The Chase in Lean

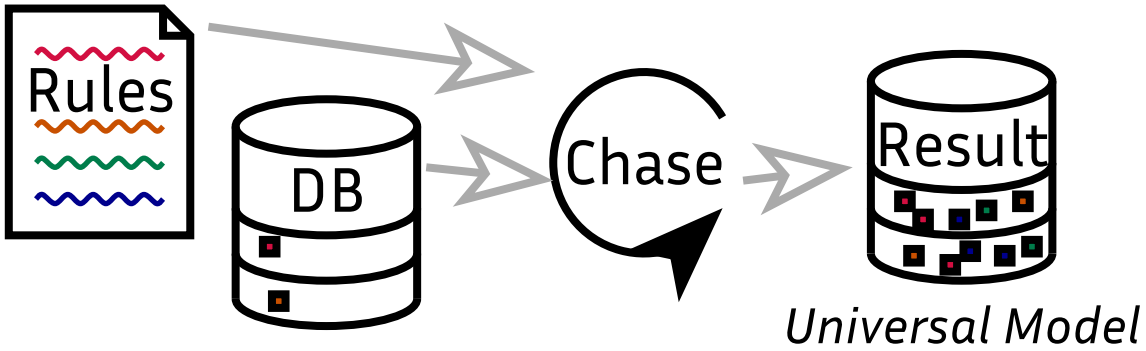
Crafting a Formal Library for Existential Rule Research

Lukas Gerlach

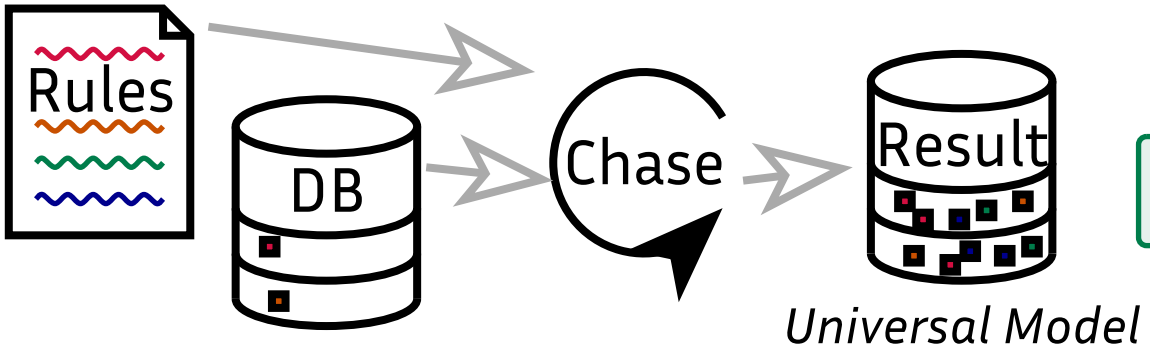
Knowledge-Based Systems Group, TU Dresden, Germany

21.07.2026

Chase Crash Course



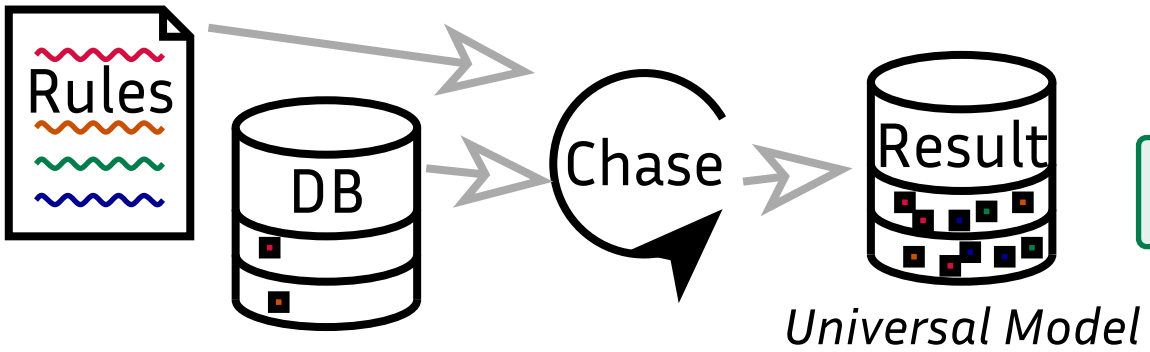
Chase Crash Course



Universal Model

$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

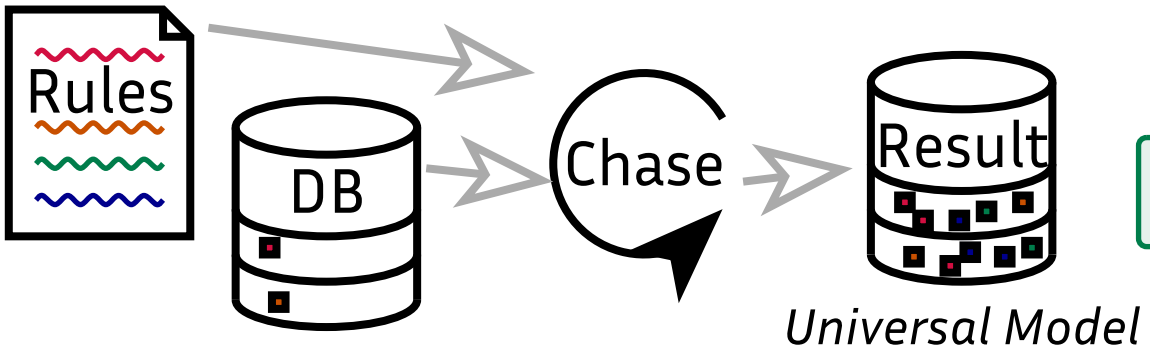
Chase Crash Course



$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

DB: $\mathrm{R}(c, d)$

Chase Crash Course

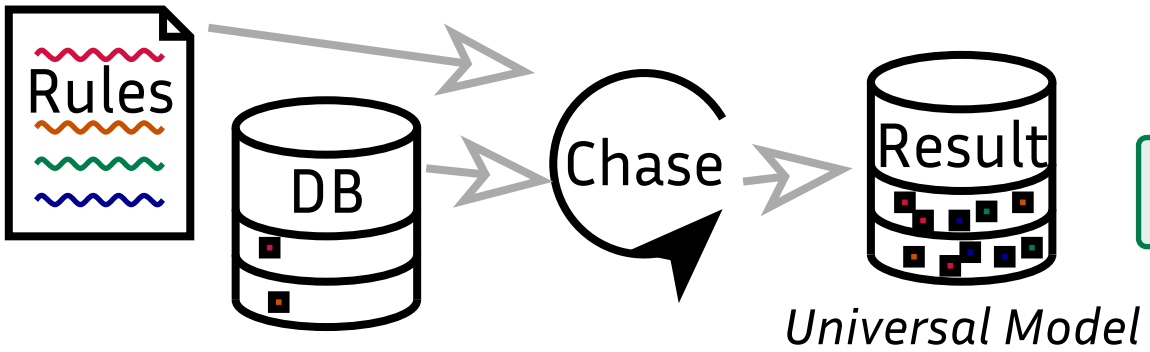


$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

DB: $\mathrm{R}(c, d)$

Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

Chase Crash Course



Universal Model

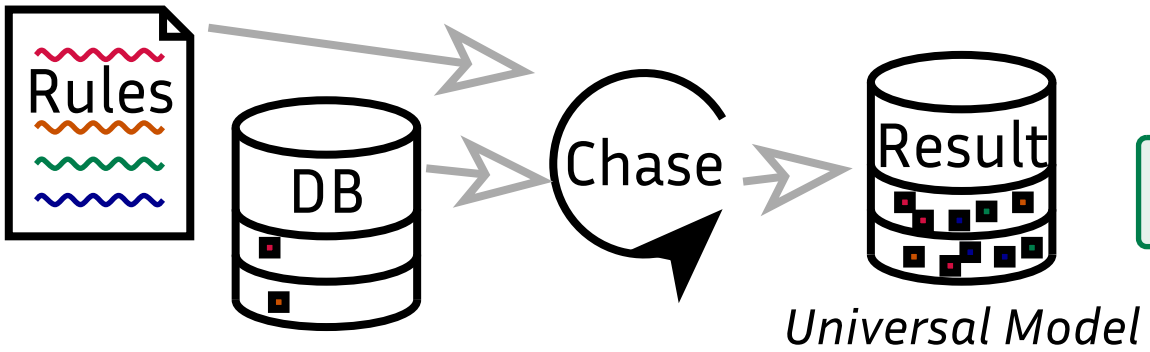
$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

DB: $\mathrm{R}(c, d)$

Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

Step 2: $\mathrm{R}(f_z(d), f_z(f_z(d))), \ldots$

Chase Crash Course



$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

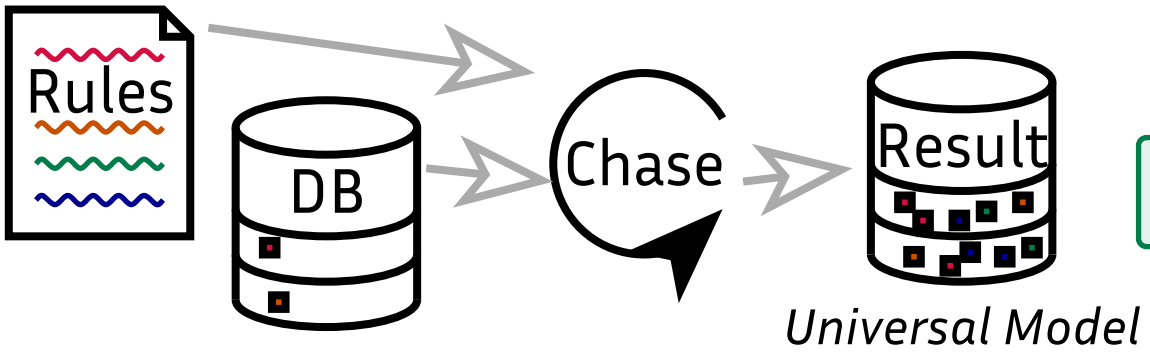
DB: $\mathrm{R}(c, d)$

Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

Step 2: $\mathrm{R}(f_z(d), f_z(f_z(d))), \ldots$

Step N: ...

Chase Crash Course



Universal Model

$$\mathrm{R}(x, y) \rightarrow \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

DB: $\mathrm{R}(c, d)$

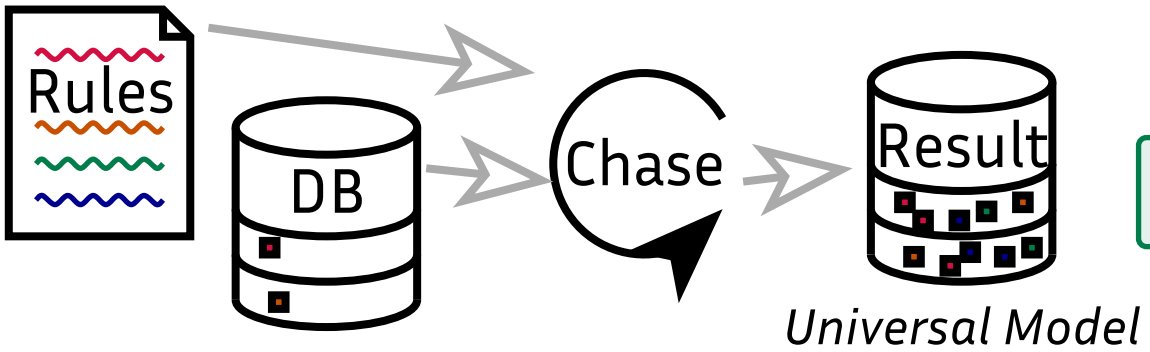
Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

Step 2: $\mathrm{R}(f_z(d), f_z(f_z(d)))$, ...

Step N: ...

DB: $\mathrm{R}(c, d)$

Chase Crash Course

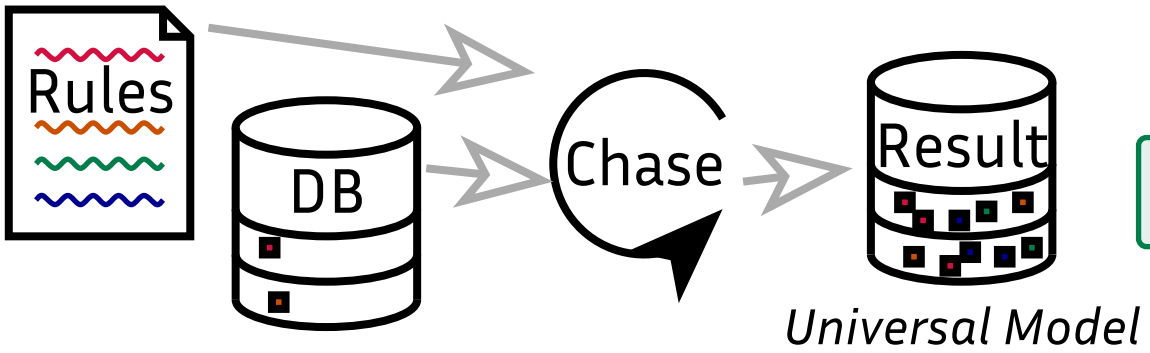


$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

DB: $\mathrm{R}(c, d)$
Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$
Step 2: $\mathrm{R}(f_z(d), f_z(f_z(d)))$, ...
Step N: ...

DB: $\mathrm{R}(c, d)$
Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

Chase Crash Course



Universal Model

$$\mathrm{R}(x, y) \to \exists z.\ \mathrm{R}(y, z) \wedge \mathrm{R}(z, y)$$

Skolem Chase

DB: $\mathrm{R}(c, d)$
Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$
Step 2: $\mathrm{R}(f_z(d), f_z(f_z(d)))$, ...
Step N: ...

Restricted Chase

DB: $\mathrm{R}(c, d)$
Step 1: $\mathrm{R}(d, f_z(d)), \mathrm{R}(f_z(d), d)$

We still use Skolem functions instead of nulls (as a naming convention).

Formal Verification – Why Bother? 🤷

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

.

.

.

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
-
-

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
-

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Practice ⚙ – *Optimized systems are complex pieces of software …*

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Practice ⚙️ – *Optimized systems are complex pieces of software …*

- … and many works around practically motivated formalizations of Datalog already exist. Whitehead (2007), Dumbrava (2016), Schlichtkrull et al. (2024), Tantow et al. (2025)

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Practice ⚙️ – *Optimized systems are complex pieces of software …*

- … and many works around practically motivated formalizations of Datalog already exist. Whitehead (2007), Dumbrava (2016), Schlichtkrull et al. (2024), Tantow et al. (2025)

Lean 🎉 – *Let's treat theorems and proofs like (open source) software.*

-
-

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Practice ⚙️ – *Optimized systems are complex pieces of software …*

- … and many works around practically motivated formalizations of Datalog already exist. Whitehead (2007), Dumbrava (2016), Schlichtkrull et al. (2024), Tantow et al. (2025)

Lean 🎉 – *Let's treat theorems and proofs like (open source) software.*

- A formal library can function as ground truth for both theory and practice.
-

Formal Verification – Why Bother? 🤷

Theory 🤓 – *Proving things is hard and intuitions might be deceiving.*

- Universal Chase Termination has been believed to be a semi-decidable problem. This is not true! Deutsch et al. (2008), Grahne and Onet (2018), Carral et al. (2025)
- Sufficient conditions for chase non-termination had subtle but crucial correctness issues in the past. Carral et al. (2017), Gerlach and Carral (2023b)
- Against intuition, prioritizing Datalog rules can jeopardize termination. Carral et al. (2022)

Practice ⚙️ – *Optimized systems are complex pieces of software …*

- … and many works around practically motivated formalizations of Datalog already exist. Whitehead (2007), Dumbrava (2016), Schlichtkrull et al. (2024), Tantow et al. (2025)

Lean 🎉 – *Let's treat theorems and proofs like (open source) software.*

- A formal library can function as ground truth for both theory and practice.
- Mathlib and CSLib receive a lot of attention. Baanen et al. (2026), Barrett et al. (2026)

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
```

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
```

Lean Crash Course ⌨

We show the proof state here.

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
```

Lean Crash Course ⌨

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
```

```
n m k : Nat
⊢ n.add (m.add k) = (n.add m).add k
```

Lean Crash Course ⌨

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
  | zero => rfl -- both sides are definitionally equal
```

```
n m : Nat
⊢ n.add (m.add 0) = (n.add m).add 0
```

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
  | zero => rfl -- both sides are definitionally equal
  | succ k ih => -- after unfolding the definition, applying the ih closes the goal
```

```
n m k : Nat
ih : n.add (m.add k) = (n.add m).add k
⊢ n.add (m.add k.succ) = (n.add m).add k.succ
```

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
  | zero => rfl -- both sides are definitionally equal
  | succ k ih => -- after unfolding the definition, applying the ih closes the goal
    simp only [Nat'.add]
```

```
n m k : Nat
ih : n.add (m.add k) = (n.add m).add k
⊢ (n.add (m.add k)).succ =
  ((n.add m).add k).succ
```

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
  | zero => rfl -- both sides are definitionally equal
  | succ k ih => -- after unfolding the definition, applying the ih closes the goal
    simp only [Nat'.add]
    rw [ih]
```

Goals accomplished 🎉

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k with
  | zero => rfl -- both sides are definitionally equal
  | succ k ih => -- after unfolding the definition, applying the ih closes the goal
    simp only [Nat'.add]
    rw [ih]
```

Goals accomplished 🎉

Lean Crash Course ⌨️

```
-- Let's (re)define the naturals ...
inductive Nat' where
| zero : Nat'
| succ : Nat' -> Nat'
-- ... addition ...
def Nat'.add (n : Nat') : Nat' -> Nat'
| .zero => n
| .succ m => (n.add m).succ
-- ... and prove that add is associative.
theorem Nat'.add_assoc (n m k : Nat') : n.add (m.add k) = (n.add m).add k := by
  induction k <;> grind [add] -- or simply use grind in all cases
```

Goals accomplished 🎉

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\ d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$
Step 2: $\mathrm{R}(f_z(d),\ f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\ y) \rightarrow \exists z.\ \mathrm{R}(y,\ z) \land \mathrm{R}(z,\ y)$$

DB: $\mathrm{R}(c,\ d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\ d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$
Step 2: $\mathrm{R}(f_z(d),\ f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\ y) \rightarrow \exists z.\ \mathrm{R}(y,\ z) \wedge \mathrm{R}(z,\ y)$$

DB: $\mathrm{R}(c,\ d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

```
-- Skolem/Restricted behavior can be generalized to checking "obs"
structure Chase (obs : ObsolescenseCondition) (rs : RuleSet) where
```

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\ d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$
Step 2: $\mathrm{R}(f_z(d),\ f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\ y) \rightarrow \exists z.\ \mathrm{R}(y,\ z) \land \mathrm{R}(z,\ y)$$

DB: $\mathrm{R}(c,\ d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

```
-- Skolem/Restricted behavior can be generalized to checking "obs"
structure Chase (obs : ObsolescenseCondition) (rs : RuleSet) where
  seq : List (ChaseNode obs rs) -- does not work...
  <conditions>
```

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\ d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$
Step 2: $\mathrm{R}(f_z(d),\ f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\ y) \rightarrow \exists z.\ \mathrm{R}(y,\ z) \wedge \mathrm{R}(z,\ y)$$

DB: $\mathrm{R}(c,\ d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

```
-- Skolem/Restricted behavior can be generalized to checking "obs"
structure Chase (obs : ObsolescenseCondition) (rs : RuleSet) where
  seq : Nat -> (ChaseNode obs rs) -- still does not work...
  <conditions>
```

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\, d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\, f_z(d)), \mathrm{R}(f_z(d),\, d)$
Step 2: $\mathrm{R}(f_z(d),\, f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\, y) \rightarrow \exists z.\, \mathrm{R}(y,\, z) \wedge \mathrm{R}(z,\, y)$$

DB: $\mathrm{R}(c,\, d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\, f_z(d)), \mathrm{R}(f_z(d),\, d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

```
-- Skolem/Restricted behavior can be generalized to checking "obs"
structure Chase (obs : ObsolescenseCondition) (rs : RuleSet) where
  seq : Nat -> Option (ChaseNode obs rs) -- getting somewhere!
  <conditions>
```

Cut to the Chase! 📏

DB: $\mathrm{R}(c,\ d)$ *(Skolem)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$
Step 2: $\mathrm{R}(f_z(d),\ f_z(f_z(d)))$, ...
Step N: ...

$$\mathrm{R}(x,\ y) \rightarrow \exists z.\ \mathrm{R}(y,\ z) \land \mathrm{R}(z,\ y)$$

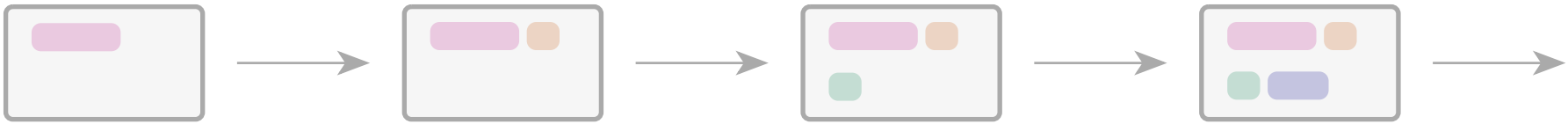
DB: $\mathrm{R}(c,\ d)$ *(Restricted)*
Step 1: $\mathrm{R}(d,\ f_z(d)), \mathrm{R}(f_z(d),\ d)$

How do we capture the differences (Skolem/Restricted and Finite/Infinite)?

```
-- Skolem/Restricted behavior can be generalized to checking "obs"
structure Chase (obs : ObsolescenseCondition) (rs : RuleSet) where
  seq : PossiblyInfiniteList (ChaseNode obs rs)
  <conditions>
```

Basic ingredients of our framework 🍳

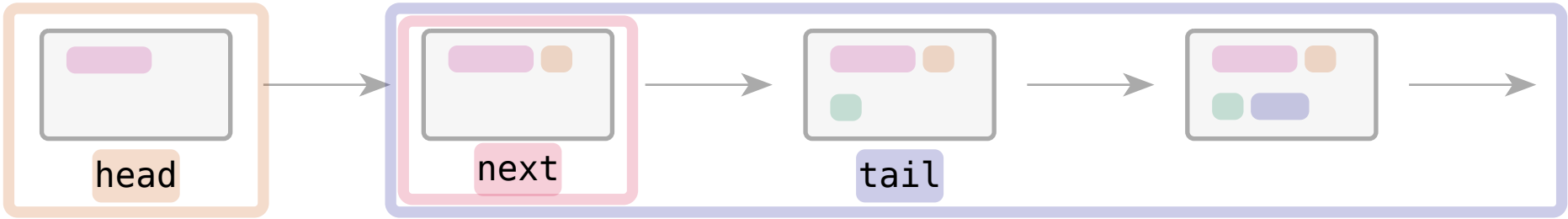
We need to be able to talk about the chase to state theorems in the first place.



1.
2.
3.
4.

Basic ingredients of our framework 🍳

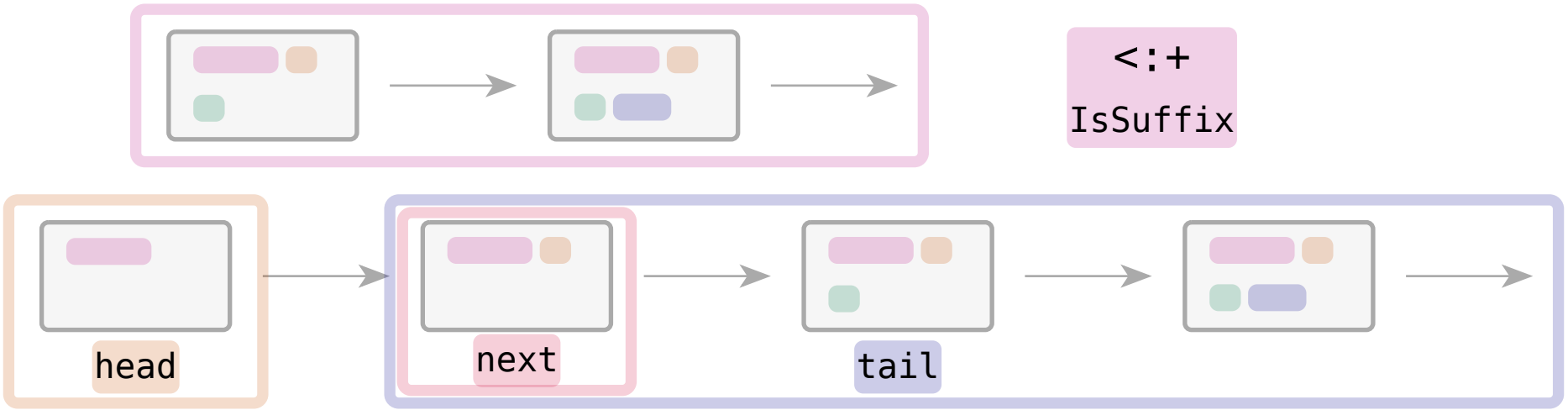
We need to be able to talk about the chase to state theorems in the first place.



1. Basic (coinductive) accessors: `head`, `next` and `tail`
2.
3.
4.

Basic ingredients of our framework 🍳

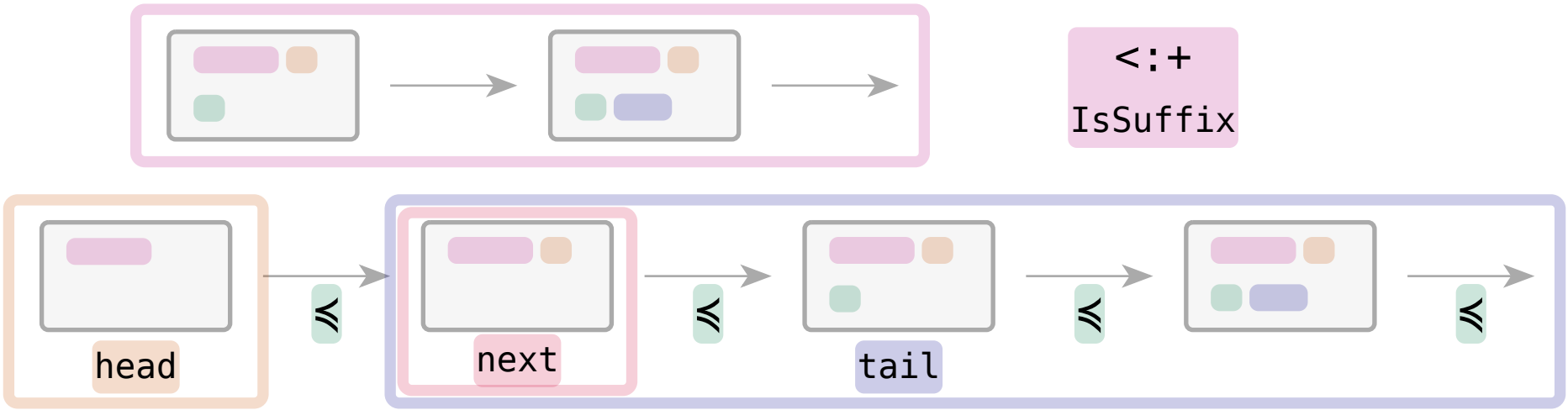
We need to be able to talk about the chase to state theorems in the first place.



1. Basic (coinductive) accessors: `head`, `next` and `tail`
2. A suffix relation `<:+` on chase sequences.
3.
4.

Basic ingredients of our framework 🍳

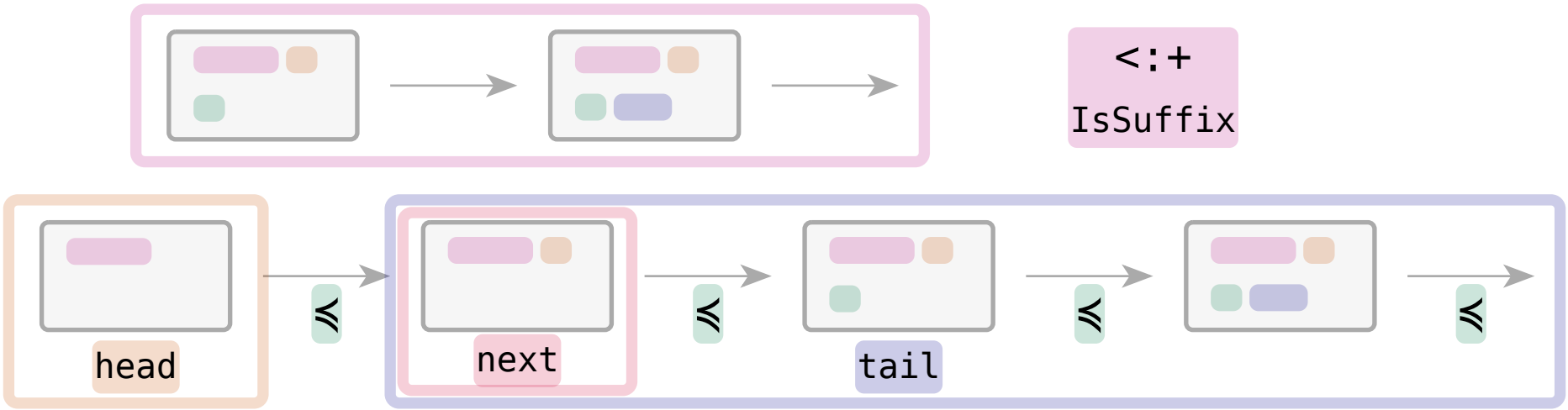
We need to be able to talk about the chase to state theorems in the first place.



1. Basic (coinductive) accessors: `head`, `next` and `tail`
2. A suffix relation `<:+` on chase sequences.
3. A predecessor relation $\preccurlyeq$ on nodes in a single chase sequence.
4.

Basic ingredients of our framework 🔍

We need to be able to talk about the chase to state theorems in the first place.



1. Basic (coinductive) accessors: `head`, `next` and `tail`
2. A suffix relation `<:+` on chase sequences.
3. A predecessor relation $\preccurlyeq$ on nodes in a single chase sequence.
4. An induction principle for nodes in the chase (next slide).

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.

Interestingly, the coinductive nature of our chase definition comes in handy.

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.
Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.

Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.

Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
  (head : motive ⟨cd.head, cd.head_mem⟩) -- base case holds
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.

Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
  (head : motive ⟨cd.head, cd.head_mem⟩) -- base case holds
  (step : ∀ (cd2 : Chase obs rules),  -- induction step holds
    (suffix : cd2 <:+ cd) ->
    motive ⟨cd2.head, ...⟩ -> -- this is the induction hypothesis
    ∀ next, (next_mem : next ∈ cd2.next) ->
    motive ⟨next, ...⟩)
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.
Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
  (head : motive ⟨cd.head, cd.head_mem⟩) -- base case holds
  (step : ∀ (cd2 : Chase obs rules),  -- induction step holds
    (suffix : cd2 <:+ cd) ->
    motive ⟨cd2.head, ...⟩ -> -- this is the induction hypothesis
    ∀ next, (next_mem : next ∈ cd2.next) ->
    motive ⟨next, ...⟩)
  (node : Node cd) : -- base and step hold, so for each node ...
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.
Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
  (head : motive ⟨cd.head, cd.head_mem⟩) -- base case holds
  (step : ∀ (cd2 : Chase obs rules),  -- induction step holds
    (suffix : cd2 <:+ cd) ->
    motive ⟨cd2.head, ...⟩ -> -- this is the induction hypothesis
    ∀ next, (next_mem : next ∈ cd2.next) ->
    motive ⟨next, ...⟩)
  (node : Node cd) : -- base and step hold, so for each node ...
  motive node := by sorry -- ... the motive holds
```

Defining our own induction principle 👀

We can define induction principles even on non-inductive types.

Interestingly, the coinductive nature of our chase definition comes in handy.

```
theorem mem_rec {cd : Chase obs rules}
  {motive : Node cd -> Prop} -- some possible property of a Node
  (head : motive ⟨cd.head, cd.head_mem⟩) -- base case holds
  (step : ∀ (cd2 : Chase obs rules),  -- induction step holds
    (suffix : cd2 <:+ cd) ->
    motive ⟨cd2.head, ...⟩ -> -- this is the induction hypothesis
    ∀ next, (next_mem : next ∈ cd2.next) ->
    motive ⟨next, ...⟩)
  (node : Node cd) : -- base and step hold, so for each node ...
  motive node := by sorry -- ... the motive holds
```

Let's prove something simple 🔍

Consider a chase node n. For each following node m, we have $n \subseteq m$.

Let's prove something simple 🔍

Consider a chase node n. For each following node m, we have $n \subseteq m$.

❓ Does it suffice to prove the following: *Each chase node is a superset of the first node.*

Let's prove something simple 🔍

Consider a chase node n. For each following node m, we have $n \subseteq m$.

❓ Does it suffice to prove the following: *Each chase node is a superset of the first node.*

Not if the chase is forced to start on a Database! If we relax this (as we did), then yes. Because then, the `tail` (and thereby every suffix) of a `Chase` is again a `Chase`.

Let's prove something simple 🔍

Consider a chase node n. For each following node m, we have $n \subseteq m$.

❓ Does it suffice to prove the following: *Each chase node is a superset of the first node.*

Not if the chase is forced to start on a Database! If we relax this (as we did), then yes. Because then, the `tail` (and thereby every suffix) of a `Chase` is again a `Chase`.

```
theorem facts_node_subset_every_mem :
    ∀ node ∈ cd, cd.head.facts ⊆ node.facts := by
  intro node node_mem
  let node' : cd.Node := ⟨node, node_mem⟩
  show cd.head.facts ⊆ node'.val.facts
  induction node' using mem_rec with
  | head => apply Set.subset_refl
  | step cd2 suffix subset next next_eq =>
    apply Set.subset_trans subset
    rw [cd2.facts_next next_eq]
    apply Set.subset_union_of_subset_left
    apply Set.subset_refl
```

```
theorem facts_node_subset_of_prec :
    node1 ≼ node2 ->
    node1.val.facts ⊆ node2.val.facts := by
  rintro ⟨cd2, suffix, head_eq, mem⟩
  rw [← head_eq]
  apply facts_node_subset_every_mem
  exact mem
```

Let's prove something simple 🔍

```
theorem facts_node_subset_every_mem :
    ∀ node ∈ cd, cd.head.facts ⊆ node.facts := by
  intro node node_mem
  let node' : cd.Node := ⟨node, node_mem⟩
  show cd.head.facts ⊆ node'.val.facts
  induction node' using mem_rec with
  | head => apply Set.subset_refl
  | step cd2 suffix subset next next_eq =>
    apply Set.subset_trans subset
    rw [cd2.facts_next next_eq]
    apply Set.subset_union_of_subset_left
    apply Set.subset_refl
```

Results that are already part of the library 📋

1. The chase yields a universal model. (Deutsch et al. 2008)
   - Involves creating an sequence of infinitely many homomorphisms.
   - `cb.result.universallyModelsKb kb`
2.
3.

Results that are already part of the library 📋

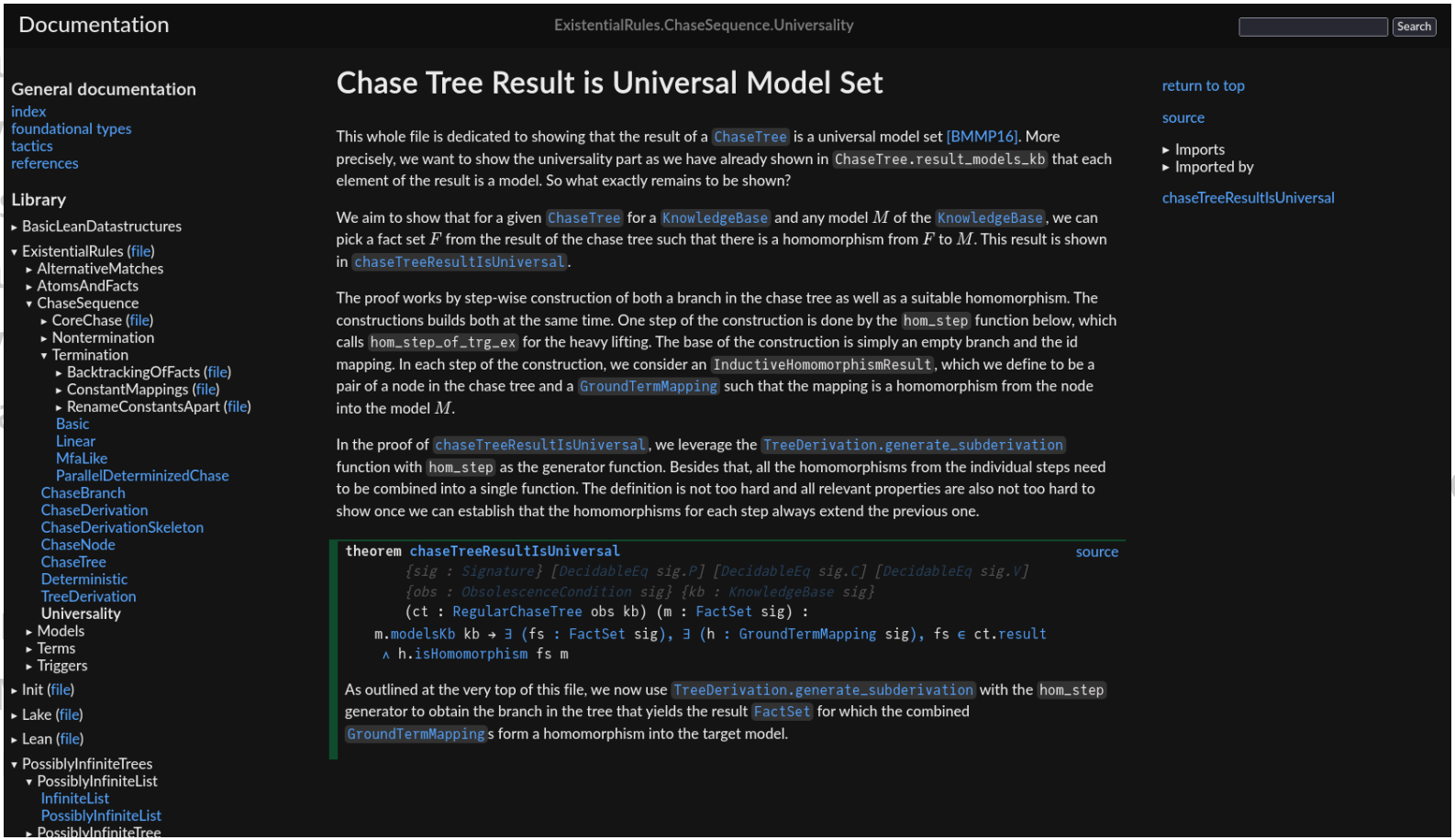
1. The chase yields a universal model. (Deutsch et al. 2008)
   - Involves creating an sequence of infinitely many homomorphisms.
   - `cb.result.universallyModelsKb kb`
2. The chase without alternative matches yields a core. (Krötzsch 2020)
   - Involves machinery similar to 1 (for non-obvious reasons).
   - `¬cb.has_alt_match → cb.result.isStrongCore`
3.

Results that are already part of the library 📋

1. The chase yields a universal model. (Deutsch et al. 2008)
   - Involves creating an sequence of infinitely many homomorphisms.
   - `cb.result.universallyModelsKb kb`
2. The chase without alternative matches yields a core. (Krötzsch 2020)
   - Involves machinery similar to 1 (for non-obvious reasons).
   - `¬cb.has_alt_match → cb.result.isStrongCore`
3. Each chase an every rule set that fulfills an MFA-like condition terminates; generalizing (Grau et al. 2013, Carral et al. 2017, Gerlach and Carral 2023a)
   - Termination conditions are generic over an `ObsolescenceCondition`.
   - ```
     rs.isMfa rs_finite (BlockingObsolescence obs rs) →
     rs.terminates obs (RegularChaseNode obs rs)
     ```
     ```

Results that are already part of the library 📋

1. The cha
   - Involv
   - cb.re
2. The cha
   - Involv
   - ¬cb.ha
3. Each ch … es;
   general
   - Termi
   - rs.isH
     rs.te



Lessons Learned & Outlook 🔭

Lessons Learned & Outlook 🔭

Invest in Frameworks 🚀 – *Think about what API you would expect as user.*

- Go bottom up and have sensible mini-libraries.
- This should involve (a lot of) obvious theorems (many of them provable via `rfl`).
- Dare to do large refactorings/rewrites, especially in the beginning.

Lessons Learned & Outlook 🔭

Invest in Frameworks 🚀 – *Think about what API you would expect as user.*

- Go bottom up and have sensible mini-libraries.
- This should involve (a lot of) obvious theorems (many of them provable via `rfl`).
- Dare to do large refactorings/rewrites, especially in the beginning.

What's next? 🔮

- Proving correctness of RPC. (Gerlach and Carral 2023b)
- Core chase formalization. (Deutsch et al. 2008)
- An executable (naive) chase procedure.
- *Your contribution?*

Lessons Learned & Outlook 🔭

Invest in Frameworks 🚀 – *Think about what API you would expect as user.*

- Go bottom up and have sensible mini-libraries.
- This should involve (a lot of) obvious theorems (many of them provable via `rfl`).
- Dare to do large refactorings/rewrites, especially in the beginning.

What's next? 🔮

- Proving correctness of RPC. (Gerlach and Carral 2023b)
- Core chase formalization. (Deutsch et al. 2008)
- An executable (naive) chase procedure.
- *Your contribution?*

Let's start to formally represent our knowledge about knowledge representation!

Lessons Learned & Outlook 🔭

Invest in Frameworks 🚀 – *Think about what API you would expect as user.*

- Go bottom up and have sensible mini-libraries.
- This should involve (a lot of) obvious theorems (many of them provable via `rfl`).
- Dare to do large refactorings/rewrites, especially in the beginning.

What's next? 🔮

- Proving correctness of RPC. (Gerlach and Carral 2023b)
- Core chase formalization. (Deutsch et al. 2008)
- An executable (naive) chase procedure.
- *Your contribution?*

Visit https://dmfa.dev/lean 🤓

Let's start to formally represent our knowledge about knowledge representation!

References

Baanen A, Ballard MR, Commelin J, et al (2026) Growing Mathlib: maintenance of a large scale mathematical library. In: Paiva V de, Koepke P (eds) Intelligent Computer Mathematics. Springer Nature Switzerland, Cham, pp 51–70

Barrett C, Chaudhuri S, Montesi F, et al (2026) CSLib: The Lean Computer Science Library. https://arxiv.org/abs/2602.04846

Carral D, Dragoste I, Krötzsch M (2017) Restricted Chase (Non)Termination for Existential Rules with Disjunctions. In: Sierra C (ed) Proceedings of the Twenty-Sixth International Joint Conference on Artificial

References

Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. ijcai.org, pp 922–928

Carral D, Gerlach L, Larroque L, Thomazo M (2025) Restricted Chase Termination: You Want More than Fairness. Proc ACM Manag Data 3:109:1–109:17. https://doi.org/10.1145/3725246

Carral D, Larroque L, Mugnier M-L, Thomazo M (2022) Normalisations of Existential Rules: Not so Innocuous!. In: Kern-Isberner G, Lakemeyer G, Meyer T (eds) Proceedings of the 19th International Conference on

References

Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022

Deutsch A, Nash A, Remmel JB (2008) The chase revisited. In: Lenzerini M, Lembo D (eds) Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada. ACM, pp 149–158

Dumbrava S-G (2016) Formalisation en Coq de Bases de Données Relationnelles et Déductives -et Mécanisation de Datalog. (A Coq

References

Formalization of Relational and Deductive Databases -and a Mechanizations of Datalog). Doctoral dissertation

Fagin R, Kolaitis PG, Miller RJ, Popa L (2005) Data exchange: semantics and query answering. Theor Comput Sci 336:89–124. https://doi.org/10.1016/J.TCS.2004.10.033

Gerlach L, Carral D (2023b) Do Repeat Yourself: Understanding Sufficient Conditions for Restricted Chase Non-Termination. In: Marquis P, Son TC, Kern-Isberner G (eds) Proceedings of the 20th International Conference

References

on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023. pp 301–310

Gerlach L, Carral D (2023a) General Acyclicity and Cyclicity Notions for the Disjunctive Skolem Chase. In: Williams B, Chen Y, Neville J (eds) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023. AAAI Press, pp 6372–6379

References

Gerlach L, Carral D, Hecher M (2024) Finite Groundings for ASP with Functions: A Journey through Consistency. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024. ijcai.org, pp 3386–3394

Gerlach L, Larroque L, Marcinkowski J, Ostropolski-Nalewaja P (2025) About the Multi-Head Linear Restricted Chase Termination. In: Ortiz M, Wassermann R, Schaub T (eds) Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, KR 2025, Melbourne, Australia, November 1-17, 2025

References

Gogacz T, Marcinkowski J (2014) All-Instances Termination of Chase is Undecidable. In: Esparza J, Fraigniaud P, Husfeldt T, Koutsoupias E (eds) Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II. Springer, pp 293–304

Gogacz T, Marcinkowski J, Pieris A (2023) Uniform Restricted Chase Termination. SIAM J Comput 52:641–683. https://doi.org/10.1137/20M1377035

References

Grahne G, Onet A (2018) Anatomy of the Chase. Fundam Informaticae 157:221–270. https://doi.org/10.3233/FI-2018-1627

Grau BC, Horrocks I, Krötzsch M, et al (2013) Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. J Artif Intell Res 47:741–808. https://doi.org/10.1613/JAIR.3949

Harel D (1986) Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. J ACM 33:224–248. https://doi.org/10.1145/4904.4993

References

Ivliev A, Gerlach L, Meusel S, et al (2024) Nemo: Your Friendly and Versatile Rule Reasoning Toolkit. In: Marquis P, Ortiz M, Pagnucco M (eds) Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024

Krötzsch M (2020) Computing Cores for Existential Rules with the Standard Chase and ASP. In: Calvanese D, Erdem E, Thielscher M (eds) Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020. pp 603–613

References

Krötzsch M, Rudolph S (2011) Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In: Walsh T (ed) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. IJCAI/AAAI, pp 963–968

Leclère M, Mugnier M-L, Thomazo M, Ulliana F (2019) A Single Approach to Decide Chase Termination on Linear Existential Rules. In: Barceló P, Calautti M (eds) 22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, p 18:1–18:19

References

Schlichtkrull A, Rydhof Hansen R, Nielson F (2024) Isabelle-verified correctness of Datalog programs for program analysis. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC 2024). ACM, pp 1731–1734

Tantow J, Gerlach L, Mennicke S, Krötzsch M (2025) Verifying Datalog Reasoning with Lean. In: Forster Y, Keller C (eds) 16th International Conference on Interactive Theorem Proving, ITP 2025, Reykjavik, Iceland, September 28 - October 1, 2025. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, p 36:1–36:19

References

Whitehead N (2007) A Certified Distributed Security Logic for Authorizing Code. In: Altenkirch T, McBride C (eds) Types for Proofs and Programs. Springer, pp 253–268