Global Caching in Tableaux for Description Logic with Definite Descriptions

Michał Zawidzki
✉ michal.zawidzki@filhist.uni.lodz.pl

ExtenDD seminar, November 4, 2026

What do definite descriptions add to plain $\mathscr{ALC}$?

Syntax

$$C ::= A \mid \neg C \mid C \sqcap C \mid \exists r.C \mid \{\iota C\} \mid \iota C.C$$

with $\sqcup$ and $\forall r$ as standard abbreviations.

What do definite descriptions add to plain $\mathcal{ALC}$?

Syntax

$$C ::= A \mid \neg C \mid C \sqcap C \mid \exists r.C \mid \{\iota C\} \mid \iota C.C$$

with $\sqcup$ and $\forall r$ as standard abbreviations.

Semantics

$$\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle,$$

where:

- $\Delta^{\mathcal{I}} \neq \varnothing$
- $\cdot^{\mathcal{I}} : \mathcal{A} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}}), \quad \cdot^{\mathcal{I}} : \mathcal{R} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$

What do definite descriptions add to plain $\mathcal{ALC}$?

Syntax

$$C ::= A \mid \neg C \mid C \sqcap C \mid \exists r.C \mid \{\iota C\} \mid \iota C.C$$

with $\sqcup$ and $\forall r$ as standard abbreviations.

Semantics

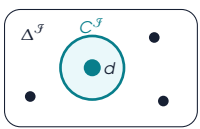
$$\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle,$$

where:

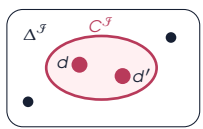
- $\Delta^{\mathcal{I}} \neq \varnothing$
- $\cdot^{\mathcal{I}} : \mathcal{A} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}}), \quad \cdot^{\mathcal{I}} : \mathcal{R} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$

Local descriptions

$$\{\iota C\}^{\mathcal{I}} = \begin{cases} \{d\}, & C^{\mathcal{I}} = \{d\}, \\ \varnothing, & \text{otherwise.} \end{cases}$$



$(\{\iota C\})^{\mathcal{I}} = C^{\mathcal{I}} = \{d\}$



$(\{\iota C\})^{\mathcal{I}} = \varnothing$

What do definite descriptions add to plain $\mathcal{ALC}$?

Syntax

$$C ::= A \mid \neg C \mid C \sqcap C \mid \exists r.C \mid \{\iota C\} \mid \iota C.C$$

with $\sqcup$ and $\forall r$ as standard abbreviations.

Semantics

$$\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle,$$

where:

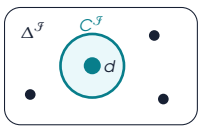
- $\Delta^{\mathcal{I}} \neq \varnothing$
- $\cdot^{\mathcal{I}} : \mathcal{A} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}}), \quad \cdot^{\mathcal{I}} : \mathcal{R} \longrightarrow \mathcal{P}(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$

Local descriptions

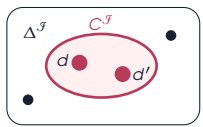
$$\{\iota C\}^{\mathcal{I}} = \begin{cases} \{d\}, & C^{\mathcal{I}} = \{d\}, \\ \varnothing, & \text{otherwise.} \end{cases}$$

Global descriptions

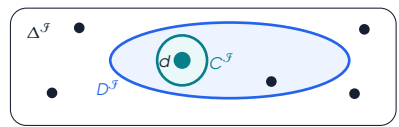
$$(\iota C.D)^{\mathcal{I}} = \begin{cases} \Delta^{\mathcal{I}}, & C^{\mathcal{I}} = \{d\} \subseteq D^{\mathcal{I}}, \\ \varnothing, & \text{otherwise.} \end{cases}$$



$(\{\iota C\})^{\mathcal{I}} = C^{\mathcal{I}} = \{d\}$



$(\{\iota C\})^{\mathcal{I}} = \varnothing$



$|C^{\mathcal{I}}| = 1$ and $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ imply $(\iota C.D)^{\mathcal{I}} = \Delta^{\mathcal{I}}$

The tableau calculus: the DD rules

Rules for local DDs

$$\frac{a : \{\iota C\}}{a : C}$$

(a DD-satisfying individual satisfies the DD-concept)

$$\frac{a : \{\iota C\},\ b : C,\ c : C,\ b : D}{c : D}$$

(no two distinct individuals satisfy a DD-concept)

$$\frac{a : \neg\{\iota C\}}{a : \neg C \mid \begin{array}{l} b : C,\ b : A_C, \\ a : C,\ a : \neg A_C \end{array}}$$

(if an individual does not satisfy a local DD, either it does not satisfy the DD-concept, or another individual also satisfies this DD-concept)

The tableau calculus: the DD rules

Rules for local DDs

$$\frac{a : \{\iota C\}}{a : C}$$

(a DD-satisfying individual satisfies the DD-concept)

$$\frac{a : \{\iota C\},\ b : C,\ c : C,\ b : D}{c : D}$$

(no two distinct individuals satisfy a DD-concept)

$$\frac{a : \neg\{\iota C\}}{a : \neg C \mid \begin{array}{l} b : C,\ b : A_C, \\ a : C,\ a : \neg A_C \end{array}}$$

(if an individual does not satisfy a local DD, either it does not satisfy the DD-concept, or another individual also satisfies this DD-concept)

Rules for global DDs

$$\frac{a : \iota C.D}{b : \{\iota C\},\ b : D}$$

(satisfaction of a global DD requires a witness)

$$\frac{a : \neg\iota C.D,\ b : E}{b : \neg\{\iota C\} \mid b : \neg D}$$

(non-satisfaction of a global DD means that no individual is a witness)

The tableau calculus: the DD rules

Rules for local DDs

$$\frac{a : \{\iota C\}}{a : C}$$

(a DD-satisfying individual satisfies the DD-concept)

$$\frac{a : \{\iota C\},\ b : C,\ c : C,\ b : D}{c : D}$$

(no two distinct individuals satisfy a DD-concept)

$$\frac{a : \neg\{\iota C\}}{a : \neg C \mid \begin{array}{l} b : C,\ b : A_C, \\ a : C,\ a : \neg A_C \end{array}}$$

(if an individual does not satisfy a local DD, either it does not satisfy the DD-concept, or another individual also satisfies this DD-concept)

Rules for global DDs

$$\frac{a : \iota C.D}{b : \{\iota C\},\ b : D}$$

(satisfaction of a global DD requires a witness)

$$\frac{a : \neg\iota C.D,\ b : E}{b : \neg\{\iota C\} \mid b : \neg D}$$

(non-satisfaction of a global DD means that no individual is a witness)

Analytic cut around local DDs

$$\frac{a : \{\iota C\},\ b : D}{b : C \mid b : \neg C}$$

(each individual must be determined w.r.t. each DD-concept)

The tableau calculus: the DD rules

Rules for local DDs

$$\frac{a : \{\iota C\}}{a : C}$$

(a DD-satisfying individual satisfies the DD-concept)

$$\frac{a : \{\iota C\},\ b : C,\ c : C,\ b : D}{c : D}$$

(no two distinct individuals satisfy a DD-concept)

$$\frac{a : \neg\{\iota C\}}{a : \neg C \mid \begin{array}{l} b : C,\ b : A_C, \\ a : C,\ a : \neg A_C \end{array}}$$

(if an individual does not satisfy a local DD, either it does not satisfy the DD-concept, or another individual also satisfies this DD-concept)

Remark: A tableau node is a **set of labelled concepts and role assertions**. A tableau is a **derivation tree** (not an $\mathcal{ALC}$(DD)-interpretation).

Rules for global DDs

$$\frac{a : \iota C.D}{b : \{\iota C\},\ b : D}$$

(satisfaction of a global DD requires a witness)

$$\frac{a : \neg\iota C.D,\ b : E}{b : \neg\{\iota C\} \mid b : \neg D}$$

(non-satisfaction of a global DD means that no individual is a witness)

Analytic cut around local DDs

$$\frac{a : \{\iota C\},\ b : D}{b : C \mid b : \neg C}$$

(each individual must be determined w.r.t. each DD-concept)

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.

$a : \exists r.C$
$a : \forall r.D$

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.



Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.

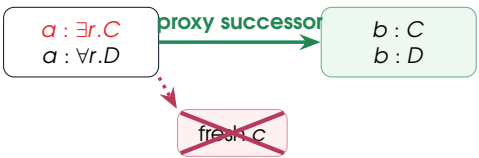


No new individual is needed on this branch.

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.

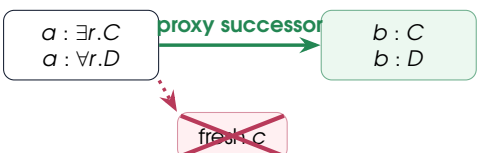


No new individual is needed on this branch.

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.



No new individual is needed on this branch.

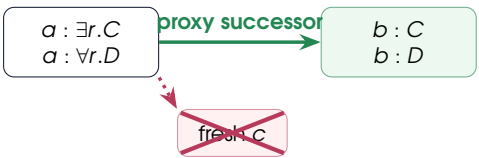
Blocking negative descriptions

The right branch of the rule for $\neg\{\iota C\}$ creates two C-witnesses with opposite marker literals.

Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.



No new individual is needed on this branch.

Blocking negative descriptions

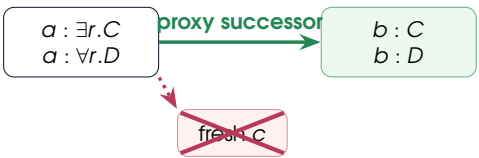
The right branch of the rule for $\neg\{\iota C\}$ creates two C-witnesses with opposite marker literals.



Blocking: Why a branch does not grow forever

Pattern-based blocking for $\exists r.C$

We reuse an existing individual that satisfies C and every universal r-successor constraint imposed at the source.



No new individual is needed on this branch.

Blocking negative descriptions

The right branch of the rule for $\neg\{\iota C\}$ creates two C-witnesses with opposite marker literals.



Once A_C occurs, later expansions for $\neg\{\iota C\}$ or $\neg\iota C.D$ are blocked: C is already known not to be singleton.

How do blocking mechanisms work

$$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$$

How do blocking mechanisms work

$$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$$

$$\mathcal{B}_0 \quad = \quad \{a \quad : \quad M\}$$

How do blocking mechanisms work

$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$

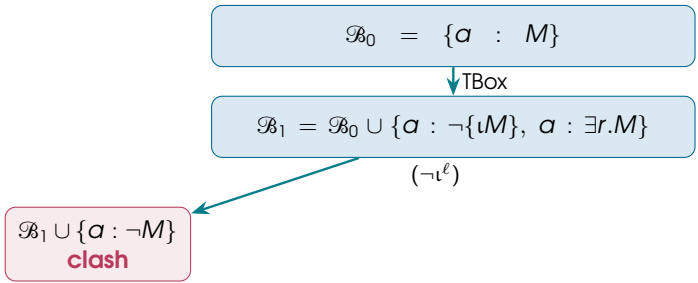
$$\mathscr{B}_0 \;=\; \{a \;:\; M\}$$

TBox

$$\mathscr{B}_1 \;=\; \mathscr{B}_0 \cup \{a \;:\; \neg\{\iota M\},\ a \;:\; \exists r.M\}$$

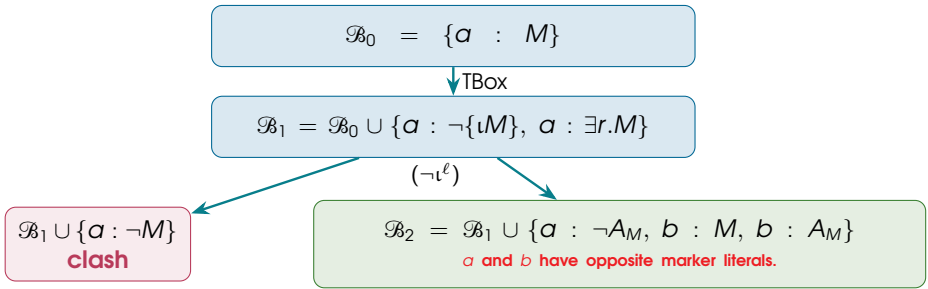
How do blocking mechanisms work

$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$



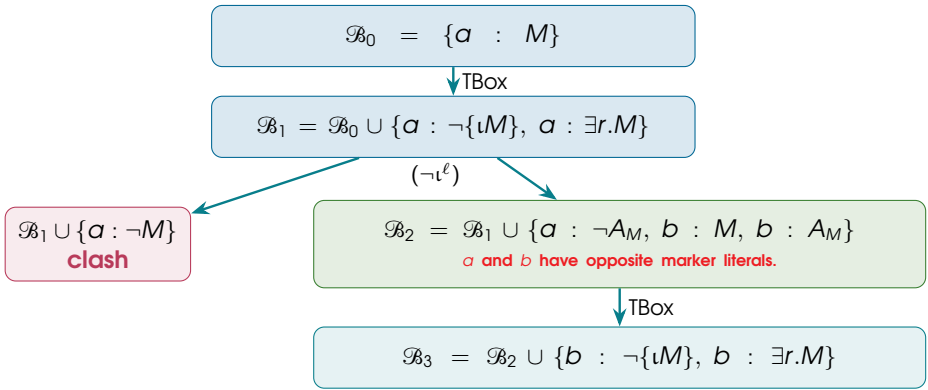
How do blocking mechanisms work

$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$



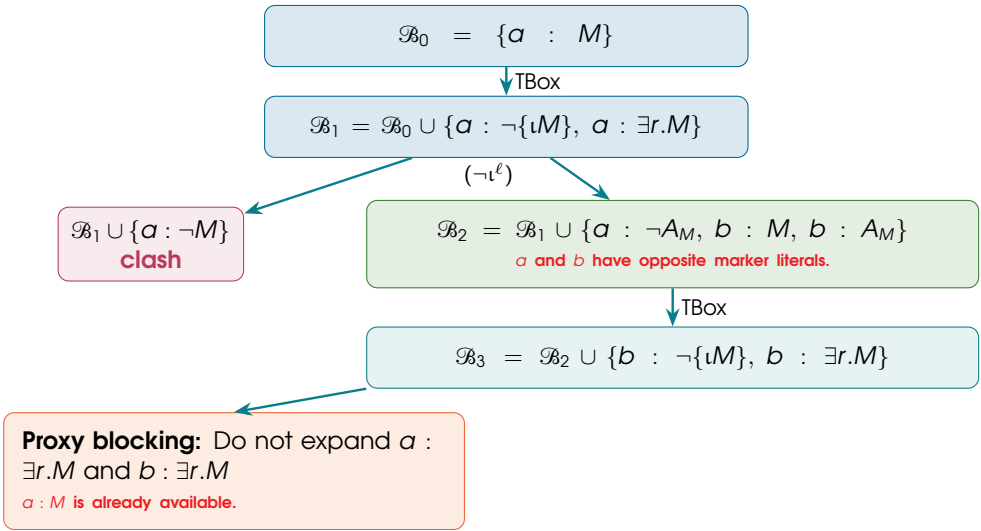
How do blocking mechanisms work

$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$

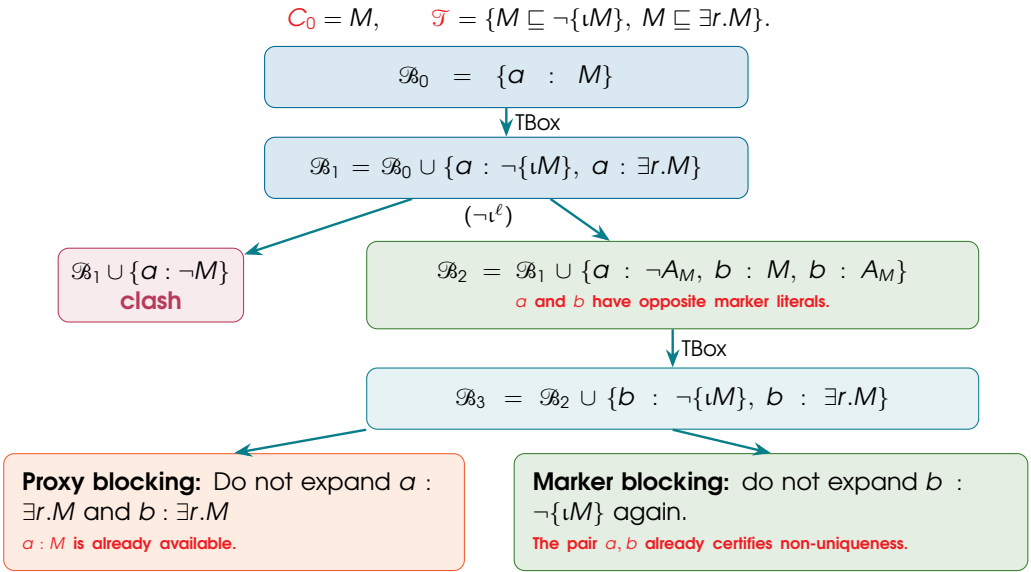


How do blocking mechanisms work

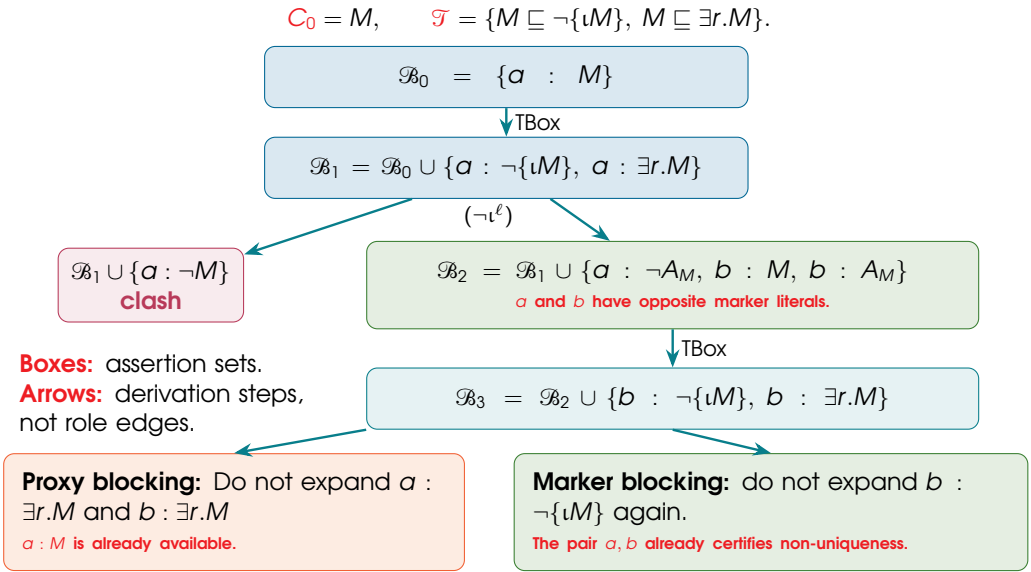
$C_0 = M, \qquad \mathcal{T} = \{M \sqsubseteq \neg\{\iota M\},\ M \sqsubseteq \exists r.M\}.$



How do blocking mechanisms work



How do blocking mechanisms work

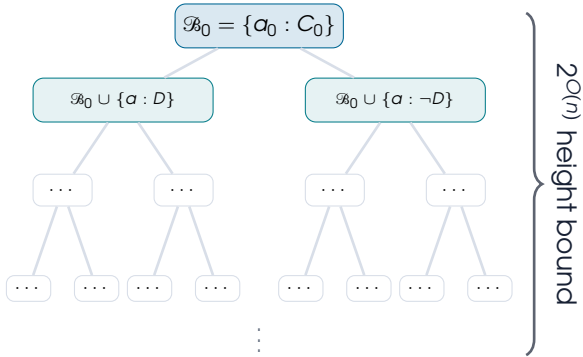


Why the derivation-tree bound is doubly exponential

- Blocking bounds the number of individuals on one branch by $2^{O(n)}$.

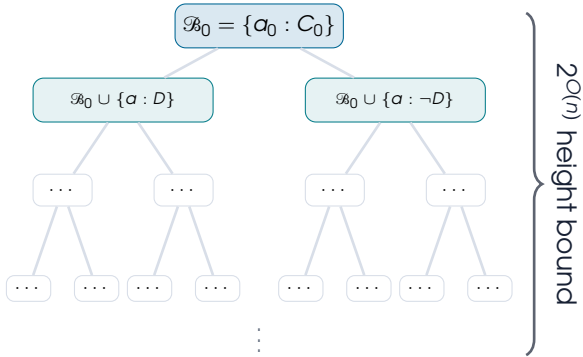
Why the derivation-tree bound is doubly exponential

- Blocking bounds the number of individuals on one branch by $2^{O(n)}$.
- Therefore the height of the **derivation tree** can be exponential.



Why the derivation-tree bound is doubly exponential

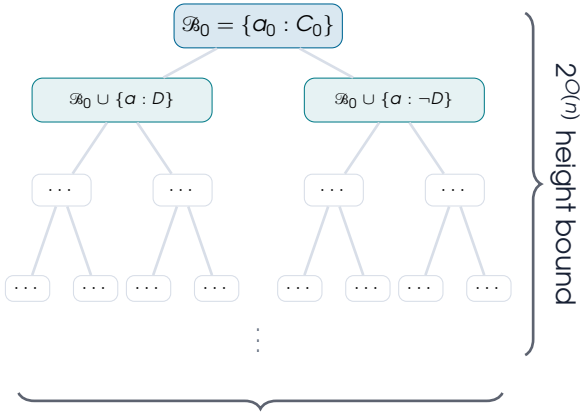
- Blocking bounds the number of individuals on one branch by $2^{O(n)}$.
- Therefore the height of the **derivation tree** can be exponential.
- **Branching rules** still **duplicate** the remaining proof search below each alternative.



Why the derivation-tree bound is doubly exponential

- Blocking bounds the number of individuals on one branch by $2^{O(n)}$.
- Therefore the height of the **derivation tree** can be exponential.
- **Branching rules** still **duplicate** the remaining proof search below each alternative.

Remark: Since nodes of a derivation tree are (sets of) labelled concepts, exponential height and binary branching allow a **doubly exponential upper bound** (of the tree, not an interpretation).



Why a branch can contain exponentially many individuals

An encoding of linear-size k-bit counter

1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Why a branch can contain exponentially many individuals

An encoding of linear-size k-bit counter

1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Blocking waits for a repeated pattern

A new bit pattern requires a new individual. Only when 0 returns can an earlier individual serve as proxy.

Why a branch can contain exponentially many individuals

An encoding of linear-size k-bit counter

1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Blocking waits for a repeated pattern

A new bit pattern requires a new individual. Only when 0 returns can an earlier individual serve as proxy.

For $k = 3$: $2^k = 2^3 = 8$ individuals are accumulated on **one derivation branch**.

Why a branch can contain exponentially many individuals

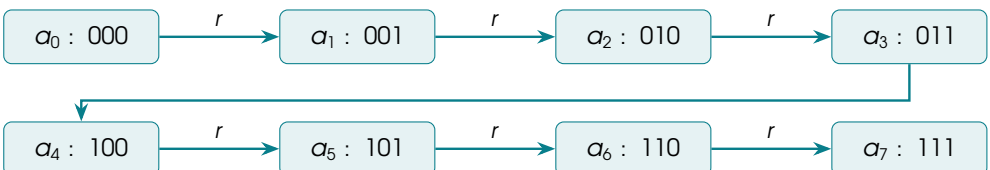
An encoding of linear-size k-bit counter

1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Blocking waits for a repeated pattern

A new bit pattern requires a new individual. Only when 0 returns can an earlier individual serve as proxy.

For $k = 3$: $2^k = 2^3 = 8$ individuals are accumulated on **one derivation branch**.



Why a branch can contain exponentially many individuals

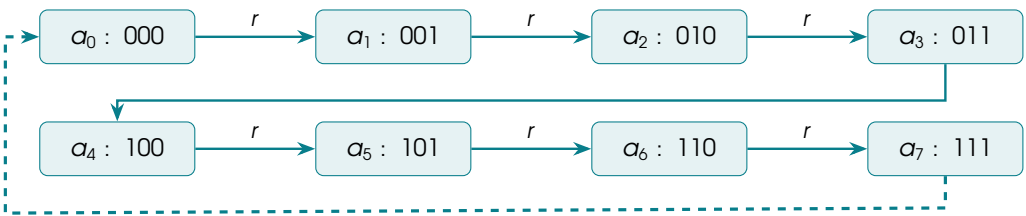
An encoding of linear-size k-bit counter

1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Blocking waits for a repeated pattern

A new bit pattern requires a new individual. Only when 0 returns can an earlier individual serve as proxy.

For $k = 3$: $2^k = 2^3 = 8$ individuals are accumulated on **one derivation branch**.



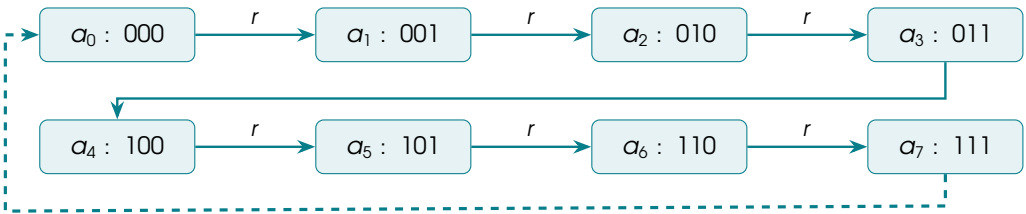
At 111, we reuse 000: no 9th individual is needed.

Why a branch can contain exponentially many individuals

An encoding of linear-size k-bit counter
1. Start at 0.
2. Every r-successor increments the bits modulo 2^k.

Blocking waits for a repeated pattern
A new bit pattern requires a new individual. Only when 0 returns can an earlier individual serve as proxy.

For $k = 3$: $2^k = 2^3 = 8$ individuals are accumulated on **one derivation branch**.



At 111, we reuse 000: no 9th individual is needed.

Remark: Number of individuals $N = 2^k = 2^{\Theta(n)}$; each individual carries only $O(n)$ closure concepts.

Branch height $O(nN)$: **the exponential factor comes from the individuals.**

A possible doubly exponential derivation

Let's extend the counter TBox by $\top \sqsubseteq P \sqcup Q$, $\top \sqsubseteq \exists s.Z$, $Z \sqsubseteq \bot$.

A possible doubly exponential derivation

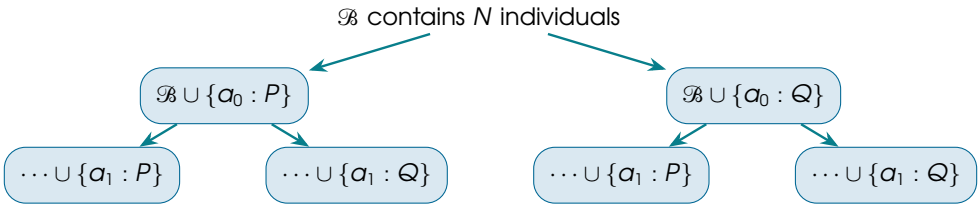
Let's extend the counter TBox by $\top \sqsubseteq P \sqcup Q$, $\top \sqsubseteq \exists s.Z$, $Z \sqsubseteq \bot$.

A poorly chosen, yet legal order: we build the $N = 2^k$ counter individuals, then branch on $P \sqcup Q$ at each, and only then fulfil an s-request.

A possible doubly exponential derivation

Let's extend the counter TBox by $\top \sqsubseteq P \sqcup Q$, $\top \sqsubseteq \exists s.Z$, $Z \sqsubseteq \bot$.

A poorly chosen, yet legal order: we build the $N = 2^k$ counter individuals, then branch on $P \sqcup Q$ at each, and only then fulfil an s-request.



Remark: The doubly exponential blow-up above is dependent on the rule-application order and is not unavoidable. The example only serves illustrating that **doubly exponential tableaux are possible**.

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \left(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\right),$$

where:

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \left(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\right),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \left(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\right),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \big(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\big),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \big(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\big),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.

The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \big(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\big),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.

Γ is frozen

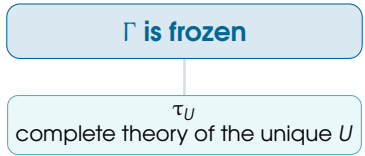
The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \big(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\big),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.



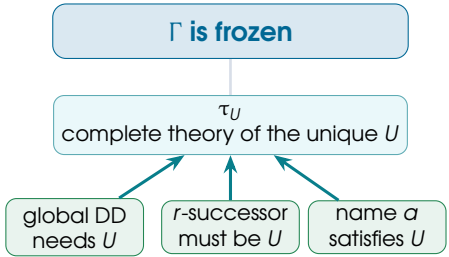
The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \left(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\right),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.



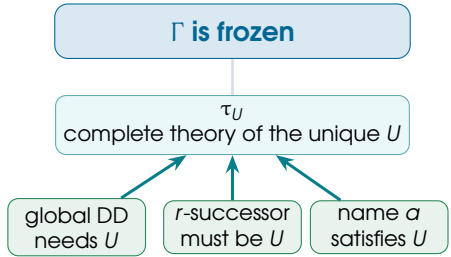
The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \left(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\right),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.



All three requests use the same complete theory.

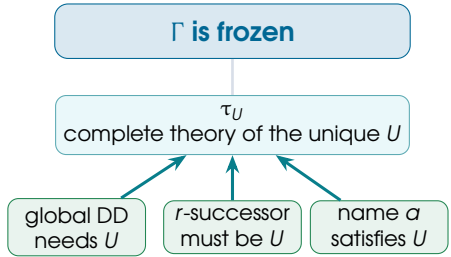
The fixed global profile Γ

Let C contain the subjects C of $\{\iota C\}$ and $\iota C.D$, including nested DDs. A **global profile** Γ is:

$$\Gamma = \big(\sigma, (\tau_C)_{\sigma(C)=1}, (\tau_a)_{a \in N_I}\big),$$

where:

- $\sigma(C) \in \{0, 1, 2\}$ records whether C is empty, singleton, or has at least two instances.
- τ_C is the **complete theory** (type) of the unique C-individual when $\sigma(C) = 1$.
- τ_a is the type assigned to each input individual name a.
- If $\sigma(C) = 2$, add A_C and $\neg A_C$ to the common closure used by every type.



All three requests use the same complete theory.

Remark: A **complete type** contains E or $\neg E$ for every closure concept; it is not yet known whether it is satisfiable.

Singleton agreement: $\sigma(C) = 1$ and $C \in t$ force $t = \tau_C$.

How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.

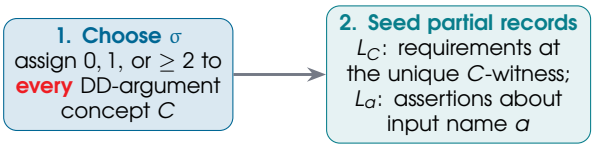
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.

1. Choose σ
assign 0, 1, or ≥ 2 to **every** DD-argument concept C

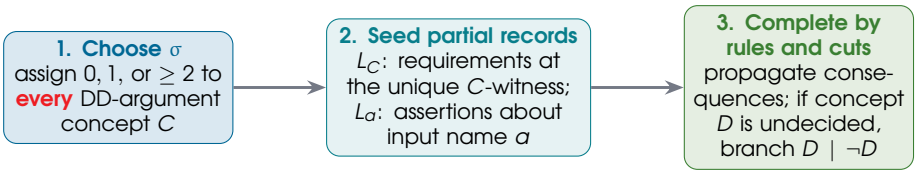
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



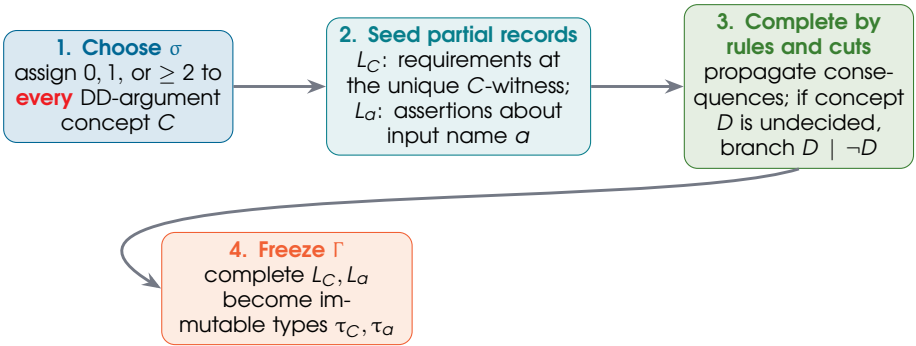
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



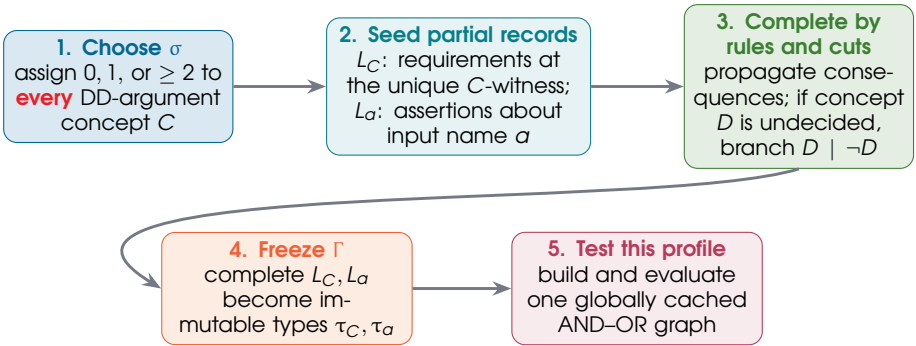
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



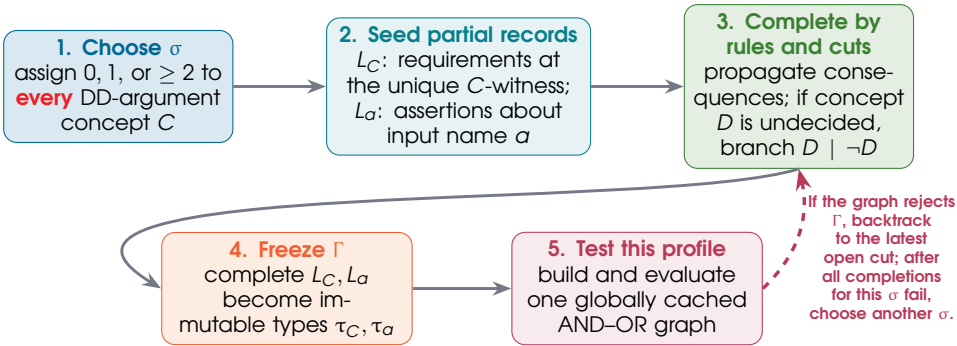
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



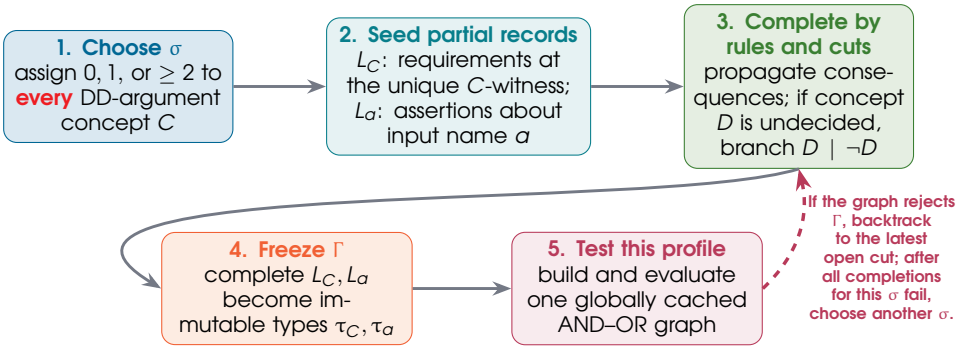
How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



How one complete profile is built and tested

Profiles are generated **one at a time**; the cached graph is built only after one profile is complete and frozen.



Complete profile Γ = numerical assignment + all designated complete types:

$$\#\Gamma \quad \leq \quad 3^{|\,\mathrm{dom}(\sigma)|}\left(2^{|\,\mathrm{cl}_\sigma\,|}\right)^{|\{C\,|\,\sigma(C)=1\}|+|N_I|} \quad = \quad 2^{O(n^2)}.$$

Running example: Determining a global profile Γ

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

Running example: Determining a global profile Γ

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

1. Choose a candidate

$(\sigma(U), \sigma(M)) \in \{0, 1, 2\}^2$.

Here: $(1, 2)$.

No $\sigma(Z)$: Z is not a DD subject.

Running example: Determining a global profile Γ

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

1. Choose a candidate

$(\sigma(U), \sigma(M)) \in \{0,1,2\}^2$.

Here: $(1,2)$.

No $\sigma(Z)$: Z is not a DD subject.

2. Seed the records

$L_U = \{U, \{\iota U\}, Q\}$

$L_a = \{U\}$.

After choosing σ: We use the rule for $\iota U.Q$ to impose Q.

We introduce the marker A_M.

Running example: Determining a global profile Γ

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

1. Choose a candidate

$(\sigma(U), \sigma(M)) \in \{0,1,2\}^2$.

Here: $(1,2)$.

No $\sigma(Z)$: Z is not a DD subject.

2. Seed the records

$L_U = \{U, \{\iota U\}, Q\}$
$L_a = \{U\}$.

After choosing σ: We use the rule for $\iota U.Q$ to impose Q.
We introduce the marker A_M.

3. Propagate

Under this σ, L_U and L_a must agree.
Both acquire Q, $\neg M$, $\neg Z$.

We use singleton propagation and the TBox rules.

Running example: Determining a global profile Γ

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

1. Choose a candidate

$(\sigma(U), \sigma(M)) \in \{0,1,2\}^2$.

Here: $(1,2)$.

No $\sigma(Z)$: Z is not a DD subject.

2. Seed the records

$L_U = \{U, \{\iota U\}, Q\}$

$L_a = \{U\}$.

After choosing σ: We use the rule for $\iota U.Q$ to impose Q.

We introduce the marker A_M.

3. Propagate

Under this σ, L_U and L_a must agree.

Both acquire Q, $\neg M$, $\neg Z$.

We use singleton propagation and the TBox rules.

Remark: A candidate is not yet a solution. We complete and freeze Γ, then expand and evaluate its graph. A numerical choice may fail only there. Early numerical pruning is optional, not used here.

Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

One candidate for $\tau_U = \tau_a$
U, Q hold; the so-far chosen concepts are:
$\neg M, \neg Z, \neg P$
$\neg\exists s.M, \neg\exists r.Z, \neg\exists r.U$.
We complete the type in the last choice:

Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

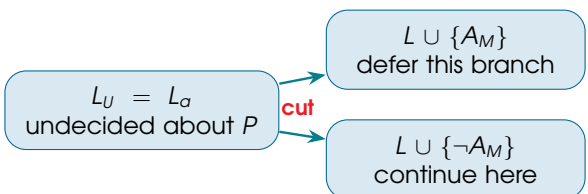
One candidate for $\tau_U = \tau_a$
U, Q hold; the so-far chosen concepts are:
$\neg M, \neg Z, \neg P$
$\neg\exists s.M, \neg\exists r.Z, \neg\exists r.U$.
We complete the type in the last choice:

$L_U = L_a$
undecided about P

Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

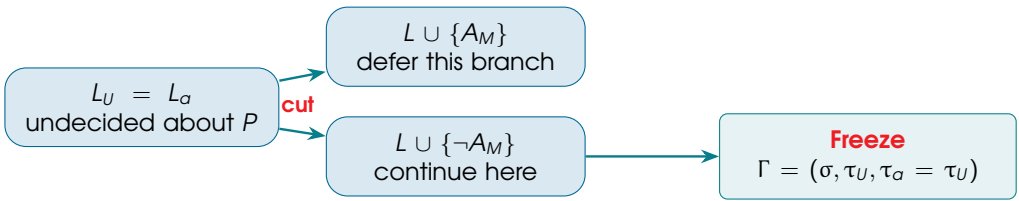
One candidate for $\tau_U = \tau_a$
U, Q hold; the so-far chosen concepts are:
$\neg M, \neg Z, \neg P$
$\neg\exists s.M, \neg\exists r.Z, \neg\exists r.U$.
We complete the type in the last choice:



Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

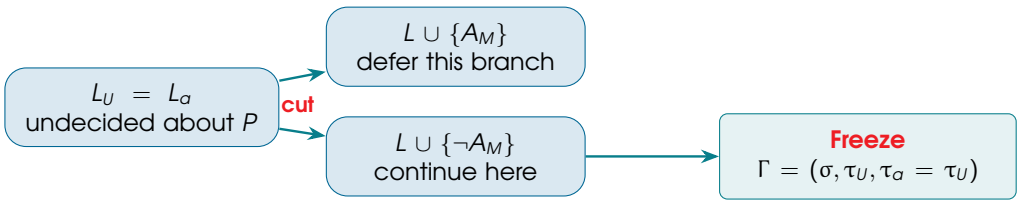
One candidate for $\tau_U = \tau_a$
U, Q hold; the so-far chosen concepts are:
$\neg M, \neg Z, \neg P$
$\neg\exists s.M, \neg\exists r.Z, \neg\exists r.U$.
We complete the type in the last choice:



Running example: Determining a global profile Γ (cont'd)

Expressions to decide
Atoms: U, M, Z, Q, P, A_M.
Existentials: $\exists s.M, \exists r.Z, \exists r.U$.
These choices determine all complex concepts and DDs.

One candidate for $\tau_U = \tau_a$
U, Q hold; the so-far chosen concepts are:
$\neg M, \neg Z, \neg P$
$\neg\exists s.M, \neg\exists r.Z, \neg\exists r.U$.
We complete the type in the last choice:



Remark: A different completion is a different profile. Upon rejection, we resume an untried cut.

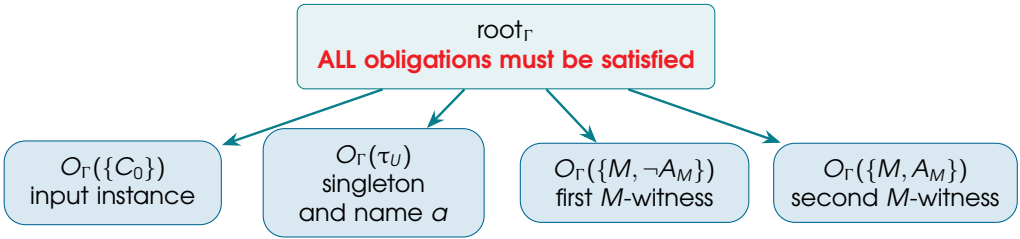
We never change the singleton type inside an already cached profile.

Running example: Root obligations

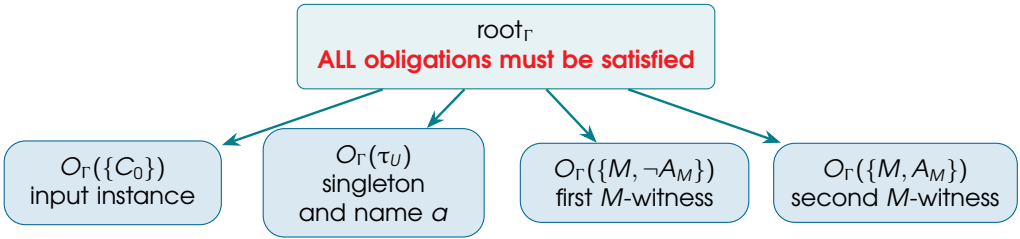
root_Γ

ALL obligations must be satisfied

Running example: Root obligations



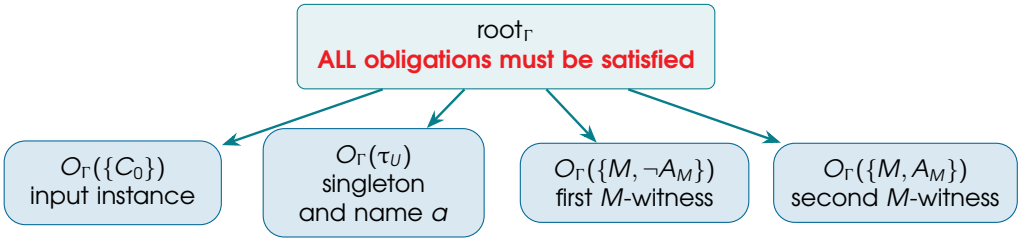
Running example: Root obligations



A profile assumption is not yet a model fact.

E.g., $\sigma(M) = 2$ creates two existence tasks. Their complete witness types are found **during graph expansion**, not stored in Γ.

Running example: Root obligations



A profile assumption is not yet a model fact.
E.g., $\sigma(M) = 2$ creates two existence tasks. Their complete witness types are found **during graph expansion**, not stored in Γ.

Opposite marker values force distinct witnesses.
Their types may be reused elsewhere. The edges from H_Γ are proof dependencies, **not role edges**.

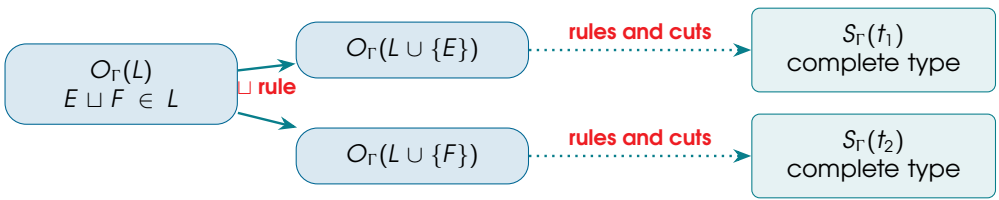
OR-tasks

$O_\Gamma(L)$: **can one individual satisfy all of L under this profile?**
L may be incomplete. One rule extends it; several steps may be needed to reach a complete type.

OR-tasks

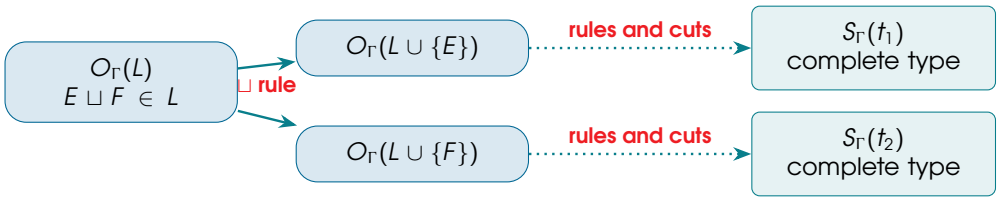
$O_\Gamma(L)$: **can one individual satisfy all of L under this profile?**

L may be incomplete. One rule extends it; several steps may be needed to reach a complete type.



OR-tasks

$O_\Gamma(L)$: **can one individual satisfy all of L under this profile?**
L may be incomplete. One rule extends it; several steps may be needed to reach a complete type.



OR: one successful alternative.

AND: all existential requirements.

We cache both incomplete OR-labels and complete AND-types, as different node kinds.

AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.

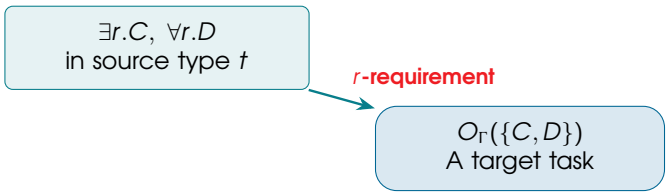
AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.

$\exists r.C,\ \forall r.D$
in source type t

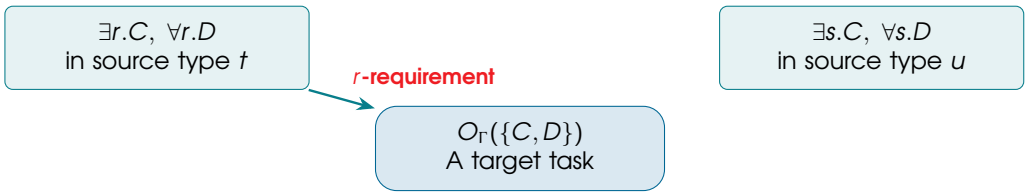
AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.



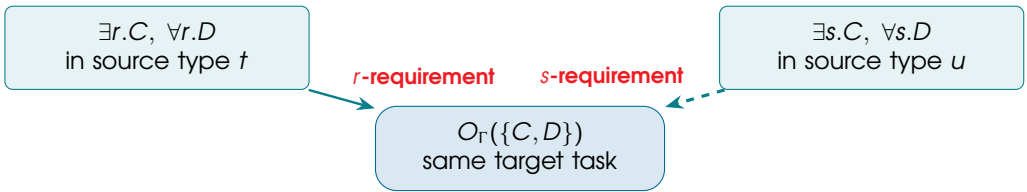
AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.



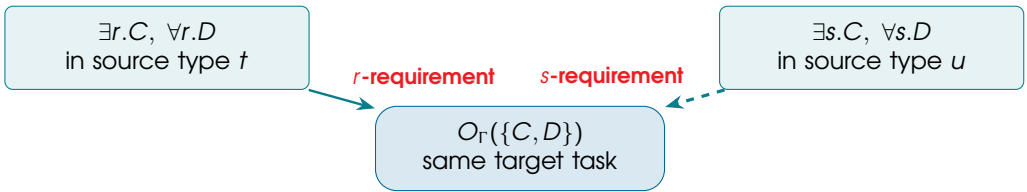
AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.



AND-tasks

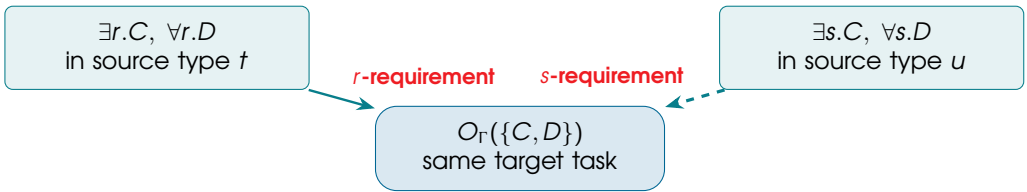
From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.



Same constraints, different roles: reuse.
Keep r or s on the dependency edge for model reconstruction, not in the target's key.

AND-tasks

From $\exists r.E \in t$ follows request $O_\Gamma(P_r(t, E))$, where $P_r(t, E) = \{E\} \cup \{F : \forall r.F \in t\}$.
The role r selects the source's restrictions; the task then asks only for the resulting **set of concepts**.



Same constraints, different roles: reuse.
Keep r or s on the dependency edge for model reconstruction, not in the target's key.

The filler alone is insufficient.
With $\forall r.\neg D$, request $\{C, \neg D\}$.
It differs from both $\{C\}$ and $\{C, D\}$.

What exactly is in the cache?

One entry = one proof-search node.

$(\Gamma,\ \text{kind},\ L) \ \mapsto\ (\text{node ID},\ \text{expansion/failure flags},\ \text{dependencies}).$

What exactly is in the cache?

One entry = one proof-search node.

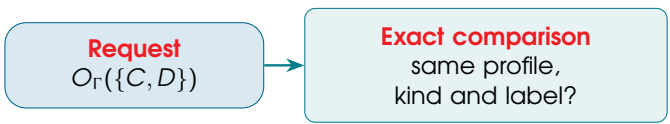
$(\Gamma,\ \text{kind},\ L) \ \mapsto\ (\text{node ID},\ \text{expansion/failure flags},\ \text{dependencies}).$

Request
$O_\Gamma(\{C, D\})$

What exactly is in the cache?

One entry = one proof-search node.

$(\Gamma,\ \text{kind},\ L)\ \mapsto\ (\text{node ID},\ \text{expansion/failure flags},\ \text{dependencies}).$



What exactly is in the cache?

One entry = one proof-search node.

$(\Gamma,\ \text{kind},\ L) \mapsto$ (node ID, expansion/failure flags, dependencies).



What exactly is in the cache?

One entry = one proof-search node.

$(\Gamma,\ \text{kind},\ L) \mapsto$ (node ID, expansion/failure flags, dependencies).



What exactly is in the cache?

One entry = one proof-search node.
$(\Gamma,\ \text{kind},\ L) \mapsto$ (node ID, expansion/failure flags, dependencies).



Equality identifies a cached node.
E.g., $O_\Gamma(\{C\}) \neq O_\Gamma(\{C, D\})$.
Its status can still be unresolved.
A hit does not mean success.

What exactly is in the cache?

One entry = one proof-search node.
$(\Gamma,\ \text{kind},\ L) \mapsto$ (node ID, expansion/failure flags, dependencies).



Equality identifies a cached node.
E.g., $O_\Gamma(\{C\}) \neq O_\Gamma(\{C, D\})$.
Its status can still be unresolved.
A hit does not mean success.

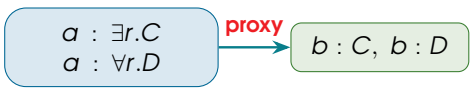
Inclusion selects a candidate witness:
$t \supseteq \{C, D\}$ may solve the task.
Reuse $S_\Gamma(t)$, not its label as an OR-node.

Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.

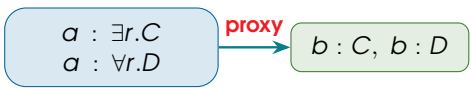
Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.

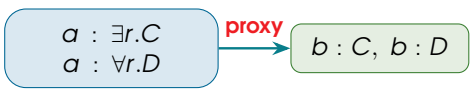


The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.

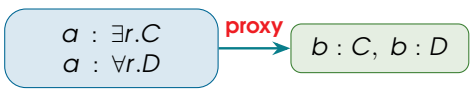
Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.



The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



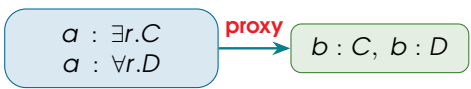
The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.

alternative 1

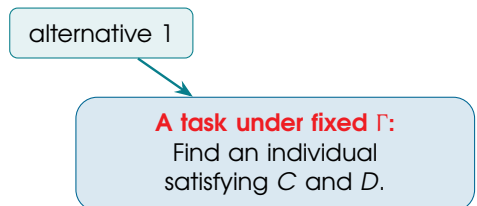
Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



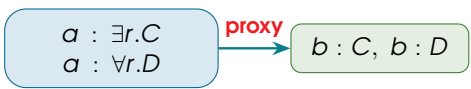
The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.



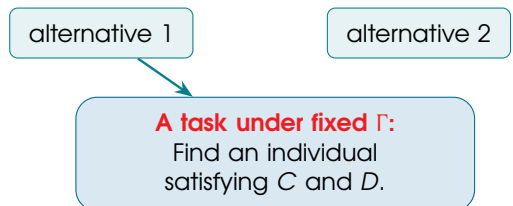
Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



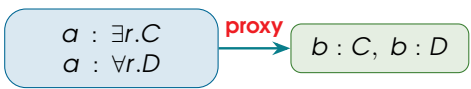
The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.



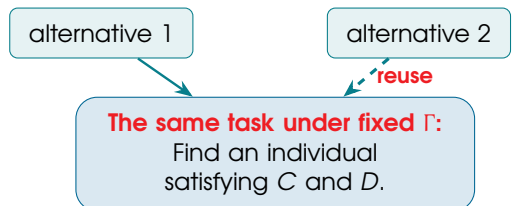
Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



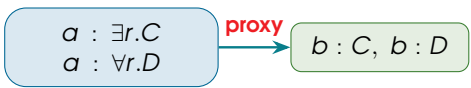
The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.



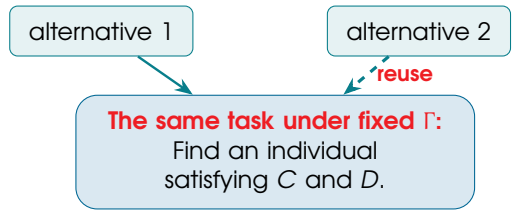
Blocking vs. caching

Blocking: on one derivation branch
An existing individual already has every concept required of the new successor. We avoid introducing another one.



The proxy is part of **this branch's** model construction. It says nothing about failures found on another alternative branch.

Caching: across alternative branches
We represent the same request once. We reuse its node and the search below it.



Remark: Solving amounts to applying rules, exploring alternatives, and checking successors.

Reusing a node shares that work; it does not yet prove that the shared node is satisfiable.

Running example: The tableau repeats a failure

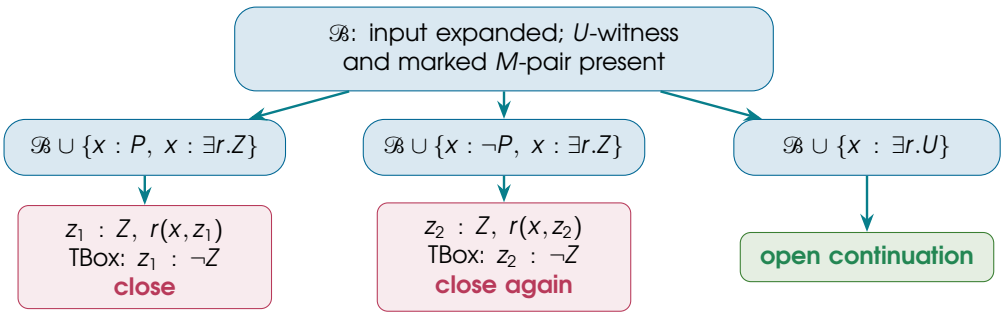
$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$

Running example: The tableau repeats a failure

$\mathcal{T} = \{M \sqsubseteq \neg U,\ M \sqsubseteq \exists s.M,\ Z \sqsubseteq \bot\}, \quad \mathcal{A} = \{a : U\}.$

$C_0 = \iota U.Q \sqcap M \sqcap \neg\{\iota M\} \sqcap D, \quad D = (P \sqcap \exists r.Z) \sqcup (\neg P \sqcap \exists r.Z) \sqcup \exists r.U.$



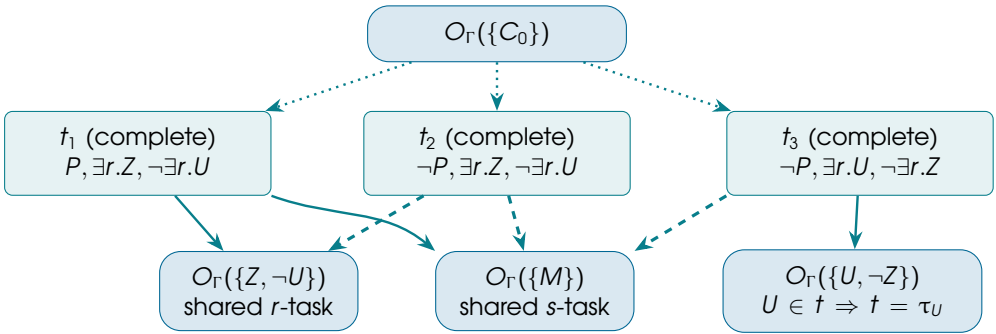
Two alternatives, the same successor question.

Running example: A cached run

The profile from slides 11–12 is already frozen. Complete input types must contain $M, \neg U, \neg Z, \exists s.M$ (TBox requirements); every type agrees with $\iota U.Q$.

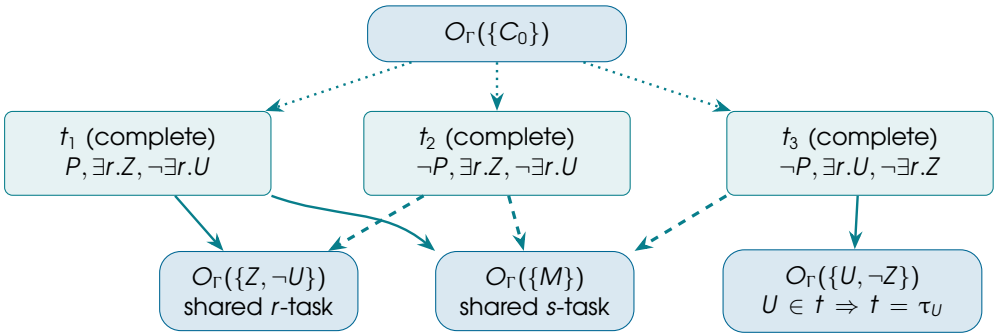
Running example: A cached run

The profile from slides 11–12 is already frozen. Complete input types must contain $M, \neg U, \neg Z, \exists s.M$ (TBox requirements); every type agrees with $\iota U.Q$.



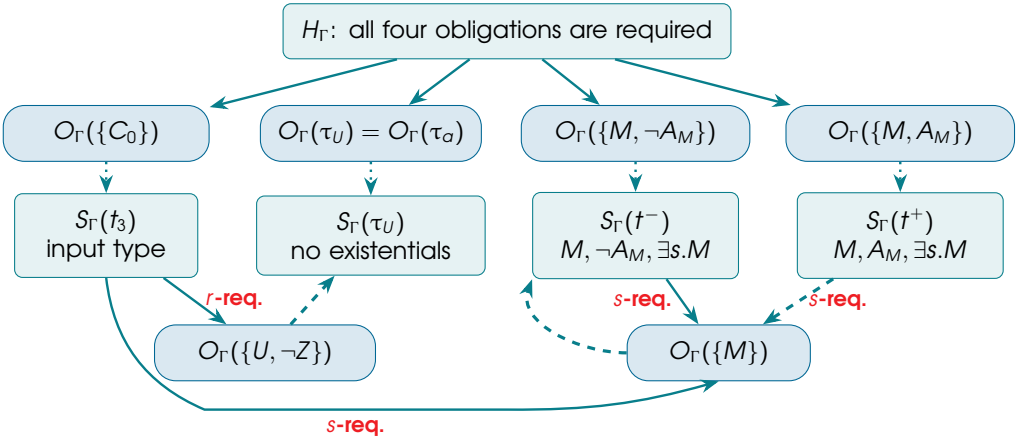
Running example: A cached run

The profile from slides 11–12 is already frozen. Complete input types must contain $M, \neg U, \neg Z, \exists s.M$ (TBox requirements); every type agrees with $\iota U.Q$.

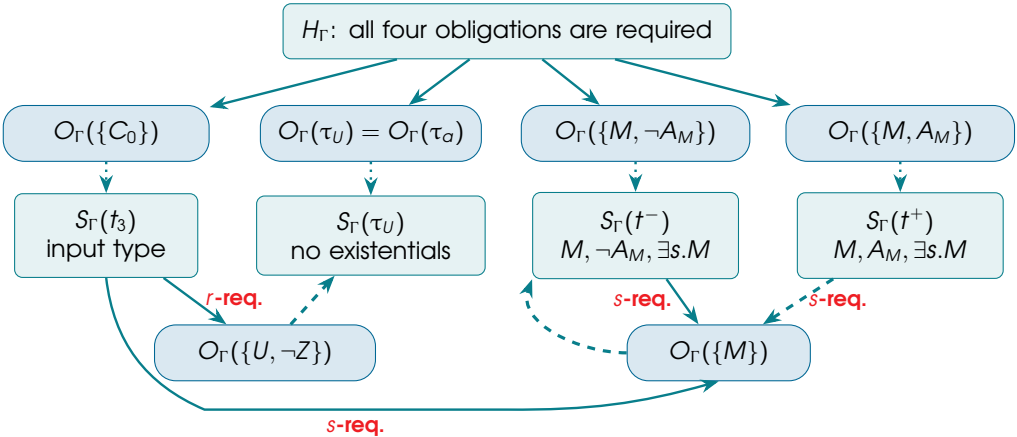


Three representative **complete** types are shown and all their existential dependencies are drawn. Dotted paths stand for several completion steps.

Running example: The rest of the root



Running example: The rest of the root



This is a **candidate support graph**; we still have to propagate all failures before accepting it.

Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.

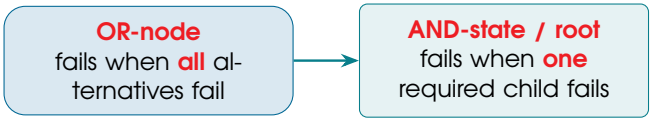
Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.

OR-node
fails when **all** alternatives fail

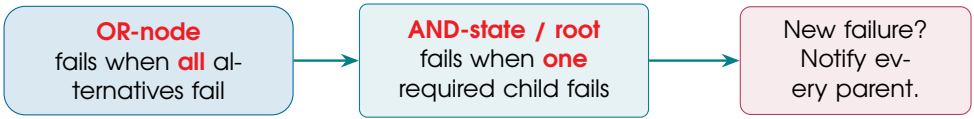
Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.



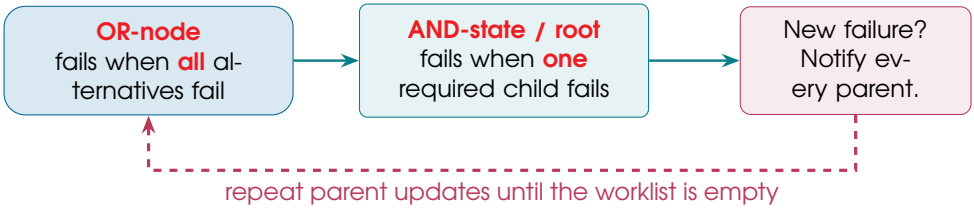
Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.



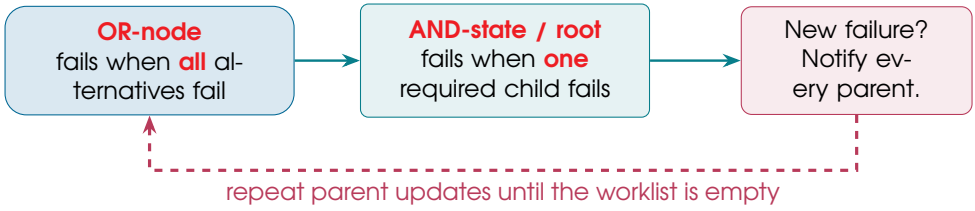
Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.



Running example: Propagating failure

We finish expansion first. We mark clashes and OR-nodes with no admissible continuation as failed.
We put newly failed nodes on a worklist. All other nodes start unresolved.



Worklist empty: the remaining part survives:
Every OR-node has an unfailed option;
Every AND-node has all required children unfailed.
We accept the profile exactly when its root H_Γ survives.

Running example: Propagting failure (cont'd)

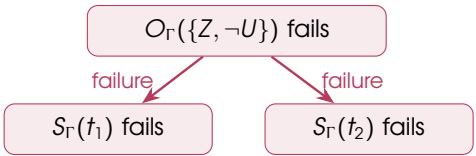
Failure travels backwards

$Z \sqsubseteq \bot$ leaves no completion of a request containing Z.

Running example: Propagting failure (cont'd)

Failure travels backwards

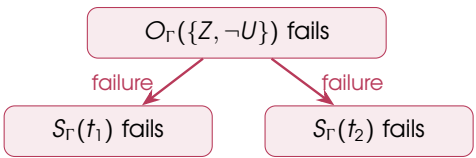
$Z \sqsubseteq \bot$ leaves no completion of a request containing Z.



Running example: Propagting failure (cont'd)

Failure travels backwards

$Z \sqsubseteq \bot$ leaves no completion of a request containing Z.



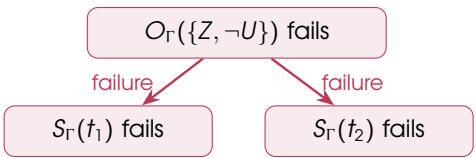
A supported cycle

t^- needs an M-successor and can provide one. There is no additional impossible requirement.

Running example: Propagting failure (cont'd)

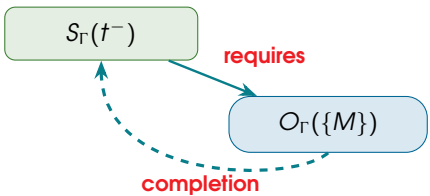
Failure travels backwards

$Z \sqsubseteq \bot$ leaves no completion of a request containing Z.



A supported cycle

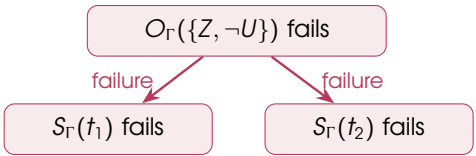
t^- needs an M-successor and can provide one. There is no additional impossible requirement.



Running example: Propagting failure (cont'd)

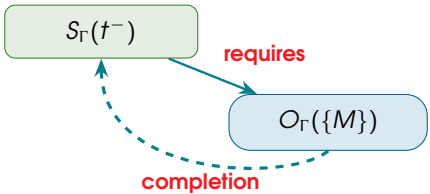
Failure travels backwards

$Z \sqsubseteq \bot$ leaves no completion of a request containing Z.



A supported cycle

t^- needs an M-successor and can provide one. There is no additional impossible requirement.



t_3, τ_U, t^-, t^+ remain supported. Every required root task has a surviving completion.
Result: this Γ is accepted; the input is satisfiable.

Why the construction is correct?

Model $\Rightarrow$ accepted profile
Read σ and the designated types. Mark two distinct witnesses for each non-singleton subject. Model witnesses support every task.

Why the construction is correct?

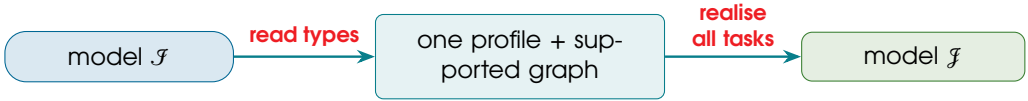
Model $\Rightarrow$ accepted profile
Read σ and the designated types. Mark two distinct witnesses for each non-singleton subject. Model witnesses support every task.

Accepted profile $\Rightarrow$ model
Retain types covering all root tasks and their successor requirements. Interpret their concepts and role edges as just shown.

Why the construction is correct?

Model ⇒ accepted profile
Read σ and the designated types. Mark two distinct witnesses for each non-singleton subject. Model witnesses support every task.

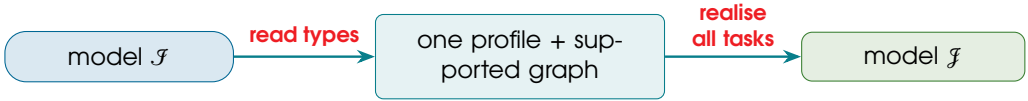
Accepted profile ⇒ model
Retain types covering all root tasks and their successor requirements. Interpret their concepts and role edges as just shown.



Why the construction is correct?

Model ⇒ accepted profile
Read σ and the designated types. Mark two distinct witnesses for each non-singleton subject. Model witnesses support every task.

Accepted profile ⇒ model
Retain types covering all root tasks and their successor requirements. Interpret their concepts and role edges as just shown.



$\sigma(C) = 0$
No retained type contains C.
So $C^{\mathcal{J}} = \varnothing$.

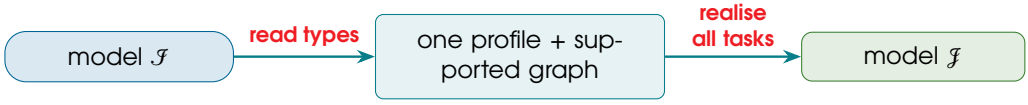
$\sigma(C) = 1$
$C \in t$ forces $t = \tau_C$.
The mandatory τ_C exists.

$\sigma(C) = 2$
Both marker signs are realised.
They give distinct C-elements.

Why the construction is correct?

Model ⇒ accepted profile
Read σ and the designated types. Mark two distinct witnesses for each non-singleton subject. Model witnesses support every task.

Accepted profile ⇒ model
Retain types covering all root tasks and their successor requirements. Interpret their concepts and role edges as just shown.



$\sigma(C) = 0$
No retained type contains C.
So $C^{\mathcal{J}} = \varnothing$.

$\sigma(C) = 1$
$C \in t$ forces $t = \tau_C$.
The mandatory τ_C exists.

$\sigma(C) = 2$
Both marker signs are realised.
They give distinct C-elements.

Remark: The key is joint realisation: all surviving tasks fit one common profile and one model.

Why does it run in EXPTIME?

Profile size

Let n be input size, $m = |\,\mathrm{cl}_\sigma\,| = O(n)$, and $q = O(n)$ the number of designated types. A node label is an m-bit vector. A full profile uses $O(qm) = O(n^2)$ bits.

Why does it run in EXPTIME?

Profile size

Let n be input size, $m = |\,\mathrm{cl}_\sigma\,| = O(n)$, and $q = O(n)$ the number of designated types. A node label is an m-bit vector. A full profile uses $O(qm) = O(n^2)$ bits.

How many objects?

Profiles: $3^{O(n)}(2^m)^q = 2^{O(n^2)}$.
For fixed Γ: # nodes$\leq 2 \cdot 2^m + 1$.
Rule-by-rule: $O(m)$ edges per state;
$O(n)$ root tasks. Thus # edges$= O(mV + n)$.

Why does it run in EXPTIME?

Profile size

Let n be input size, $m = |\,\mathrm{cl}_\sigma\,| = O(n)$, and $q = O(n)$ the number of designated types. A node label is an m-bit vector. A full profile uses $O(qm) = O(n^2)$ bits.

How many objects?

Profiles: $3^{O(n)}(2^m)^q = 2^{O(n^2)}$.

For fixed Γ: # nodes$\leq 2 \cdot 2^m + 1$.

Rule-by-rule: $O(m)$ edges per state; $O(n)$ root tasks. Thus # edges$= O(mV + n)$.

What does processing cost?

Label comparison: $O(m)$ time.

Lookup: $O(m \log(\#\text{nodes}))$ per request.

Node check/expansion: $\mathrm{poly}(n)$ time.

Failure worklist: $O(\#\text{nodes} + E)$ updates.

Why does it run in EXPTIME?

Profile size

Let n be input size, $m = |\,\mathrm{cl}_\sigma\,| = O(n)$, and $q = O(n)$ the number of designated types. A node label is an m-bit vector. A full profile uses $O(qm) = O(n^2)$ bits.

How many objects?

Profiles: $3^{O(n)}(2^m)^q = 2^{O(n^2)}$.
For fixed Γ: # nodes$\leq 2 \cdot 2^m + 1$.
Rule-by-rule: $O(m)$ edges per state; $O(n)$ root tasks. Thus # edges$= O(mV + n)$.

What does processing cost?

Label comparison: $O(m)$ time.
Lookup: $O(m \log(\#\text{nodes}))$ per request.
Node check/expansion: $\mathrm{poly}(n)$ time.
Failure worklist: $O(\#\text{nodes} + E)$ updates.

One profile: $\mathrm{poly}(n)(V + E)\log(V + 1) = 2^{O(n)}$ time, including cache access.

All profiles: $2^{O(n^2)} \cdot 2^{O(n)} = \boxed{2^{O(n^2)}}$ deterministic time.

Summary

1. The revised DD-rules remain the **semantic core**; the proof object changes from a derivation tree to a shared AND-OR graph.

Summary

1. The revised DD-rules remain the **semantic core**; the proof object changes from a derivation tree to a shared AND-OR graph.

2. A frozen profile Γ makes uniqueness information and named-individual theories consistent across every reused obligation.

Summary

1. The revised DD-rules remain the **semantic core**; the proof object changes from a derivation tree to a shared AND-OR graph.

2. A frozen profile Γ makes uniqueness information and named-individual theories consistent across every reused obligation.

3. Each identical obligation is expanded once, giving a $2^{O(n^2)}$ worst-case bound and an ExpTime-optimal procedure.

Summary

1. The revised DD-rules remain the **semantic core**; the proof object changes from a derivation tree to a shared AND-OR graph.

2. A frozen profile Γ makes uniqueness information and named-individual theories consistent across every reused obligation.

3. Each identical obligation is expanded once, giving a $2^{O(n^2)}$ worst-case bound and an ExpTime-optimal procedure.

IN A NUTSHELL: We **(1)** build one complete profile, **(2)** freeze it, and **(3)** solve all local obligations in one globally shared AND-OR graph.

Summary

1. The revised DD-rules remain the **semantic core**; the proof object changes from a derivation tree to a shared AND-OR graph.

2. A frozen profile Γ makes uniqueness information and named-individual theories consistent across every reused obligation.

3. Each identical obligation is expanded once, giving a $2^{O(n^2)}$ worst-case bound and an ExpTime-optimal procedure.

IN A NUTSHELL: We **(1)** build one complete profile, **(2)** freeze it, and **(3)** solve all local obligations in one globally shared AND-OR graph.

References

- M. Sochański, P. A. Wałęga, M. Zawidzki. **Description Logics with Two Types of Definite Descriptions: Complexity, Expressiveness, and Automated Deduction**. AAAI 2026, pp. 19371–19379.
- M. Sochański, M. Zawidzki, P. A. Wałęga. **Python-Based Reasoner for Description Logic with Definite Descriptions**. ARQNL 2026, 15 pp.
- F. M. Donini, F. Massacci. **EXPTIME tableaux for ALC**. Artificial Intelligence 124(1) (2000), pp. 87–138.
- R. Goré, L. A. Nguyen. **ExpTime Tableaux for ALC Using Sound Global Caching**. Journal of Automated Reasoning 50(4) (2013), pp. 355–381.

Thank You.

* This research is funded by the European Union (ERC, ExtenDD, project number: 101054714). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.