VyPR2: A Framework for Runtime Verification of Python Web Services

Joshua Heneage Dawes — University of Manchester and CERN
joshua.dawes@cern.ch
Giles Reger — University of Manchester
Giovanni Franzoni — CERN
Andreas Pfeiffer — CERN
Giacomo Govi — Fermi National Accelerator Laboratory

Outline

A new temporal logic for specifying performance requirements.

Instrumentation and monitoring for this logic.

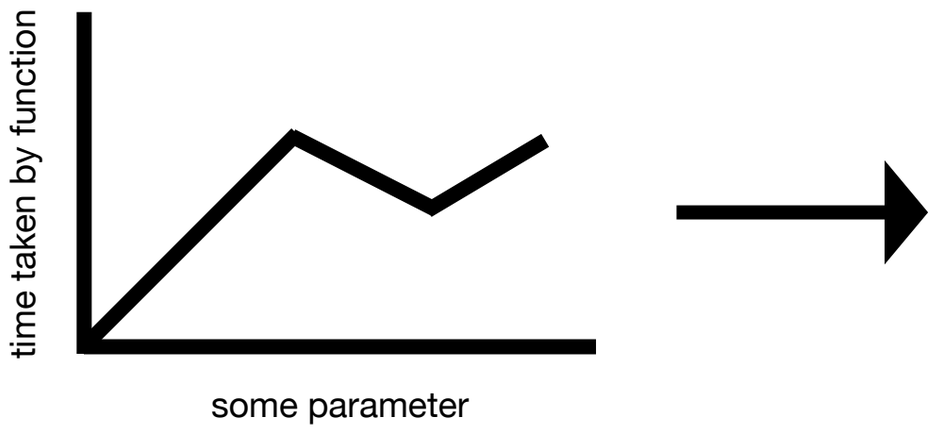
An operational view of monitoring web services with this new theory - the VyPR2 framework.

The first major application of VyPR2 to critical infrastructure on the CMS Experiment at CERN.

Towards Automated Performance Analysis of Python

Python? Language of choice for certain infrastructure on the CMS Experiment.

At CMS, performance analysis has historically been a manual effort.



When did the execution time exceed some t?

How often?

For which inputs?

On which branch in control-flow?

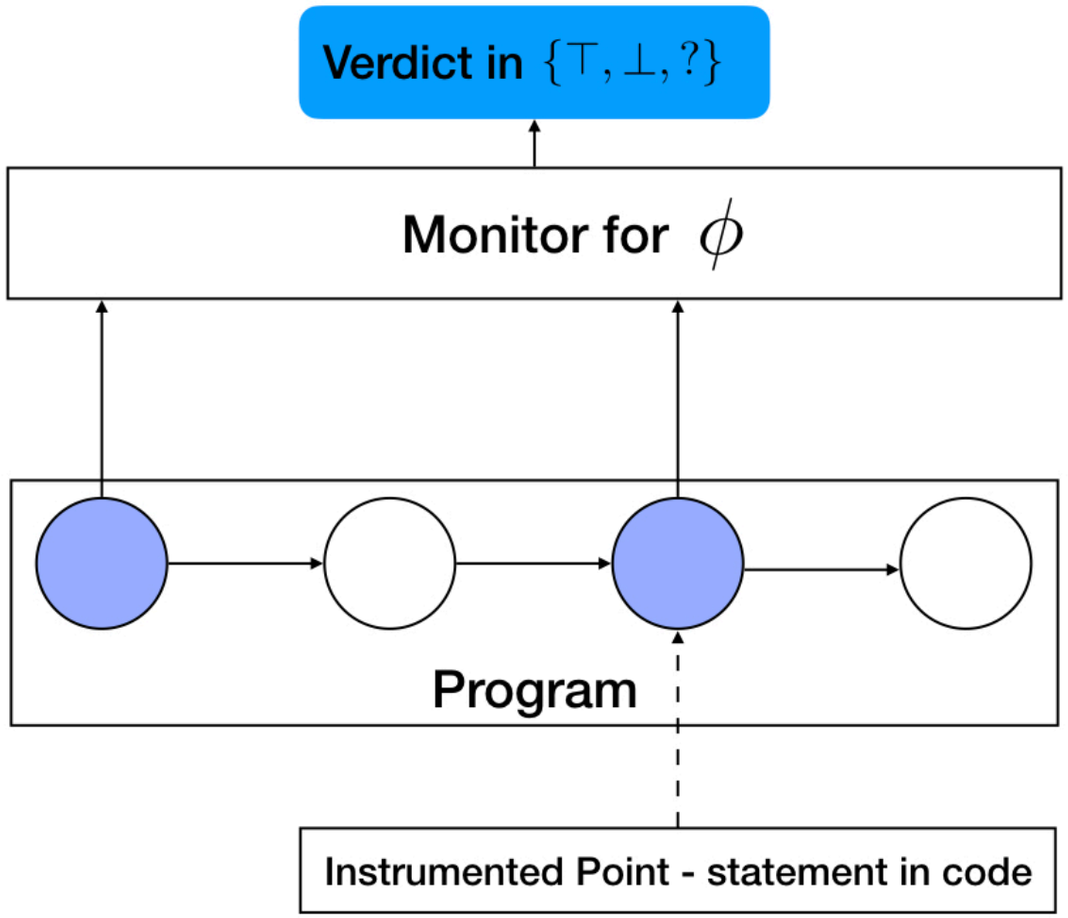
Standard profiling is nice, but…

Code bases are gaining in complexity with new use cases from detector experts.

The Large Hadron Collider will soon generate far more data after a period of upgrades.

On CMS, we need a more sophisticated performance analysis technique.

Runtime Verification



"Lightweight formal method"

Properties are normally about ordering (Linear Temporal Logic - LTL)

Sometimes with data involved (Quantified Event Automata - QEA), sometimes with time (Metric Temporal Logic - MTL).

Usually (LTL, MTL) **high-level, abstract.**

Why is “abstract” not ideal for us?

Linear Temporal Logic, Metric Temporal Logic - very abstract wrt code.

$\mathcal{G}(p \implies \mathcal{X}q)$ no meaning on its own!

Existing specification languages have shown to be unintuitive to write/understand for engineers at CMS.

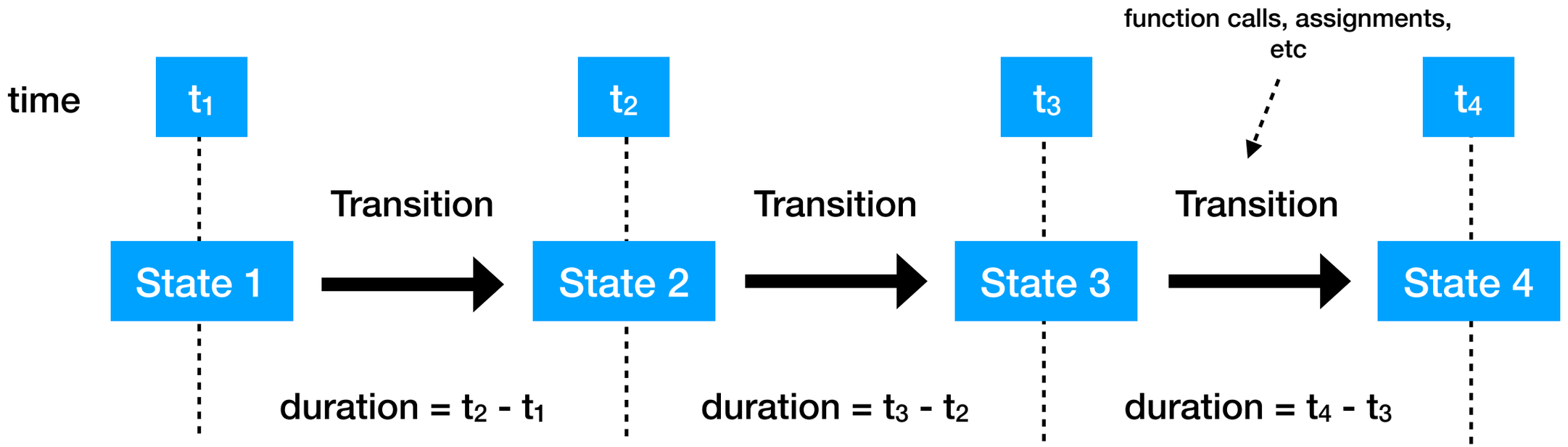
Control-Flow Temporal Logic (CFTL)

A **low-level**, linear-time temporal logic.

CFTL property + program = immediate meaning.

A logic for description of performance requirements.

General Idea



We think of program runs as **dynamic runs**:

Concrete States (instantaneous checkpoints with an associated timestamp) - $\langle t, \sigma, \tau \rangle$

Transitions (computation performed to reach one state from another) - pairs of states - $\langle \langle t, \sigma, \tau \rangle, \langle t', \sigma', \tau' \rangle \rangle$

From this, we can pick **points of interest** and require that some property holds at each one.

What kinds of properties?

"Every time *func* is called, the resulting state should leave x with a value no more than 10"

"If *auth* is changed to True, every future call to *execute* should take no more than 0.5 seconds"

"Whenever *hashes* is changed, the next call to *find_new_hashes* should take no more than 0.3 seconds"

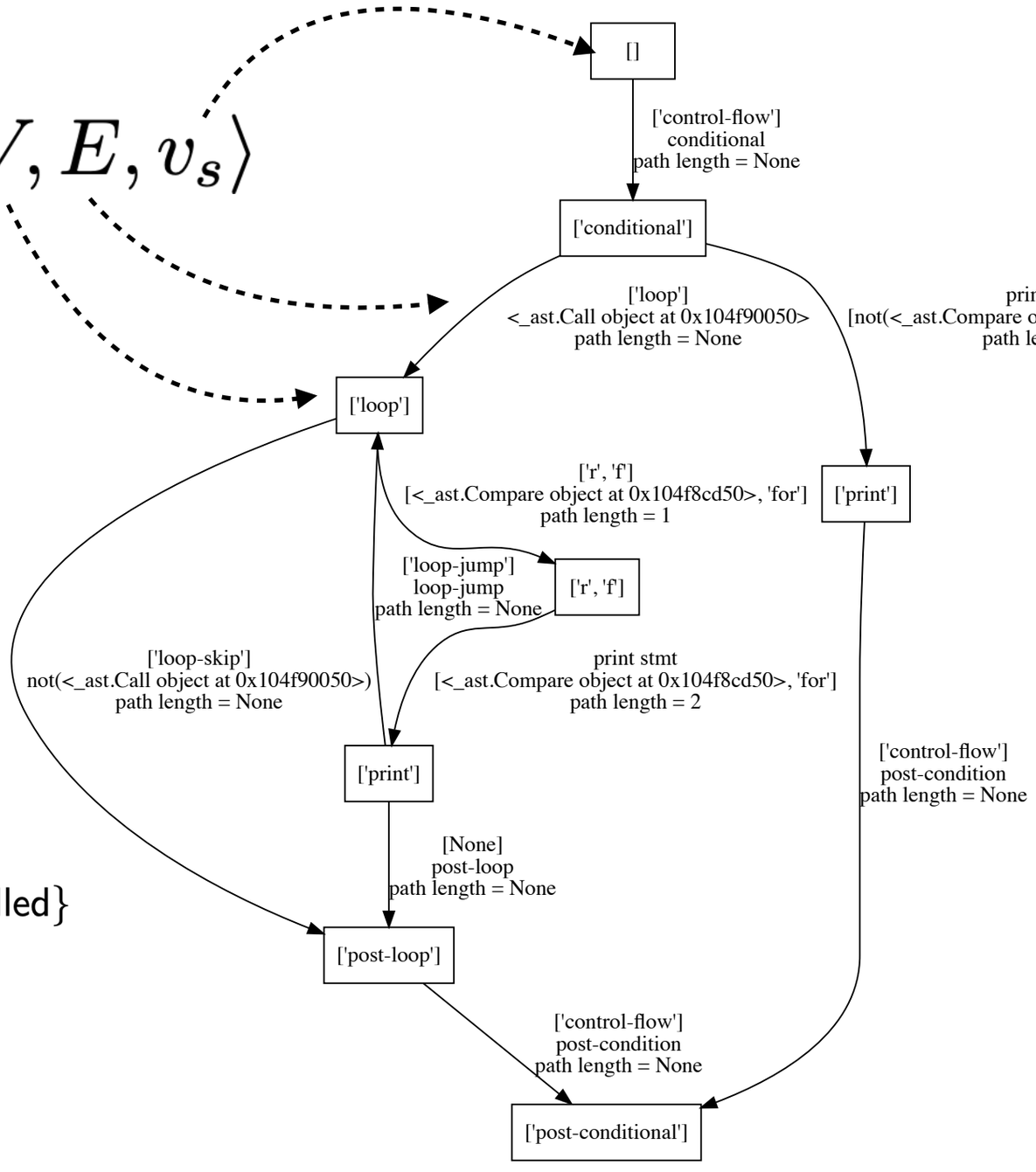
Symbolic Control-Flow Graphs (SCFGs)

$$\mathsf{SCFG}(P) = \langle V, E, v_s \rangle$$

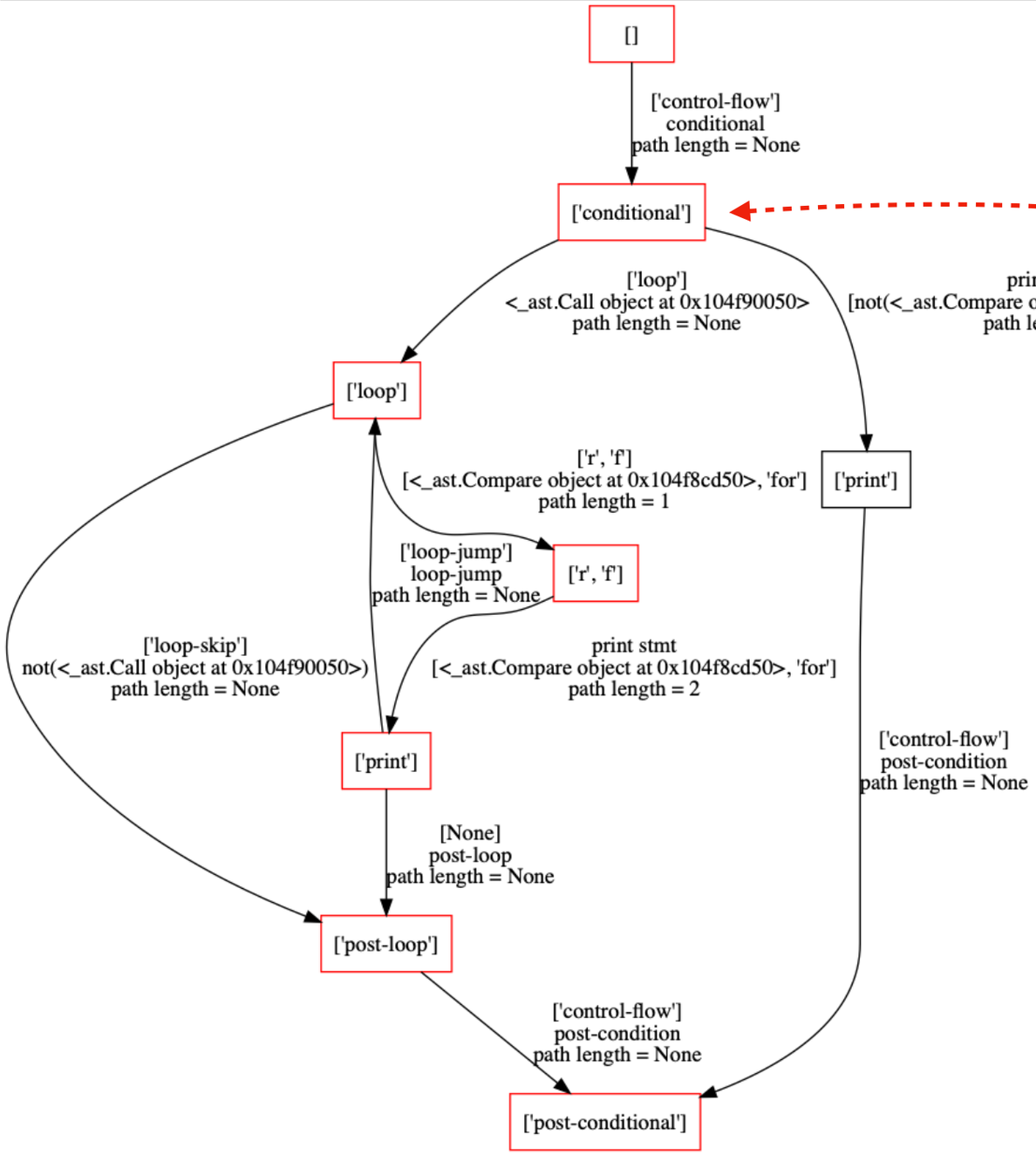
```
if n > 1:
  for i in range(n):
    r = f(i)
    print(r)
else:
  print("nope")
```

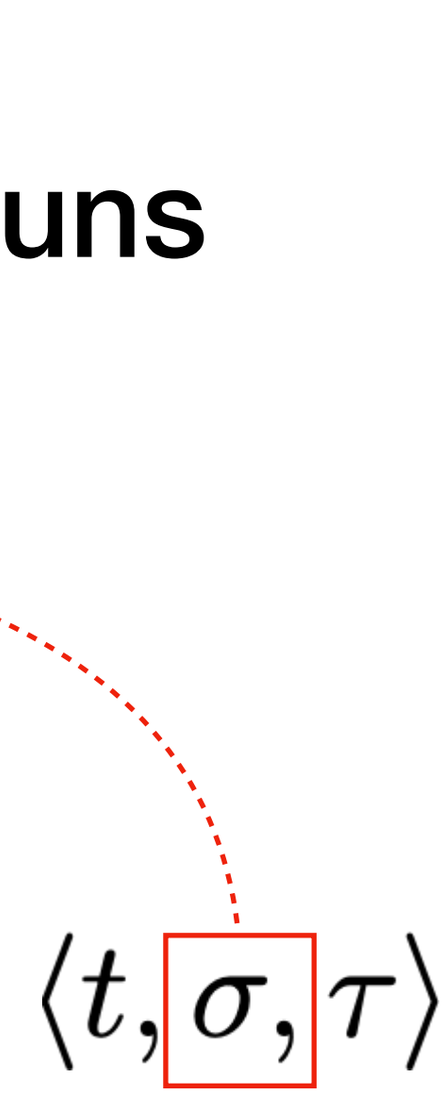
variables/functions in P

$\sigma : \mathsf{Sym} \rightarrow \{\text{undefined, unchanged, changed, called}\}$



SCFGs to Dynamic Runs





Dynamic Runs identify with paths through SCFGs.

Each concrete state in a Dynamic Run contains a symbolic state.

This will help when we need to perform instrumentation.

CERN

MANCHESTER 1824
The University of Manchester

Building CFTL Formulas

$$\varphi \equiv \forall q_1 \in \Gamma_1, \ldots, \forall q_n \in \Gamma_n : \psi(q_1, \ldots, q_n)$$

I will focus on singly-quantified formulas

$$\varphi \equiv \forall q \in \Gamma : \psi(q)$$

A boolean combination of predicates on q and surrounding concrete states/transitions.

Γ is a predicate on concrete states/transitions in a dynamic run, eg:

$$\langle\langle t, \sigma, \tau\rangle, \langle t', \sigma', \tau'\rangle\rangle \vdash \mathsf{calls}(f) \iff \sigma'(f) = \mathsf{called}$$

Syntax

points of interest selection

propositional connectives

$$
\begin{array}{lll}
\phi & := & \forall^{S} q \in \Gamma_S : \phi \mid \forall^{T} t \in \Gamma_T : \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi_A \mid \mathit{true} \\
\phi_A & := & S(x) = v \mid S(x) \in (n,m) \mid S(x) \in [n,m] \mid \\
 & & \mathsf{duration}(T) \in (n,m) \mid \mathsf{duration}(T) \in [n,m] \\
\Gamma_S & := & \mathsf{changes}(x) \mid \mathsf{future}(q, \mathsf{changes}(x)) \\
\Gamma_T & := & \mathsf{calls}(f) \mid \mathsf{future}(q, \mathsf{calls}(f)) \\
S & := & q \mid \mathsf{source}(T) \mid \mathsf{dest}(T) \mid \mathsf{next}(S, \mathsf{changes}(x)) \\
T & := & t \mid \mathsf{incident}(S) \mid \mathsf{next}(S, \mathsf{calls}(f))
\end{array}
$$

construction of predicates on states and transitions

Atoms in CFTL formulas are those terms generated by ϕ_A

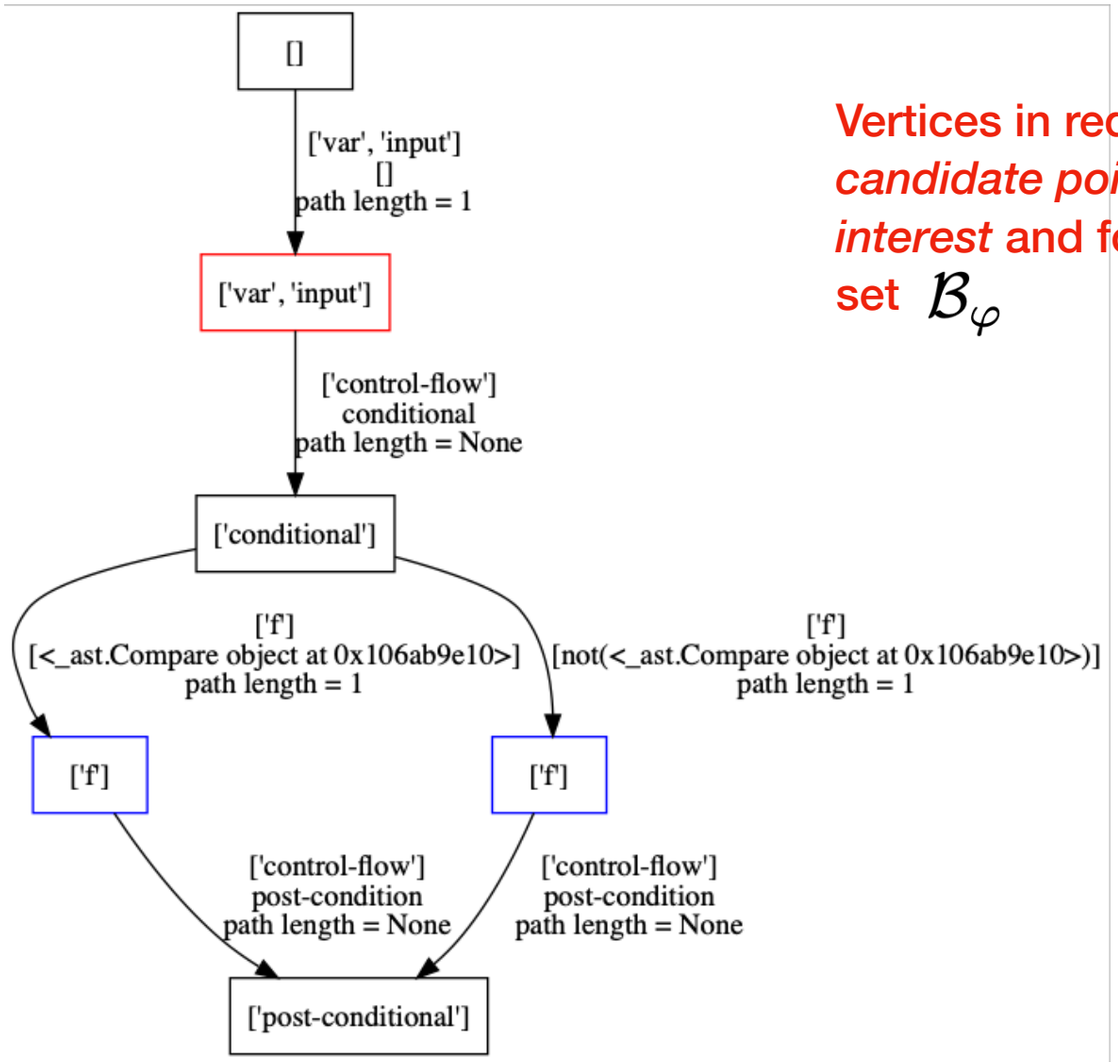
A simple example

$$\forall q \in \mathsf{changes}(\texttt{var}) :$$
$$\mathsf{duration}(\mathsf{next}(q, \mathsf{calls}(f))) \in (0, 0.1)$$

"Every time *var* changes, the next call to *f* takes less than 0.1 seconds"

How do we instrument for CFTL?

$$\forall q \in \text{changes}(\texttt{var}):$$
$$\text{duration}(\text{next}(q, \text{calls}(f))) \in (0, 0.1)$$



Vertices in red are *candidate points of interest* and form a set $\mathcal{B}_\varphi$

The final instrumentation points in blue are organised into a tree $\mathcal{H}_\varphi(\langle i_\mathcal{B}, i_\alpha \rangle)$

$i_\mathcal{B}$ index of the relevant candidate point of interest

i_α index of the relevant atom

Monitoring for CFTL

$\forall q \in \mathsf{changes}(\mathtt{var}) : \ldots$

(1)Give each candidate point of interest an index.

(2)Give each atom an index.

(3)Attach to instruments each of these indices.

(4)Instantiate a monitor whenever a state changing *var* is observed.

Monitoring for CFTL

$$\mathsf{duration}(\mathsf{next}(q, \mathsf{calls}(f))) \in (0, 0.1)$$

(1)When an instrument fires at runtime, use its candidate point of interest index to find all relevant monitors.

(2)Use the instrument’s atom index to find which part of the monitor state to update.

In practice, this is very efficient.

Our Setting

The first major application of VyPR2 was to a service used to upload Conditions data for the CMS Experiment at CERN.

Conditions data describe the alignment and calibrations of the CMS detector.

They are needed for a full reconstruction of data taken, ready for physics analysis.

VyPR2 helped us detect significant performance drops.

A Pipeline for Verification

(1)Web Services are deployed to Production Machines.

(2)After deployment, instrument based on a specification configuration file written in VyPR2's PyCFTL library.

(3)Monitor for the specification at runtime.

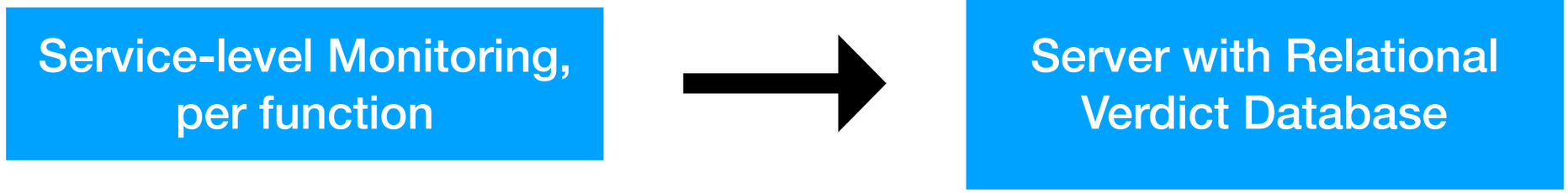
(4)Perform offline analysis using a central verdict server.

PyCFTL

$$\forall q \in \mathsf{changes}(\mathsf{val}) :$$
$$\mathsf{duration}(\mathsf{next}(q, \mathsf{calls}(\mathsf{func}))) \in [0, 3]$$

```
Forall(q = changes('val')).\
Check(lambda q : (
  q.next_call('func').duration()._in([0, 3])
))
```

A Verdict Server



For a given HTTP request, function and property combination, what were the verdicts generated by monitoring a property across all calls?

For a given verdict and subsystem, which function/property pairs generated the verdict?

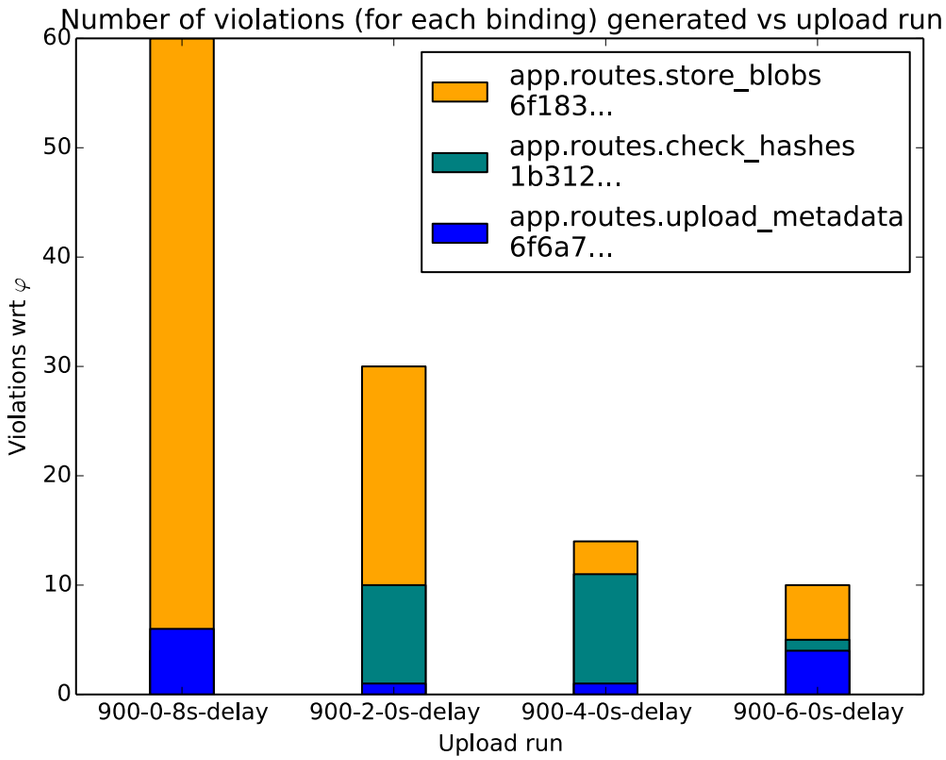
For a given function call and verdict, which lines were part of bindings that generated this verdict while monitoring some property?

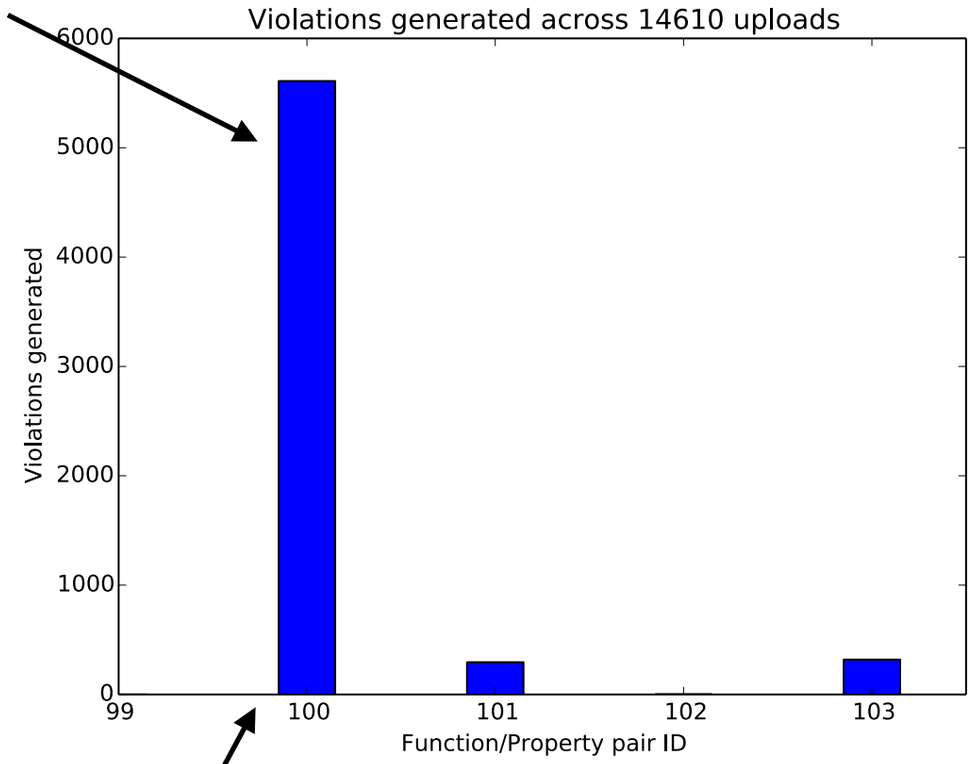
Results

Collaboration with the CMS Alignment, Calibrations and Databases group.

Recorded ~14,600 Conditions uploads over 6 months of LHC runs.

These violations are from a property over code
that was supposed to be an optimisation.





$$\forall q \in \text{changes}(\texttt{hashes}) : \text{duration}(\text{next}(q, \text{calls}(\texttt{new\_hashes}))) \in (0, 0.3)$$

The University of Manchester

To conclude…

VyPR2 is, to the best of our knowledge, the first application of Runtime Verification in High Energy Physics.

We also haven't found any related work on web services.

VyPR2 has helped to detect significant performance drops in a service that is critical to the physics reconstruction pipeline of the CMS Experiment.

Next steps: explanation (paths taken, state present, etc) of performance drops.

Using VyPR

VyPR (without extension to web services) - http://cern.ch/vypr

Feel free to contact me (joshua.dawes@cern.ch) if you have a potential use for VyPR or its web service extension VyPR2.