed phases:

- Run in hole on elevator,
- Displace and condition mud,
- Drill section,
- Circulate clean,

- Back-ream,
- Pull-out on elevator.

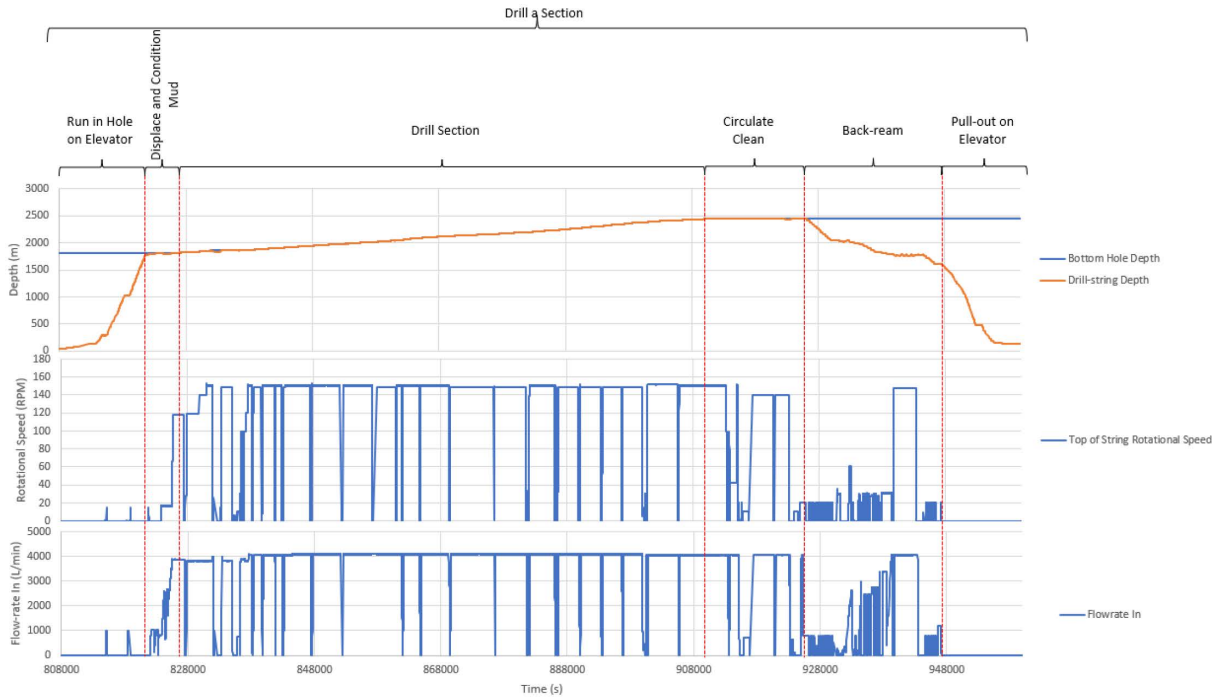

Figure 23—Example of a "drillASection" phase that is composed of the following actions: "runInHoleOnElevator", "displaceAndConditionMud", "drillSection", "circulateClean", "backReam" and "pullOutOnElevator".

Discussion

A method for automatically classifying drilling activities by recognizing sequences of micro-state patterns has been described. This method is based on lexical and syntactic grammar rules expressed using an extended Backus-Naur Form syntax. The labeling of drilling operations can occur at different levels, such as a "flow acceleration to the next flow rate," which is part of a "pump startup" task, itself belonging to a "drill a stand" action, which is a component of a "drill a section" phase. Additionally, equipment must be defined as part of the context, as the nature of certain actions cannot be distinguished without considering the equipment being used—such as differentiating drilling from coring.

Having a BNF grammar enables the implementation of programs that can label drilling operations at different levels. The advantage of using a formal grammar is that if various companies create drilling operation parsers, the labeling will remain consistent across implementations. This is crucial because it allows multiple vendors to use the labeling as an interoperability mechanism. Today's codes are not formally defined, leaving much room for interpretation by both humans and computer systems. Similarly, auto-classification of drilling activities using different algorithms, such as pattern matching or artificial neural networks, may work effectively, but they do not provide a transparent labeling of drilling operations that can be uniformly interpreted across different systems. A standardized labeling of drilling activities is essential for shared understanding among systems that need to interoperate. This issue is not limited to systems from different vendors but also arises within systems from the same organization, especially if they are developed by different software teams.

The described solution relies heavily on the concept of micro-states. Micro-states are formally defined, ensuring that different implementations of a drilling micro-state interpretation engine, as long as they adhere to the micro-state definitions, will produce identical results. However, micro-states are calculated using

inputs from one or more digital twins, and these digital twins introduce some level of uncertainty. The micro-state interpretation engine accounts for this uncertainty when calculating state values. Additionally, the engine may perform calibration and sensor fusion for signals generated by multiple digital twins. The quality of the interpreted micro-states depends on the accuracy of the estimations made by the digital twins. If these estimations are of poor quality, the resulting micro-state interpretations will also be of low quality. Unreliable micro-states can compromise the accuracy of the labeling done using the formal grammar for drilling operations. As in any processing chain, a low-quality element affects the quality of the subsequent steps. Therefore, it is essential to have robust processes in place to ensure the quality of the signals provided by the digital twins, as these form the initial inputs in the processing chain that leads to the labelling of drilling operations.

The reliance on digital twins to estimate micro-states also implies that drilling operations heavily dependent on manual work or lacking sufficient instrumentation cannot be processed using the proposed method.

The concept of a micro-state stems from the idea that the drilling process can be characterized by the state of the boundary conditions of the partial differential equations that govern the drilling process. However, this concept is not well-suited to cover activities that are not directly related to the drilling process, such as moving the rig into position, installing a blowout preventer (BOP), or deploying a marine riser. As a result, the proposed solution does not fully encompass the labeling of all drilling activities, because the notion of micro-states does not account for operations unrelated to well construction. Unfortunately, this means that the proposed method cannot entirely replace all the codes that are defined manually by personnel at the rig site.

Additionally, the definitive labeling of an activity in a drilling operation may be delayed until a sufficient sequence of micro-state patterns has been observed. Labeling drilling operations is not as instantaneous as micro-state interpretation. The latency of this labeling can vary depending on the specific drilling operation. In a real-time context, having an operation like labeling that does not respond in constant time is a significant drawback. However, it seems unlikely that the issue of latency can be fully resolved unless the rig action plan is used as an additional input for labeling. In this case, it may be possible to assign a likely label based on the expected next activity outlined in the rig action plan, combined with the start of a sequence of micro-states. In other words, it is easier to predict a sentence from just a few letters or words if you already know which sentences are plausible.

Not all use cases or situations require evaluating the entire micro-state space. If the context of a process or activity is known or can be safely assumed, the inference of an activity can rely on fewer micro-states, potentially with greater precision. For example, if it is known that a drilling BHA has been deployed into the well, all non-drilling-related activities can be excluded. Further simplification of the inference process can be achieved if applications are focused solely on "steady-state" information, rather than on timed state transitions, such as those described for "starting up pumps" or "starting/stopping" rotation. Additionally, immediate real -time signals from reliable sources, such as an "In-Slips" signal from instrumented slips, can further reduce the complexity of the activity inference component.

Conclusion

The automatic classification of drilling activities is desirable to eliminate the subjectivity involved when humans choose activity codes. Additionally, there is a lack of consensus regarding which activity codes should be used, their definitions, and the methods for automatically recognizing them. A method based on formal definitions using an extended Backus Naur Form following the ANTLR syntax has been described. It relies on the existence of micro-states that can be automatically interpreted using inputs from digital twins. The micro-states themselves are formally defined, ensuring reproducible implementations. However, this method cannot handle highly manual operations, or any activities not directly related to the drilling process.

Despite this limitation, the ANTLR-based description of drilling activity classification presents new opportunities for interoperability at the rig site:

- Unambiguous definitions of drilling activity labels allow applications to interact, as they share the same understanding of the current activity.
- Automatic labeling of drilling activities acts as an indirect source of synchronization between applications that need to interoperate.
- Extending and refining the formal grammar for drilling activity classification is straightforward, and adjusting the software that uses this grammar to recognize drilling activities is simplified.

Acknowledgement

The authors would like to thank the other team members of the workstreams "Drilling Process Protection" and "Contextual Data" of the D-WIS sub-committee of SPE-DSATS for their continuous contributions to the development of interoperability solutions for drilling operations.

Abbreviations

ADCS	Automated Drilling Control System
ANTLR	Another Tool for Language Recognition
BHA	Bottom Hole Assembly
BOP	Blow-Out Preventer
BNF	Backus Naur Form
CAODC	Canadian Association of Oilwell Drilling Contractors
CPU	Computer Processing Unit
DDR	Daily Drilling Report
DDR PLUS	Daily Drilling Report print and electronic system
DSATS	Drilling System Automation Technical Section
D-WIS	Drilling and Wells Interoperability Standards
EOS	End Of Stream
eRAP	electronic Rig Action Plan
IADC	International Association of Drilling Contractors
ISA	International Society of Automation
KPI	Key Performance Indicators
NIL	Empty token
RAP	Rig Action Plan
SPE	Society of Petroleum Engineer

Nomenclature

F_{bos_rock}	force applied by the formation on the bottom of string [MLT^{-2}] (N)
p_{atm}	atmospheric pressure [$ML^{-1}T^{-2}$] (Pa)
p_{tos}	pressure at the top of string [$ML^{-1}T^{-2}$] (Pa)
Q_{out}	flow-rate of cuttings at the outlet of the annulus [L^3T^{-1}] (m^3/s)
Q_{tos}	flow-rate at the top of string [L^3T^{-1}] (m^3/s)
v_{bos}	bottom of string axial velocity [LT^{-1}] (m/s)
v_{tos}	top of string velocity counted positively downward [LT^{-1}] (m/s)

Greek letters:

$\varepsilon_{F_{S\_rock}}$ tolerance margin for no reaction force between the string and the formation [MLT^{-2}] (N)
$\varepsilon_{p_{tos}}$ tolerance margin for unpressurized state at the top of string [$ML^{-1}T^{-2}$](Pa)
$\varepsilon_{Q_{out}}$ tolerance margin for zero flow at the well outlet [L^3T^{-1}](m^3/s)
$\varepsilon_{Q_{tos}}$ tolerance margin for zero flow at the top of string [L^3T^{-1}](m^3/s)
$\varepsilon_{v_{bos}}$ tolerance margin for considering that the bottom of string is at rest [LT^{-1}](m/s)
$\varepsilon_{v_{tos}}$ tolerance margin for a zero top of string velocity criterion [LT^{-1}](m/s)
$\varepsilon_{\dot{\theta}_{bos}}$ tolerance margin for zero rotational speed at the bottom of string [T^{-1}](rad/s)
$\varepsilon_{\dot{\theta}_{tos}}$ tolerance margin for zero rotational speed at the top of string [T^{-1}](rad/s)
$\dot{\theta}_{bos}$ angular velocity of the bottom of string [T^{-1}] (rad/s)
$\dot{\theta}_{tos}$ angular velocity of the top of string [T^{-1}] (rad/s)
$\sigma_{Q_{out}}$ standard deviation of the flow out of the annulus [L^3T^{-1}] (m^3/s)
$\sigma_{Q_{out_0}}$ threshold value [L^3T^{-1}] (m^3/s)
$\sigma_{Q_{tos}}$ standard deviation of the flowrate at the top of string [L^3T^{-1}] (m^3/s)
$\sigma_{Q_{tos_0}}$ threshold value [L^3T^{-1}] (m^3/s)
$\sigma_{\dot{\theta}_{tos}}$ standard deviation of the rotational velocity at the top of string [T^{-1}] (rad/s)
$\sigma_{\dot{\theta}_{tos_0}}$ threshold value [T^{-1}] (rd/s)

References

Adams, Jeanne C., Brainerd, Walter S., Martin, Jeanne T. et al 1992. Fortran 90 Handbook, Vol. **32**: McGraw-Hill New York.

Aho, Alfred, Sethi, Ravi, and Ullman, Jeffrey. 1985. Compilers, Principles, Techniques and Tools: Computer Science, Addison-Wesley Publishing Company.

Backus, John. 1978. The history of Fortran I, II, and III. *ACM Sigplan Notices* **13** (8): 165–180.

Backus, John W., Bauer, Friedrich L., Green, Julien et al 1960. Report on the algorithmic language ALGOL 60. *Communications of the ACM* **3** (5): 299–311.

Baumgartner, Theresa, Ridgway, Marcus, Bruce, Cameron et al 2022. Redefining Operations Reporting Codes to Support Digitalization of Well Operations. *Proc., IADC/SPE International Drilling Conference and Exhibition*. https://doi.org/10.2118/208711-MS.

Cayeux, E., Macpherson, J., Pirovolou, D. et al 2024. A General Framework to Describe Drilling Process States. *SPE Journal*: 1–18. https://doi.org/10.2118/212537-PA.

Farmanbar, Pourya, Revheim, Olav, Uberg, Anne Siw et al 2020. Digitalization of Detailed Drilling Operation Plans and Verification of Automatic Progress Tracking with an Online Drilling Simulator Environment. *Proc., IADC/SPE International Drilling Conference and Exhibition*. https://doi.org/10.2118/199666-MS.

Ginsburg, Seymour, Greibach, Sheila A., and Harrison, Michael A. 1967. Stack automata and compiling. *J ACM* **14** (1): 172–201. https://doi.org/10.1145/321371.321385.

ISA-88 Batch Process Control, 1995: International Society of Automation.

Iversen, F. P., Thorogood, J. L., Macpherson, J. D. et al 2016. Business Models and KPIs as Drivers for Drilling Automation. *Proc., SPE Intelligent Energy International Conference and Exhibition*.

Jantzen, R. E., Foreman, R. D., Keltner, L. et al 1989. Wellsite Information Transfer Specification (WITS) for Digital Rig-Site Data. *SPE Drilling Engineering* **4** (04): 291–299. https://doi.org/10.2118/16141-PA.

Krikor, Ara, Khambete, Shreepad Purushottam, Bimastianto, Paulinus Abhyudaya et al 2022. Machine Learning Delivers Automated Feedback on Real Time Key Performance Indicators During Drilling Operations. *Proc., ADIPEC*. https://doi.org/10.2118/211753-MS.

Kuilenburg, R. Van, Isbell, M., Behounek, M. et al 2024. Interoperability for Drilling Process Automation. *Proc., IADC/SPE International Drilling Conference and Exhibition*. https://doi.org/10.2118/217748-MS.

Manthorpe, William HJ. 1996. The emerging joint system of systems: A systems engineering challenge and opportunity for APL. *Johns Hopkins APL Technical Digest* **17** (3): 305–313.

McCracken, Daniel D. and Reilly, Edwin D. 2003. Backus-Naur form (BNF). In *Encyclopedia of Computer Science*, 129–131. John Wiley and Sons Ltd.

Parr, T. J. and Quong, R. W. 1995. ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience* **25** (7): 789–810. https://doi.org/10.1002/spe.4380250705.

Reilly, Edwin D. 2003. Milestones in computer science and information technology: Greenwood Publishing Group Inc.

Scott, Mona L. and Scott, Mona L. 1998. Dewey decimal classification. *Libraries Unlimited*.

Shackleton, David, van Kuilenburg, Robert, Moralez, Nathan et al 2020. New Version of IADC Daily Drilling Report Significantly Increases Granularity, Provides Opportunity to Collaborate Using a Common Digital Format. *Proc., IADC/SPE International Drilling Conference and Exhibition*. https://doi.org/10.2118/199664-MS.

Spoerker, Hermann Friedrich. 2013. Digital Modeling of the Drilling Process and Automated Operations Recognition as Basis for Optimizing Drilling Economics.

Yin, Qishuai, Yang, Jin, Hou, Xinxin et al 2020. Drilling performance improvement in offshore batch wells based on rig state classification using machine learning. *Journal of Petroleum Science and Engineering* **192**: 107306. https://www.sciencedirect.com/science/article/pii/S0920410520303831.