MonPoly/MFOTL

Seminar Advanced Software Engineering FS22
Simon Furrer

Seminars topic

- How does specification and checking work with MonPoly/MFOTL?
- For an added point: which formulas are monitorable and which are not?

TOC

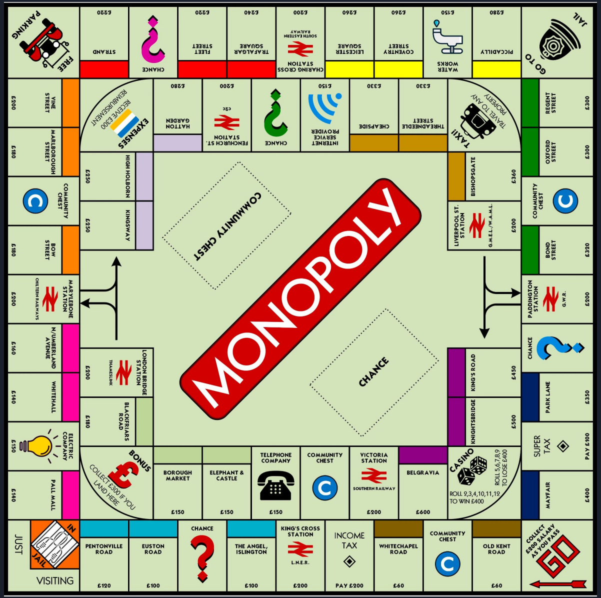
Motivation

- There is a Java application, which reads from smart meters, communicates with car chargers/heat pumps/batteries/boilers/washing machines/… and optimizes the own consumption of buildings having photovoltaics installed.
- Automatically analyze log files of that application for known problems -> trace checking
- Link detected problems with known fixes for those problems
- This reduces costs and increases the quality of doing operations

Motivation

MonPoly

- It's NOT about
- It's about …



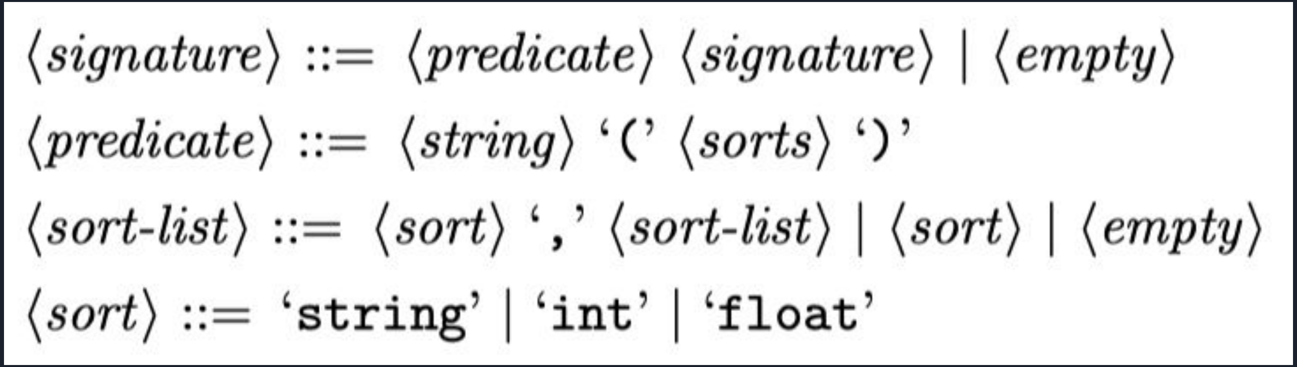
MonPoly

- MonPoly is a prototype monitoring tool
- Developed as part of an academic project at ETH Zurich
- Check the compliance of log files with respect to policies that are specified by formulas in MFOTL

MonPoly: specification

- Signatures (.sig)
- Policy Specification Language (.mfotl)
- Log entries (.log)

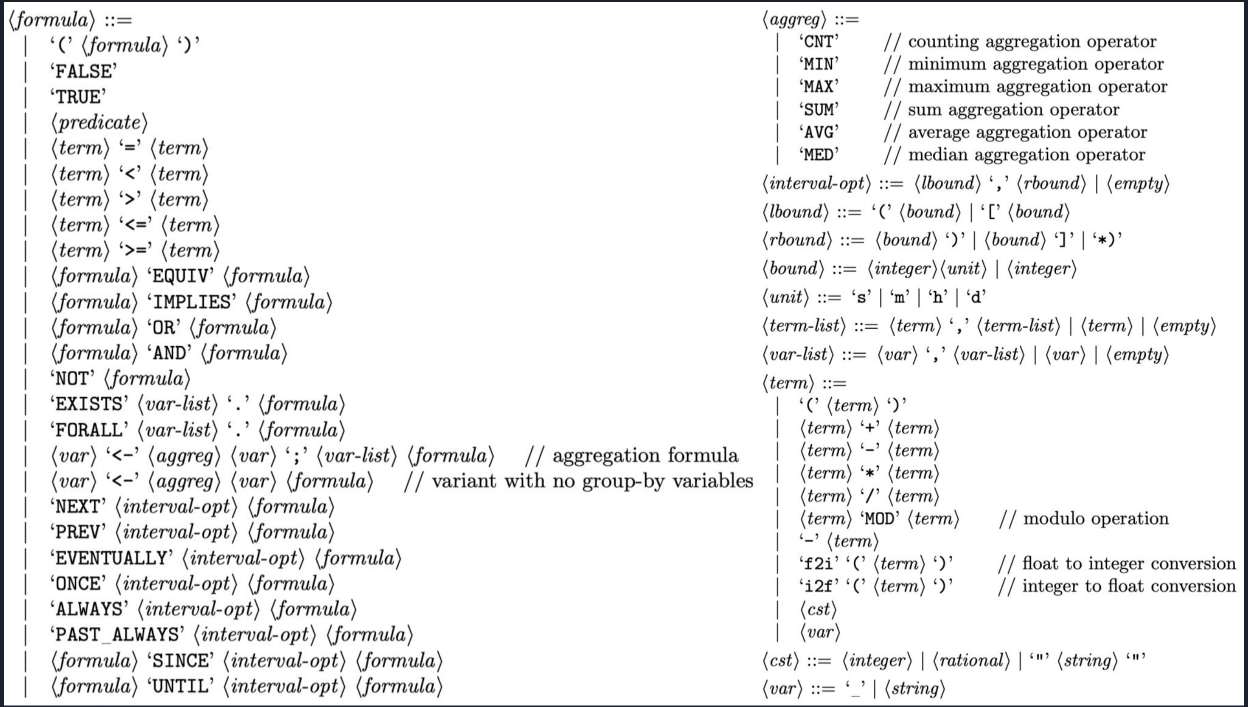
MonPoly: signatures



$$\begin{aligned}
\langle \textit{signature} \rangle &::= \langle \textit{predicate} \rangle\ \langle \textit{signature} \rangle \mid \langle \textit{empty} \rangle \\
\langle \textit{predicate} \rangle &::= \langle \textit{string} \rangle\ \text{‘(’}\ \langle \textit{sorts} \rangle\ \text{‘)’} \\
\langle \textit{sort-list} \rangle &::= \langle \textit{sort} \rangle\ \text{‘,’}\ \langle \textit{sort-list} \rangle \mid \langle \textit{sort} \rangle \mid \langle \textit{empty} \rangle \\
\langle \textit{sort} \rangle &::= \text{‘}\texttt{string}\text{’} \mid \text{‘}\texttt{int}\text{’} \mid \text{‘}\texttt{float}\text{’}
\end{aligned}$$

- Example: loglevel(a:string)

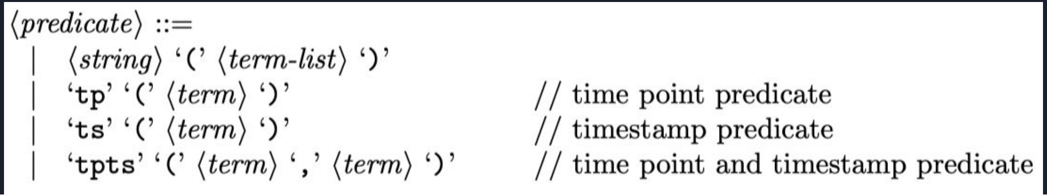
MonPoly: policy specification language



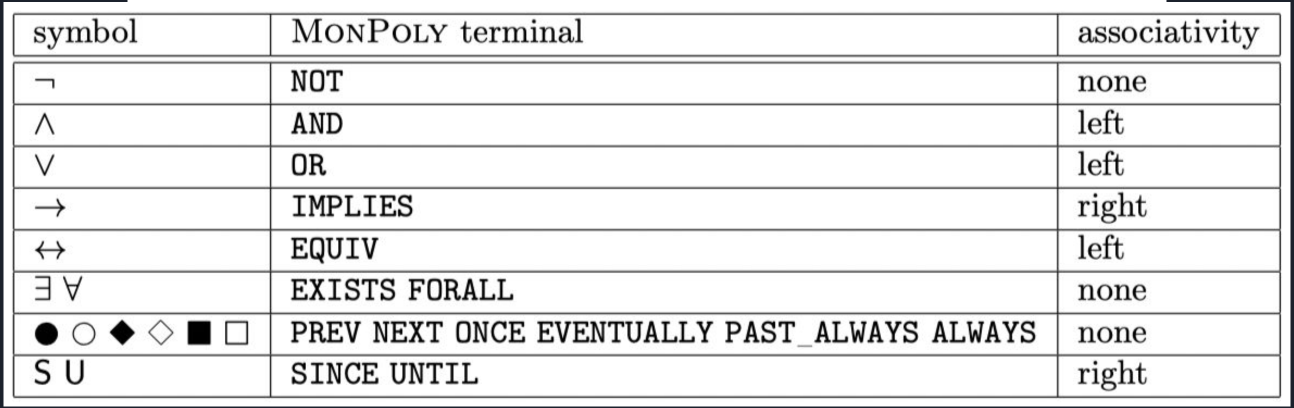
```
⟨formula⟩ ::=
  | ‘(’ ⟨formula⟩ ‘)’
  | ‘FALSE’
  | ‘TRUE’
  | ⟨predicate⟩
  | ⟨term⟩ ‘=’ ⟨term⟩
  | ⟨term⟩ ‘<’ ⟨term⟩
  | ⟨term⟩ ‘>’ ⟨term⟩
  | ⟨term⟩ ‘<=’ ⟨term⟩
  | ⟨term⟩ ‘>=’ ⟨term⟩
  | ⟨formula⟩ ‘EQUIV’ ⟨formula⟩
  | ⟨formula⟩ ‘IMPLIES’ ⟨formula⟩
  | ⟨formula⟩ ‘OR’ ⟨formula⟩
  | ⟨formula⟩ ‘AND’ ⟨formula⟩
  | ‘NOT’ ⟨formula⟩
  | ‘EXISTS’ ⟨var-list⟩ ‘.’ ⟨formula⟩
  | ‘FORALL’ ⟨var-list⟩ ‘.’ ⟨formula⟩
  | ⟨var⟩ ‘<-’ ⟨aggreg⟩ ⟨var⟩ ‘;’ ⟨var-list⟩ ⟨formula⟩   // aggregation formula
  | ⟨var⟩ ‘<-’ ⟨aggreg⟩ ⟨var⟩ ⟨formula⟩   // variant with no group-by variables
  | ‘NEXT’ ⟨interval-opt⟩ ⟨formula⟩
  | ‘PREV’ ⟨interval-opt⟩ ⟨formula⟩
  | ‘EVENTUALLY’ ⟨interval-opt⟩ ⟨formula⟩
  | ‘ONCE’ ⟨interval-opt⟩ ⟨formula⟩
  | ‘ALWAYS’ ⟨interval-opt⟩ ⟨formula⟩
  | ‘PAST_ALWAYS’ ⟨interval-opt⟩ ⟨formula⟩
  | ⟨formula⟩ ‘SINCE’ ⟨interval-opt⟩ ⟨formula⟩
  | ⟨formula⟩ ‘UNTIL’ ⟨interval-opt⟩ ⟨formula⟩
```

```
⟨aggreg⟩ ::=
  | ‘CNT’    // counting aggregation operator
  | ‘MIN’    // minimum aggregation operator
  | ‘MAX’    // maximum aggregation operator
  | ‘SUM’    // sum aggregation operator
  | ‘AVG’    // average aggregation operator
  | ‘MED’    // median aggregation operator
⟨interval-opt⟩ ::= ⟨lbound⟩ ‘,’ ⟨rbound⟩ | ⟨empty⟩
⟨lbound⟩ ::= ‘(’ ⟨bound⟩ | ‘[’ ⟨bound⟩
⟨rbound⟩ ::= ⟨bound⟩ ‘)’ | ⟨bound⟩ ‘]’ | ‘*)’
⟨bound⟩ ::= ⟨integer⟩⟨unit⟩ | ⟨integer⟩
⟨unit⟩ ::= ‘s’ | ‘m’ | ‘h’ | ‘d’
⟨term-list⟩ ::= ⟨term⟩ ‘,’ ⟨term-list⟩ | ⟨term⟩ | ⟨empty⟩
⟨var-list⟩ ::= ⟨var⟩ ‘,’ ⟨var-list⟩ | ⟨var⟩ | ⟨empty⟩
⟨term⟩ ::=
  | ‘(’ ⟨term⟩ ‘)’
  | ⟨term⟩ ‘+’ ⟨term⟩
  | ⟨term⟩ ‘-’ ⟨term⟩
  | ⟨term⟩ ‘*’ ⟨term⟩
  | ⟨term⟩ ‘/’ ⟨term⟩
  | ⟨term⟩ ‘MOD’ ⟨term⟩      // modulo operation
  | ‘-’ ⟨term⟩
  | ‘f2i’ ‘(’ ⟨term⟩ ‘)’     // float to integer conversion
  | ‘i2f’ ‘(’ ⟨term⟩ ‘)’     // integer to float conversion
  | ⟨cst⟩
  | ⟨var⟩
⟨cst⟩ ::= ⟨integer⟩ | ⟨rational⟩ | ‘"’ ⟨string⟩ ‘"’
⟨var⟩ ::= ‘_’ | ⟨string⟩
```

MonPoly: policy specification language



```
⟨predicate⟩ ::=
  | ⟨string⟩ '(' ⟨term-list⟩ ')'
  | 'tp' '(' ⟨term⟩ ')'                    // time point predicate
  | 'ts' '(' ⟨term⟩ ')'                    // timestamp predicate
  | 'tpts' '(' ⟨term⟩ ',' ⟨term⟩ ')'       // time point and timestamp predicate
```

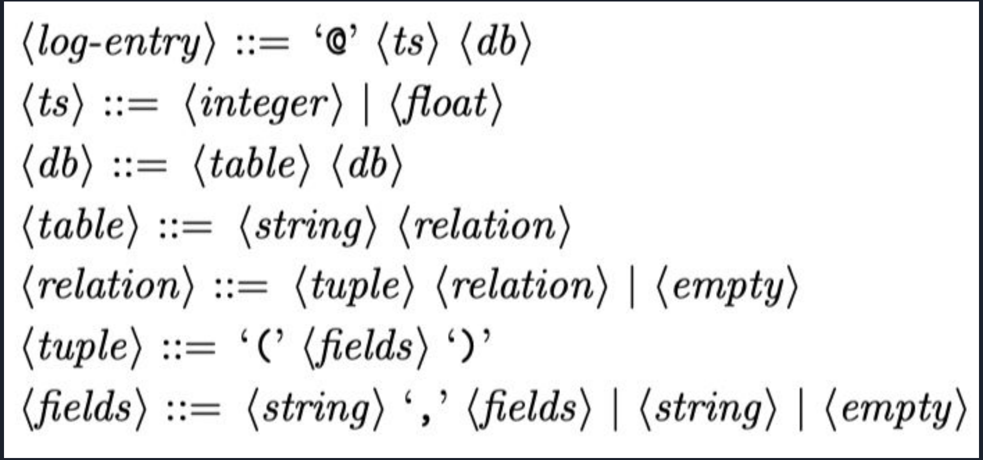


symbol	MonPoly terminal	associativity
¬	NOT	none
∧	AND	left
∨	OR	left
→	IMPLIES	right
↔	EQUIV	left
∃ ∀	EXISTS FORALL	none
● ○ ◆ ◇ ■ □	PREV NEXT ONCE EVENTUALLY PAST_ALWAYS ALWAYS	none
S U	SINCE UNTIL	right

- Example: publish(r) IMPLIES ONCE[0,7d] approve(r)
  "if a report is published then the report must have been approved within the last 7 days"

MonPoly: log entries

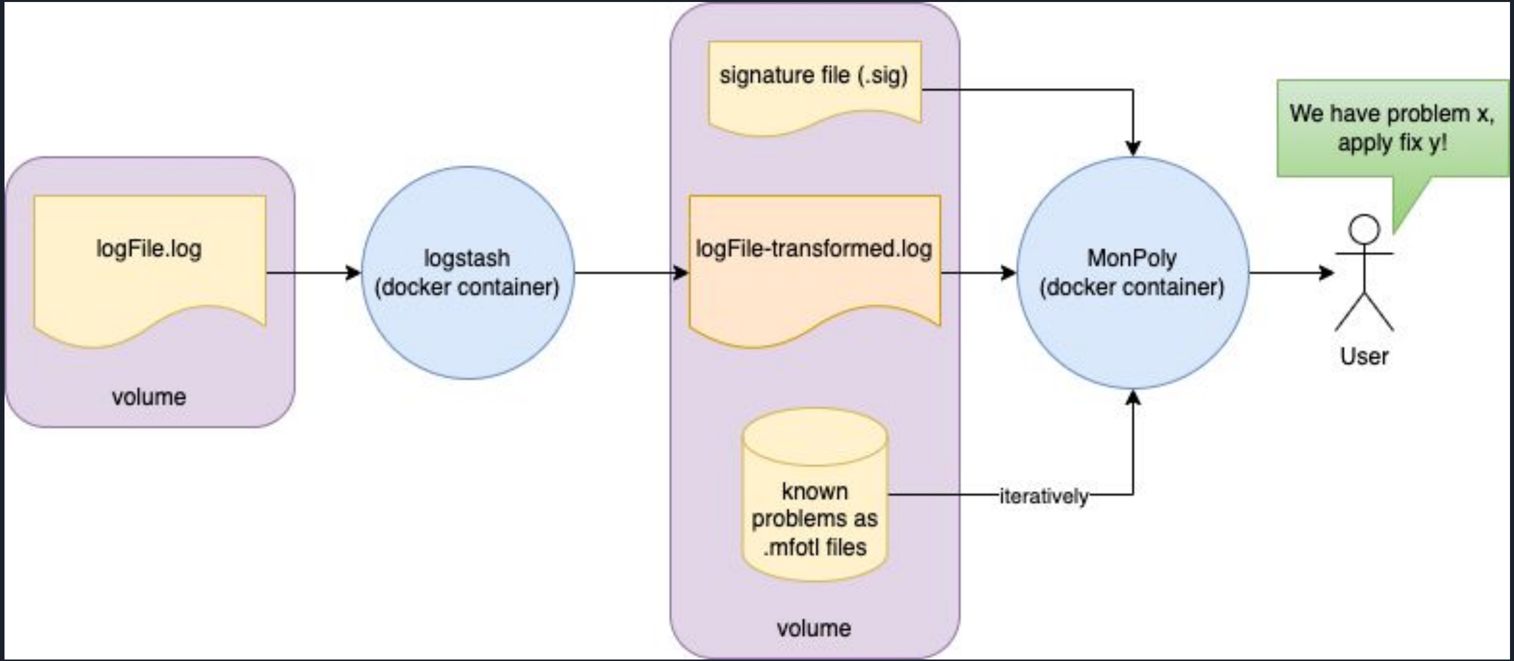
- A log file is a sequence of log entries



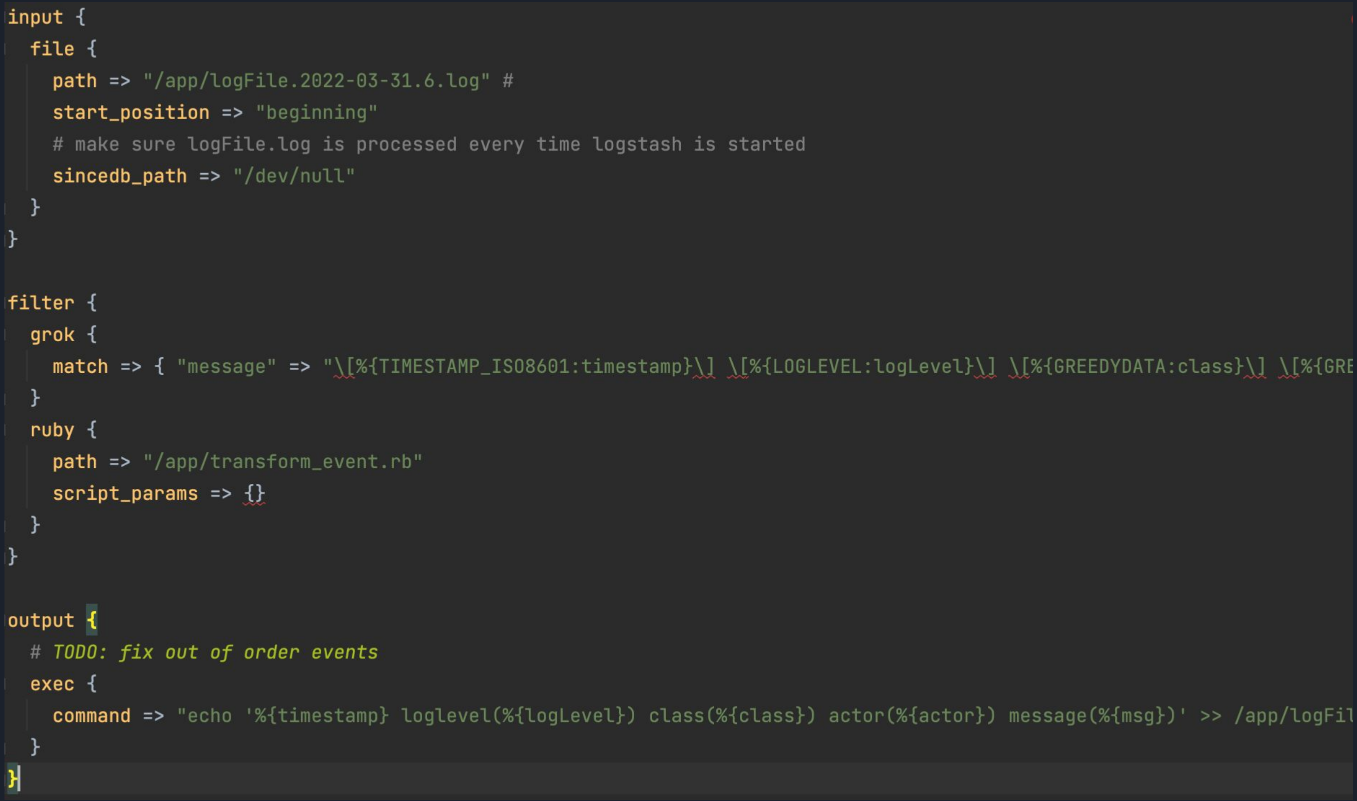
$$\langle \textit{log-entry} \rangle ::= \text{‘@’}\ \langle \textit{ts} \rangle\ \langle \textit{db} \rangle$$
$$\langle \textit{ts} \rangle ::= \langle \textit{integer} \rangle \mid \langle \textit{float} \rangle$$
$$\langle \textit{db} \rangle ::= \langle \textit{table} \rangle\ \langle \textit{db} \rangle$$
$$\langle \textit{table} \rangle ::= \langle \textit{string} \rangle\ \langle \textit{relation} \rangle$$
$$\langle \textit{relation} \rangle ::= \langle \textit{tuple} \rangle\ \langle \textit{relation} \rangle \mid \langle \textit{empty} \rangle$$
$$\langle \textit{tuple} \rangle ::= \text{‘(’}\ \langle \textit{fields} \rangle\ \text{‘)’}$$
$$\langle \textit{fields} \rangle ::= \langle \textit{string} \rangle\ \text{‘,’}\ \langle \textit{fields} \rangle \mid \langle \textit{string} \rangle \mid \langle \textit{empty} \rangle$$

- Example: @1648716381.85 loglevel("DEBUG") message("hello world!")

Artifact building: architecture



Artifact building: logstash

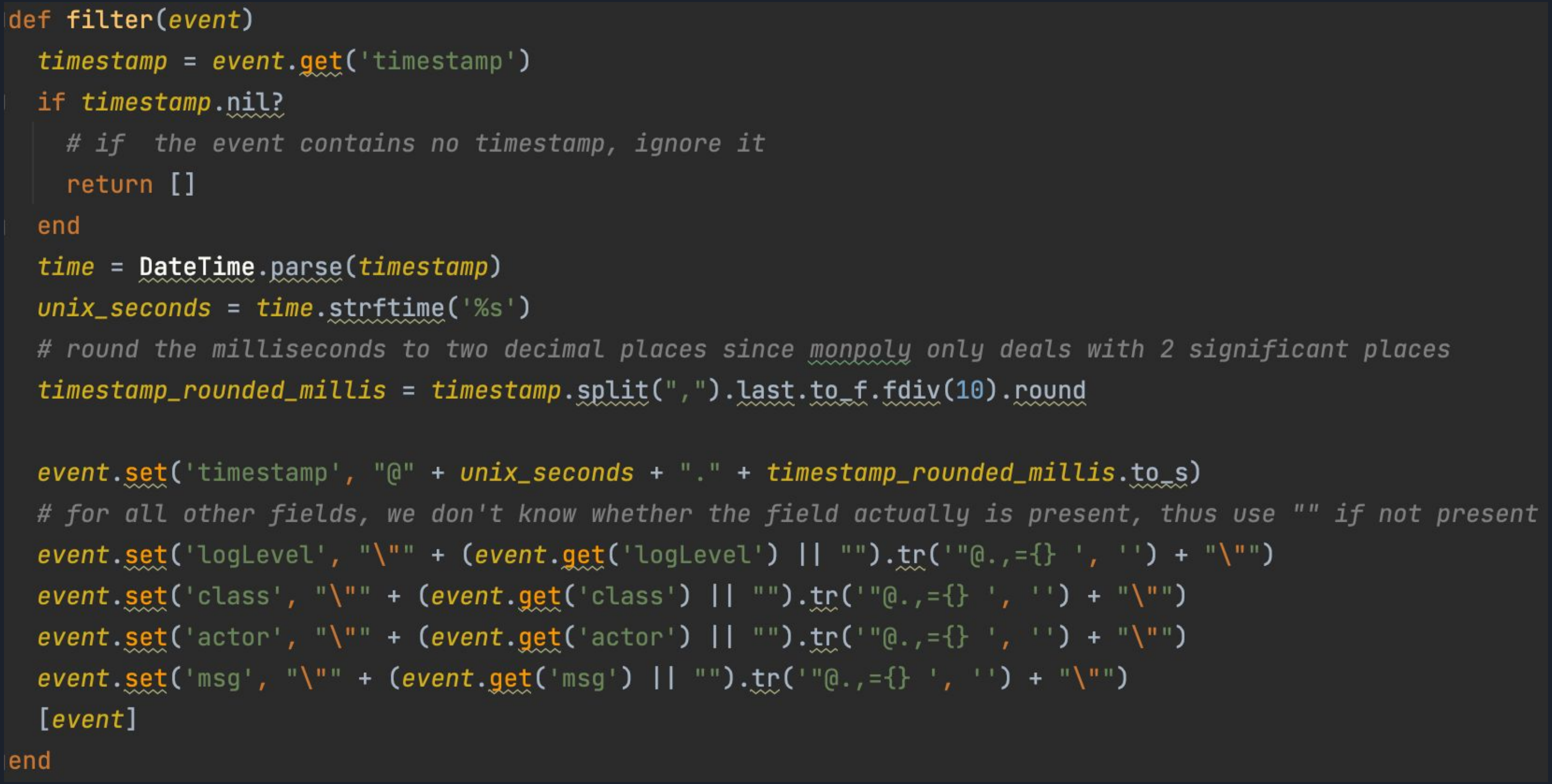


```
input {
  file {
    path => "/app/logFile.2022-03-31.6.log" #
    start_position => "beginning"
    # make sure logFile.log is processed every time logstash is started
    sincedb_path => "/dev/null"
  }
}

filter {
  grok {
    match => { "message" => "\[%{TIMESTAMP_ISO8601:timestamp}\] \[%{LOGLEVEL:logLevel}\] \[%{GREEDYDATA:class}\] \[%{GRE
  }
  ruby {
    path => "/app/transform_event.rb"
    script_params => {}
  }
}

output {
  # TODO: fix out of order events
  exec {
    command => "echo '%{timestamp} loglevel(%{logLevel}) class(%{class}) actor(%{actor}) message(%{msg})' >> /app/logFil
  }
}
```

Artifact building: lexer satisfaction

- We have to contain a newline at the end of the log
- TIMESTAMP_ISO8601 needs to be a unix timestamp, prepended with @ and optional millis
- We’re not allowed to contain the following characters in the log:
  - @
  - “
  - .
  - ,
  - {
  - }
  - (space)

Artifact building: lexer satisfaction



```
def filter(event)
  timestamp = event.get('timestamp')
  if timestamp.nil?
    # if  the event contains no timestamp, ignore it
    return []
  end
  time = DateTime.parse(timestamp)
  unix_seconds = time.strftime('%s')
  # round the milliseconds to two decimal places since monpoly only deals with 2 significant places
  timestamp_rounded_millis = timestamp.split(",").last.to_f.fdiv(10).round

  event.set('timestamp', "@" + unix_seconds + "." + timestamp_rounded_millis.to_s)
  # for all other fields, we don't know whether the field actually is present, thus use "" if not present
  event.set('logLevel', "\"" + (event.get('logLevel') || "").tr('"@.,={} ', '') + "\"")
  event.set('class', "\"" + (event.get('class') || "").tr('"@.,={} ', '') + "\"")
  event.set('actor', "\"" + (event.get('actor') || "").tr('"@.,={} ', '') + "\"")
  event.set('msg', "\"" + (event.get('msg') || "").tr('"@.,={} ', '') + "\"")
  [event]
end
```

Demo

Missing feature

- MonPoly doesn’t implement pattern matching for strings (e.g. regex)
- However, this is a crucial feature for the desired analyzer tool
- So I contacted Prof. Basin…

Missing feature

- and I got an answer from Srdjan, postdoc in prof. Basin's group:
- *In the original version of Monpoly, pattern matching on strings is indeed not supported. However, we have added that feature in the development version of Monpoly available here*
- *Namely, there are two additional "atomic" formulas:*
  - `term1 SUBSTRING term2` , where both terms evaluate to strings
  - `term1 MATCHES term2` , where term1 evaluates to string and term2 to an OCAML regular expression
- *So you could write the desired formula as:*
  - `message(x) AND x MATCHES r".*some text.*"`
  - `message(x) AND "some text" SUBSTRING x`

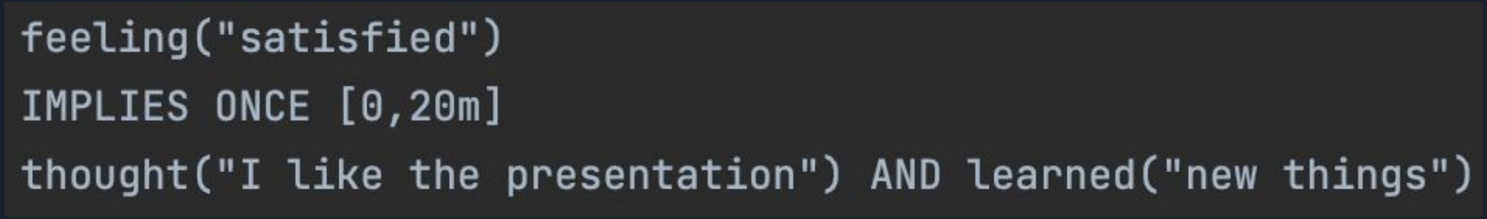
MonPoly: monitorable formulas

- Subformulas of the form `NOT psi` should contain no free variables
  e.g. `NOT loglevel(x)` is not monitorable
- What about unbounded future temporal operators? e.g. `ALWAYS loglevel(“INFO”)`
- Not monitorable -> restrict: `ALWAYS [0,3h] loglevel(“INFO”)`
- MonPoly adds a last time point (largest representable timestamp) at the end of the input event sequence, to evaluate subformulas at all time points in the original event sequence

References

- Overview of MonPoly, including its usage and history (paper)
- MonPoly source code (bitbucket), dev source code(bitbucket)
- ETH research project “Runtime Policy Monitoring and Enforcement” (link)

Q&A



```
feeling("satisfied")
IMPLIES ONCE [0,20m]
thought("I like the presentation") AND learned("new things")
```