Now You See Me, Now You Don't: Querying with Hybrid Temporal Logic

Mistral Contrastin Andrew Rice

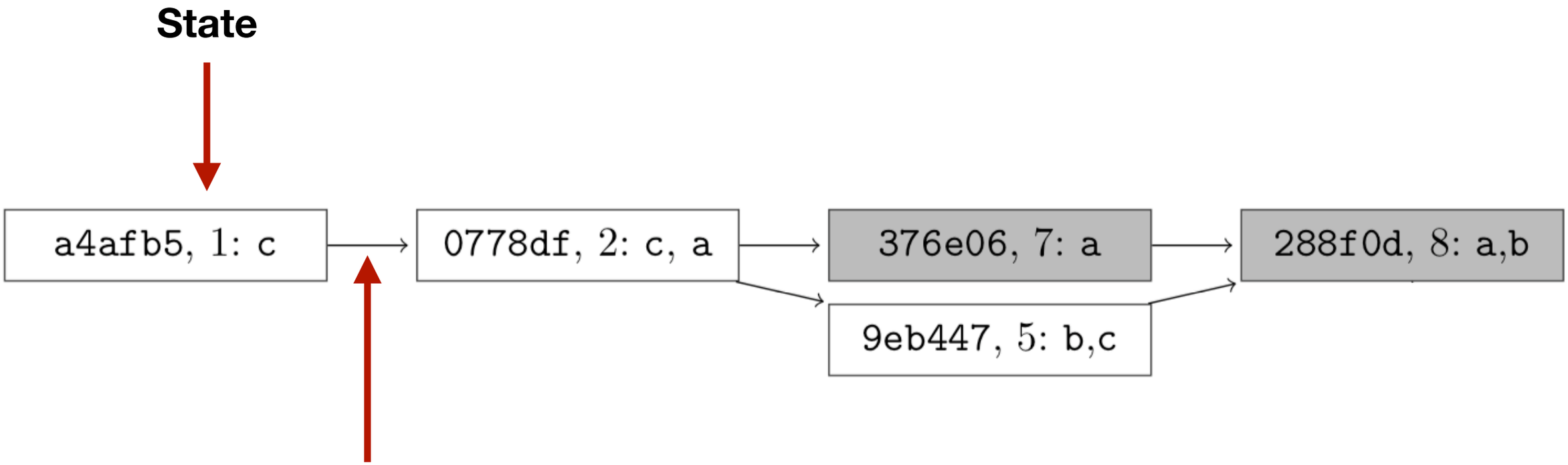
Computation Tree Logic recap

$\phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid$ **EX** $\phi \mid$ **AX** $\phi \mid$ **EF** $\phi \mid$ **AF** $\phi \mid$ **EG** $\phi \mid$ **AG** $\phi \mid$ **E**$[\phi$ **U** $\phi] \mid$ **A**$[\phi$ **U** $\phi]$

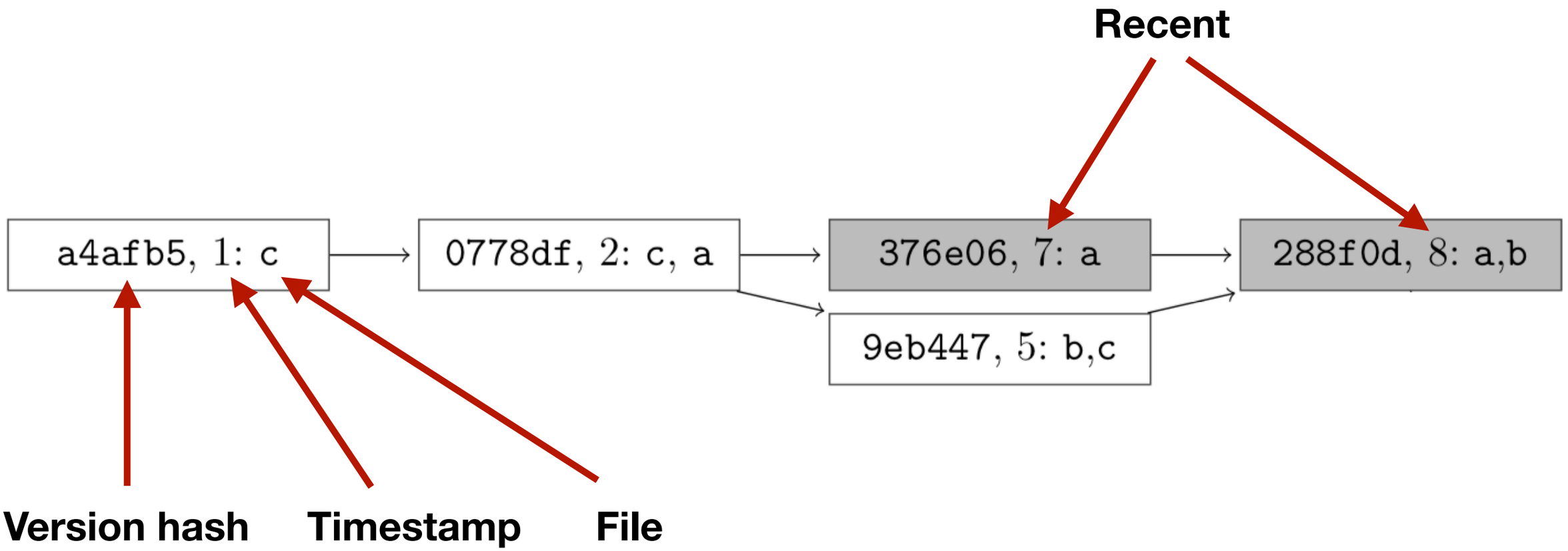
- Evaluate with respect to a Kripke structure:
  (States, StartStates, AccessibilityRelation, LabellingFx)

- `EX/AX p`: *p* holds in some/all succeeding states
- `EF/AF p`: in some/all paths we can find a point *p* holds
- `EG/AG p`: in some/all paths we can *p* holds
- `E/A[p U q]`: in some/all paths *p* holds until *q* holds

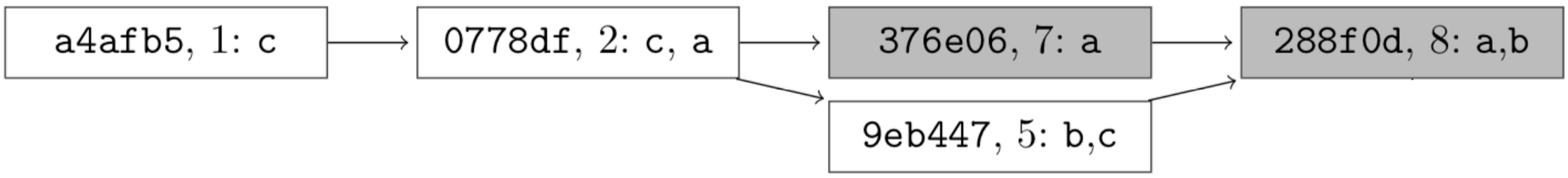
Version history looks like Kripke structure



Version history looks like Kripke structure



Some simple questions



- Does *a* continuously exist in **some** branches?
- Does *a* continuously exist in **all** branches?
- Does *a* continuously exist in some branches **starting from the first commit**?
- Are there **recent** commits where *a* is present?

Some interesting questions

- When is this bug introduced?
- Which releases are affected by a particular bug?
- What are my colleagues working on?
- Are there discrepancies in the version history? (force push)
- More temporal questions in Fritz and Murphy, 2010

Datalog is not declarative 1

Can I **reach** a point the file *a* exists?

CTL:

```
EF file(“a”)
```

Datalog:

```
eventually(V) :- version(V,_), file(“a”,V).
eventually(V) :- version(V,V’), eventually(V’).

?- eventually(V).
```

Datalog is not declarative 2

Does *a* **always** exist in **all** branches?

CTL:

```
AG file(“a”)
```

Datalog:

```
eventually(V) :- version(V,_), ! file(“a”,V).
eventually(V) :- version(V,V’), eventually(V’).

?- version(V,_), ! eventually(V).
```

Datalog LITE: A deductive query language with linear time model checking

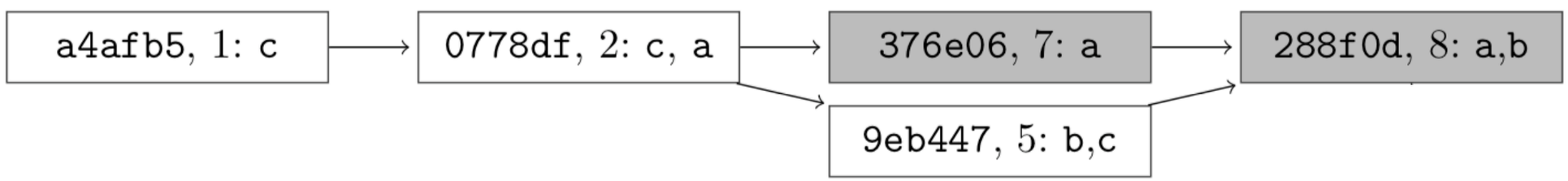
GEORG GOTTLOB
Vienna University of Technology
ERICH GRÄDEL
RWTH Aachen
and
HELMUT VEITH
Vienna University of Technology

We present Datalog LITE, a new deductive query language with a linear time model checking algorithm, i.e., linear time data complexity and program complexity. Datalog LITE is a variant of Datalog that uses stratified negation, restricted variable occurrences and a limited form of universal quantification in rule bodies.

Despite linear time evaluation, Datalog LITE is highly expressive: It encompasses popular modal and temporal logics such as CTL or the alternation-free μ-calculus. In fact these formalisms have natural presentations as fragments of Datalog LITE. Further Datalog LITE is equivalent to the alternation-free portion of guarded fixed point logic. Consequently, linear time model checking algorithms for all mentioned logics are obtained in a unified way.

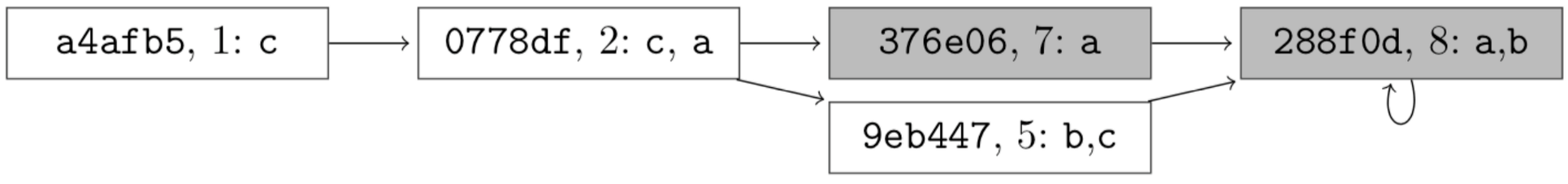
CTL is not perfect either 1: Model processing

- Version histories are not quite **left total**!
  CTL falls apart, e.g., **AG** $p \neq$ ¬**EF** ¬p

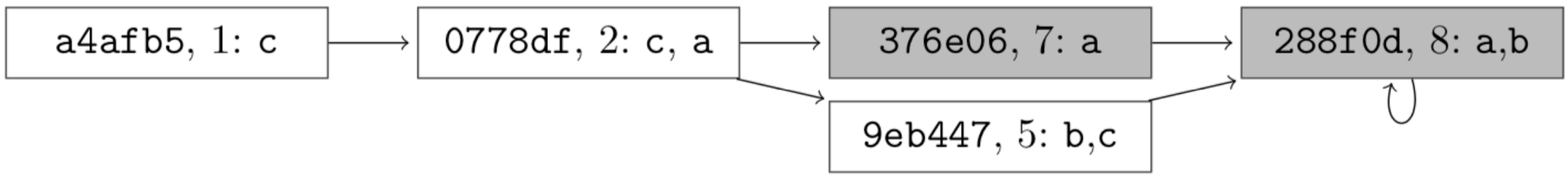


- In Datalog, we can make relations left total

```
total_version(T,T’)  :- version(T,T’).
total_version(T’,T’) :- version(_,T’), ! version(T’,_).
```

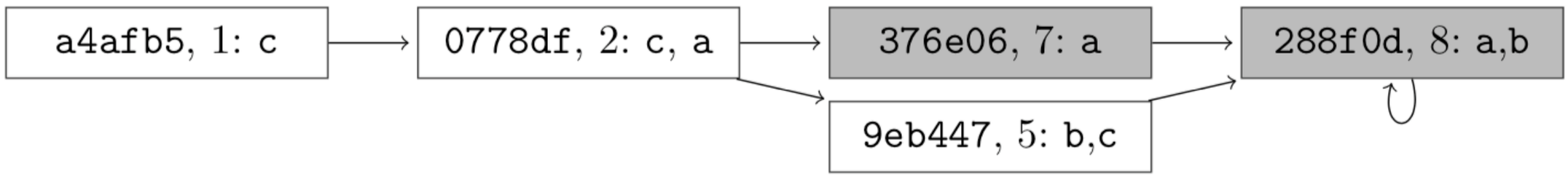


CTL is not perfect either 2



Does `EG file("a")` hold?

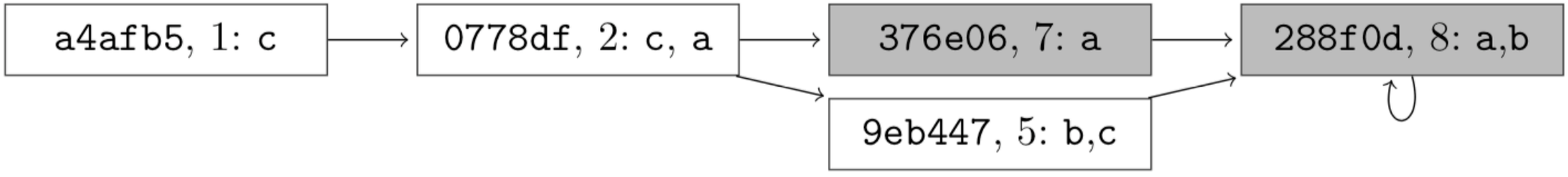
CTL is not perfect either 2: Global vs Local



Does `EG file("a")` hold?

Depends on if the model checker is **global** or **local**.
The query is not informative enough.

Hybrid temporal logic: Setting time



- We also want to **jump** to a particular moment: @

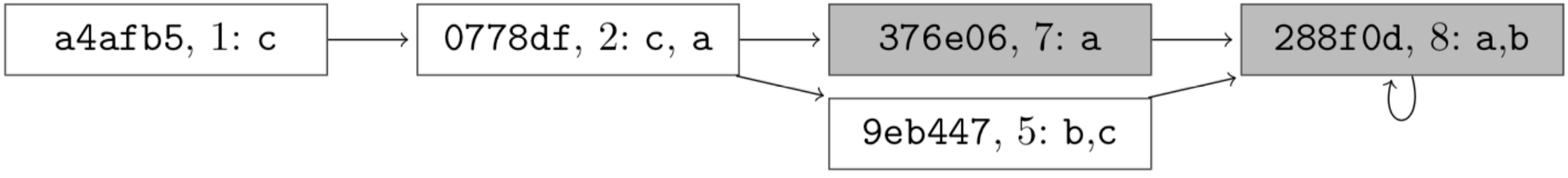
```
?- (AG file(“a”)) @ “0778df”.
```

CTL is not perfect either 3: Mixing with FOL

- How can we say “p holds in recent versions”?
- `?- EX … EX p ?`
- `?- recent, p ?`
- `recent(T) :- T > 6.`

  `?- recent(T), p ?`

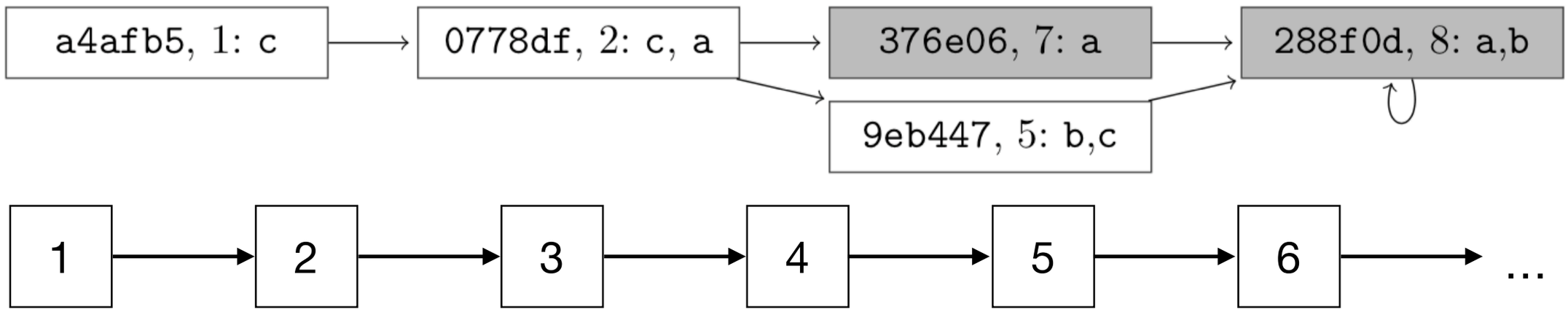
Hybrid temporal logic: Observing time



- We need to **bind** time to a term: ↓ (| in ASCII)

```
?- | Hash (even(Hash), file(“a”)).
```

CTL is not perfect either 4: Multimodality



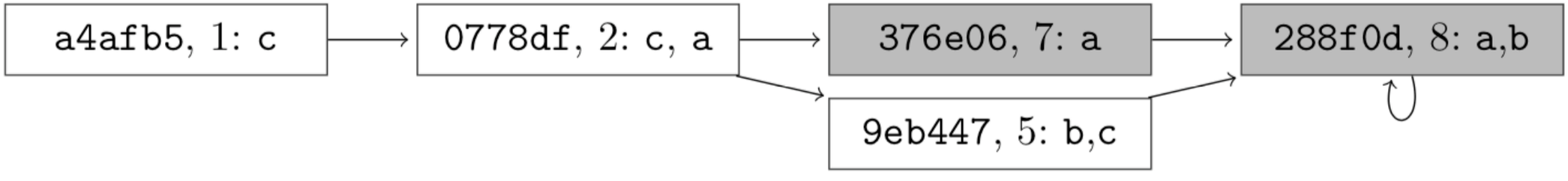
- What is the correct notion of time?

  Version hashes: `a4afb5`, `0778df`, …

  POSIX time: 1, 2, 3, 4, 5, …

- Why not both?

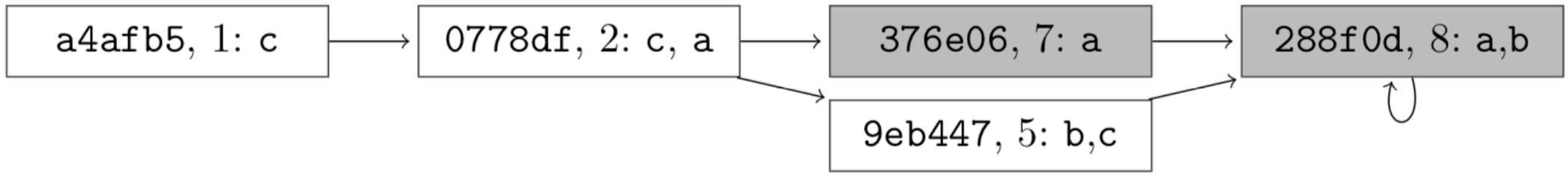
Defining accessibility relation within Datalog




```
.pred version(text,text).
version("a4afb5","0778df").
version("0778df","376e06").
version("376e06","288f0d").
version("0778df","9eb447").
version("9eb447","288f0d").
version("288f0d","288f0d").
```

```
.pred clock(int,int).
clock(1,2).
clock(2,3).
clock(3,4).
clock(4,5).
clock(5,6).
clock(6,7).
clock(7,8).
clock(8,9).
clock(9,-1).
clock(-1,-1).
```

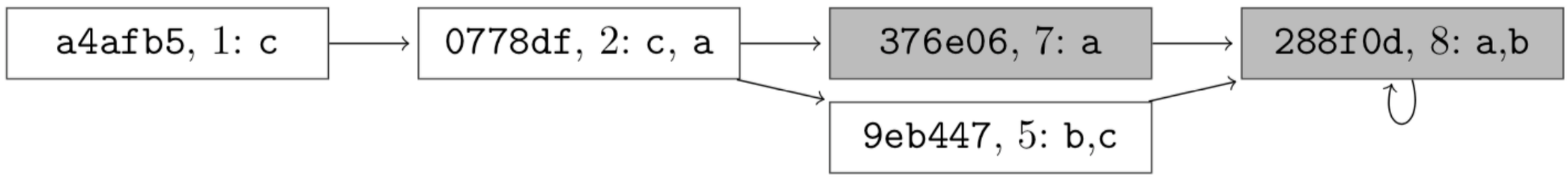
Defining a temporal predicate




```
.pred file(text) @ version.
file("a") @ "0778df".
file("a") @ "376e06".
file("a") @ "288f0d".
file("b") @ "9eb447".
file("b") @ "288f0d".
file("c") @ "a4afb5".
file("c") @ "0778df".
file("c") @ "9eb447".
```

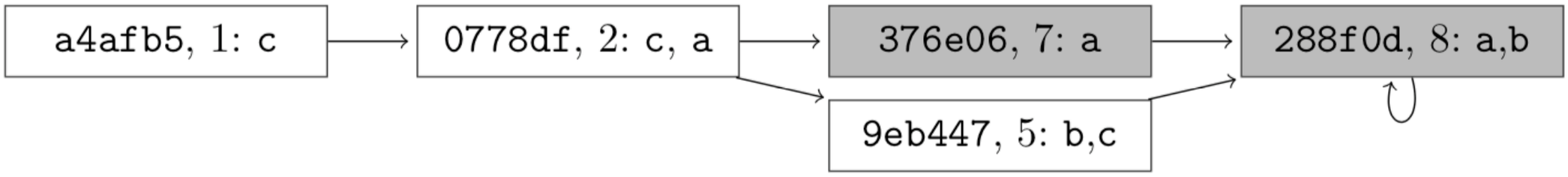
```
?- AG file(“a").

?- file(“c”), EX ! file(“c”).

?- AG (file(“a”); file(“c”)).
```

Relating distinct times




```
.pred timestamp() @ version clock.
timestamp() @ "a4afb5" @ 1.
timestamp() @ "0778df" @ 2.
timestamp() @ "9eb447" @ 5.
timestamp() @ "376e06" @ 7.
timestamp() @ "288f0d" @ 8.
```

Hybrid queries and joins



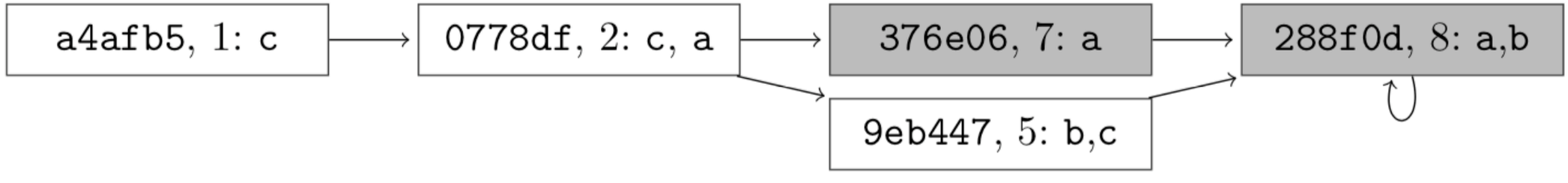
- "Which files exist in recent commits?"

```
?- timestamp(), file(File), | <clock> T recent(T).
```

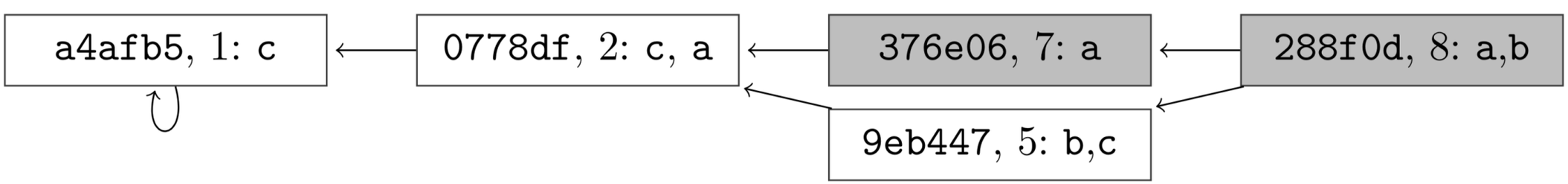
- The `timestamp` is just a *join*

```
.join timestamp.
?- file(File), | <clock> T recent(T).
```

Model processing: Reversing time!




```
.pred rev_version(text,text).
rev_version(H,H') :- version(H',H), ! version(H,H).
rev_version(H,H)  :- version(H,H'), ! version(_,H).

?- AG | <rev_version> Hash (file("c.txt") @ Hash).
```



Interesting accessibility relations in programming language context

- Program counter
- Nodes of a dataflow graph,
  e.g., Brauer et al. 2009 for pointer analysis
- ???

Thanks. Questions?

Implementation: github.com/madgen/temporalog

Email: Mistral.Contrastin@cl.cam.ac.uk
Website: dodisturb.me
Twitter: @madgen_

Compilation

- Compile to stratified Datalog
- Simple stratification criterion: no cycles with `AX`, `AF`, `AG`, `!`
- Range restriction is preserved
- Dataflow safety (well-modedness) is preserved