Ideal Downward Refinement in the $\mathcal{EL}$ Description Logic

Jens Lehmann Christoph Haase

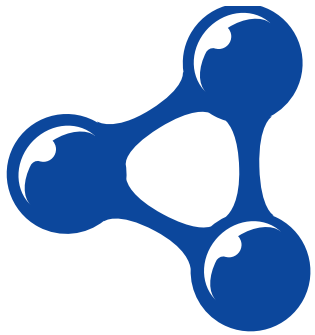
UNIVERSITÄT LEIPZIG

July 3, 2009

Outline

Outline

- Description Logics is the name of a family of languages for knowledge representation
- fragments of first order predicate logic
- less expressive power than predicate logic, but usually decidable inference problems
- intuitive variable free syntax
- basis of the ontology language OWL



Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`
- concepts
  - correspond to unary predicates
  - sets of objects
  - examples: `Student`, `Car`, `Country`

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`
- concepts
  - correspond to unary predicates
  - sets of objects
  - examples: `Student`, `Car`, `Country`
- roles
  - correspond to binary predicates
  - describe connections between objects
  - examples: `hasChild`, `isPartOf`

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`
- concepts
  - correspond to unary predicates
  - sets of objects
  - examples: `Student`, `Car`, `Country`
- roles
  - correspond to binary predicates
  - describe connections between objects
  - examples: `hasChild`, `isPartOf`
- can be combined to complex concepts, e.g.:
  $\texttt{Car} \sqcap \exists \texttt{hasEngine}.\texttt{Mercedes}$

construct	syntax	semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$

Table: $\mathcal{EL}$ concept constructors

$\mathcal{EL}$ - Syntax, Semantics, Reasoning

construct	syntax	semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$

Table: $\mathcal{EL}$ concept constructors

- DLs provide a variety of reasoning/inference tasks
  - subsumption between concepts ($C \sqsubseteq D$), e.g. $\texttt{Mother} \sqsubseteq \texttt{Female}$
  - satisfiability of concepts ($C \equiv \bot$), e.g. $\texttt{Mother} \sqcap \texttt{Male} \equiv \bot$

$\mathcal{EL}$ - Syntax, Semantics, Reasoning

construct	syntax	semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$

Table: $\mathcal{EL}$ concept constructors

- DLs provide a variety of reasoning/inference tasks
  - subsumption between concepts ($C \sqsubseteq D$), e.g. $\texttt{Mother} \sqsubseteq \texttt{Female}$
  - satisfiability of concepts ($C \equiv \bot$), e.g. $\texttt{Mother} \sqcap \texttt{Male} \equiv \bot$
- poly-time inference (efficient reasoning over large knowledge bases)
- widely used, e.g. in SNOMED-CT, GENE ontologies

Knowledge Bases

A knowledge base has the following structure:

knowledge base

TBox $\mathcal{T}$ ("terminology")

ABox $\mathcal{A}$ ("assertions")

Knowledge Bases

A knowledge base has the following structure:

knowledge base

TBox $\mathcal{T}$ ("terminology"), e.g.

$\texttt{Pupil} \equiv \texttt{Person} \sqcap \exists \texttt{studiesAt.School}$

$\texttt{UnderGradStudent} \sqsubseteq \texttt{Poor} \sqcap \texttt{EducatedPerson}$

$\texttt{BusyPhDStudent} \sqsubseteq \exists \texttt{theses.UnfinishedPhDTheses}$

ABox $\mathcal{A}$ ("assertions")

Knowledge Bases

A knowledge base has the following structure:

knowledge base

TBox $\mathcal{T}$ ("terminology"), e.g.

$\texttt{Pupil} \equiv \texttt{Person} \sqcap \exists \texttt{studiesAt.School}$

$\texttt{UnderGradStudent} \sqsubseteq \texttt{Poor} \sqcap \texttt{EducatedPerson}$

$\texttt{BusyPhDStudent} \sqsubseteq \exists \texttt{theses.UnfinishedPhDTheses}$

ABox $\mathcal{A}$ ("assertions"), e.g.

$\texttt{PhDStudent}(\texttt{CHRISTOPH})$

$\texttt{affiliation}(\texttt{CHRISTOPH}, \texttt{OXFORD\_UNIVERSITY})$

OWL

- ontology = an explicit specification of a conceptualization of a domain of interest
- OWL is an acronym for Web Ontology Language
- based on description logics
- W3C recommendation since 2004 (OWL 2 in 2009 or 2010)
- widely used standard for representing knowledge in the Semantic Web (with many application areas outside the Web)
- OWL 2 EL++ profile based on $\mathcal{EL}$

Outline

Refinement Operators - Definitions

- given a DL $\mathcal{L}$, consider the quasi-ordered space $\langle \mathcal{C}(\mathcal{L}), \sqsubseteq_{\mathcal{T}} \rangle$ over concepts of $\mathcal{L}$
- $\rho : \mathcal{C}(\mathcal{L}) \to 2^{\mathcal{C}(\mathcal{L})}$ is a downward $\mathcal{L}$ refinement operator if for any $C \in \mathcal{C}(\mathcal{L})$:

$$D \in \rho(C) \text{ implies } D \sqsubseteq_{\mathcal{T}} C$$

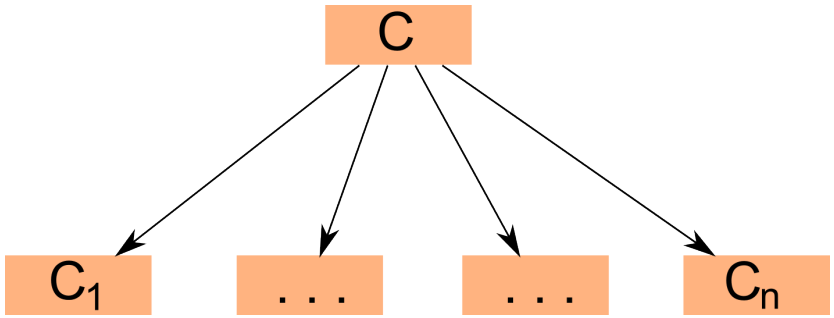
- notation: Write $C \rightsquigarrow_\rho D$ instead of $D \in \rho(C)$
- example refinement chain in $\langle \mathcal{C}(\mathcal{EL}), \sqsubseteq_{\mathcal{T}} \rangle$:

$$\top \rightsquigarrow_\rho \texttt{Male} \rightsquigarrow_\rho \texttt{Male} \sqcap \exists \texttt{hasChild}.\top$$

Properties of Refinement Operators

An $\mathcal{L}$ downward refinement operator ρ is called

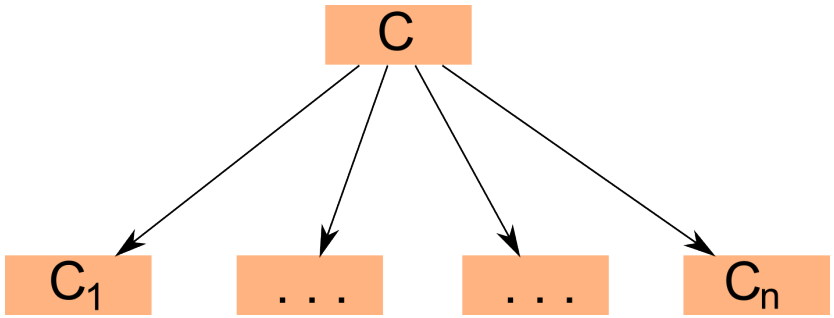
- finite iff $\rho(C)$ is finite for any concept $C \in \mathcal{C}(\mathcal{L})$

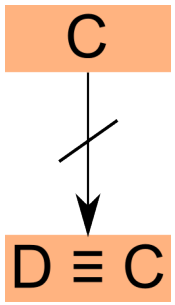


Properties of Refinement Operators

An $\mathcal{L}$ downward refinement operator ρ is called

- finite iff $\rho(C)$ is finite for any concept $C \in \mathcal{C}(\mathcal{L})$
- proper iff for $C, D \in \mathcal{C}(\mathcal{L})$, $C \rightsquigarrow_\rho D$ implies $C \not\equiv_\mathcal{T} D$

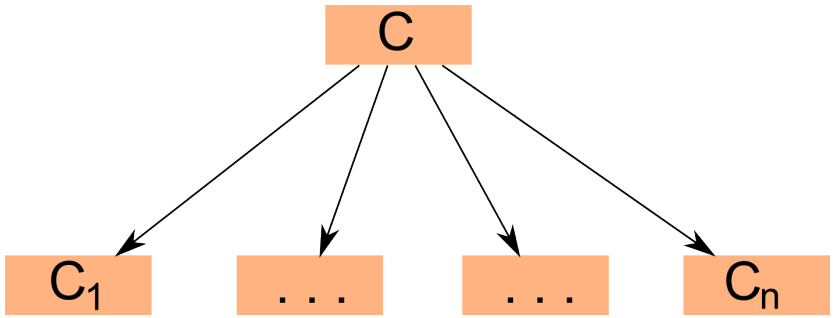


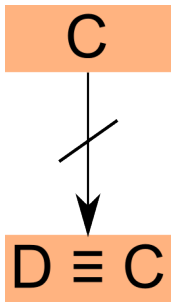


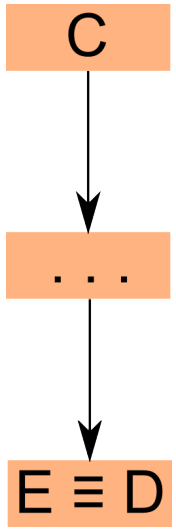
Properties of Refinement Operators

An $\mathcal{L}$ downward refinement operator ρ is called

- finite iff $\rho(C)$ is finite for any concept $C \in \mathcal{C}(\mathcal{L})$
- proper iff for $C, D \in \mathcal{C}(\mathcal{L})$, $C \rightsquigarrow_\rho D$ implies $C \not\equiv_\mathcal{T} D$
- complete iff for $C, D \in \mathcal{C}(\mathcal{L})$ with $C \sqsubset_\mathcal{T} D$ there is a concept E with $E \equiv_\mathcal{T} D$ and a refinement chain $C \rightsquigarrow_\rho \cdots \rightsquigarrow_\rho E$



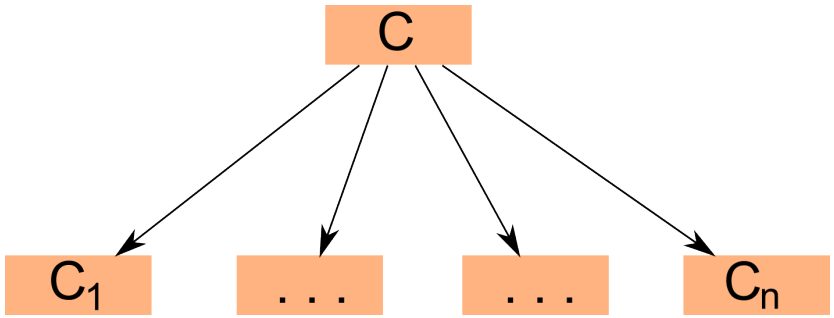


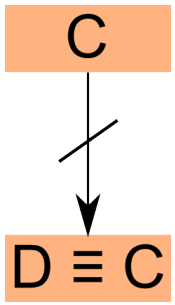


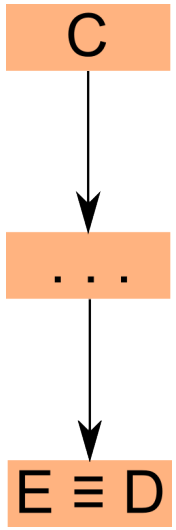
Properties of Refinement Operators

An $\mathcal{L}$ downward refinement operator ρ is called

- finite iff $\rho(C)$ is finite for any concept $C \in \mathcal{C}(\mathcal{L})$
- proper iff for $C, D \in \mathcal{C}(\mathcal{L})$, $C \rightsquigarrow_\rho D$ implies $C \not\equiv_\mathcal{T} D$
- complete iff for $C, D \in \mathcal{C}(\mathcal{L})$ with $C \sqsubset_\mathcal{T} D$ there is a concept E with $E \equiv_\mathcal{T} D$ and a refinement chain $C \rightsquigarrow_\rho \cdots \rightsquigarrow_\rho E$
- ideal = complete + proper + finite
- previous research: no ideal refinement for expressive description logics and OWL







Outline

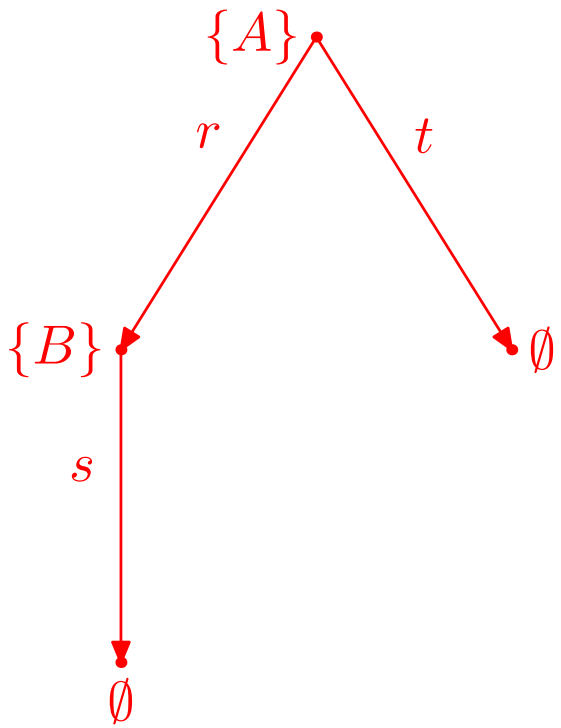
$\mathcal{EL}$ trees

- an $\mathcal{EL}$ concept can be represented by an $\mathcal{EL}$ tree and vice versa
- $\mathcal{EL}$ trees are trees $t = (V, E, \ell)$ where
  - nodes V are labeled with concepts names
  - edges E are labeled with role names

$\mathcal{EL}$ trees

- an $\mathcal{EL}$ concept can be represented by an $\mathcal{EL}$ tree and vice versa
- $\mathcal{EL}$ trees are trees $t = (V, E, \ell)$ where
  - nodes V are labeled with concepts names
  - edges E are labeled with role names

Example: $A \sqcap \exists r.(B \sqcap \exists s.\top) \sqcap \exists t.\top$ has the corresponding tree



Subsumption as Simulation between $\mathcal{EL}$ trees

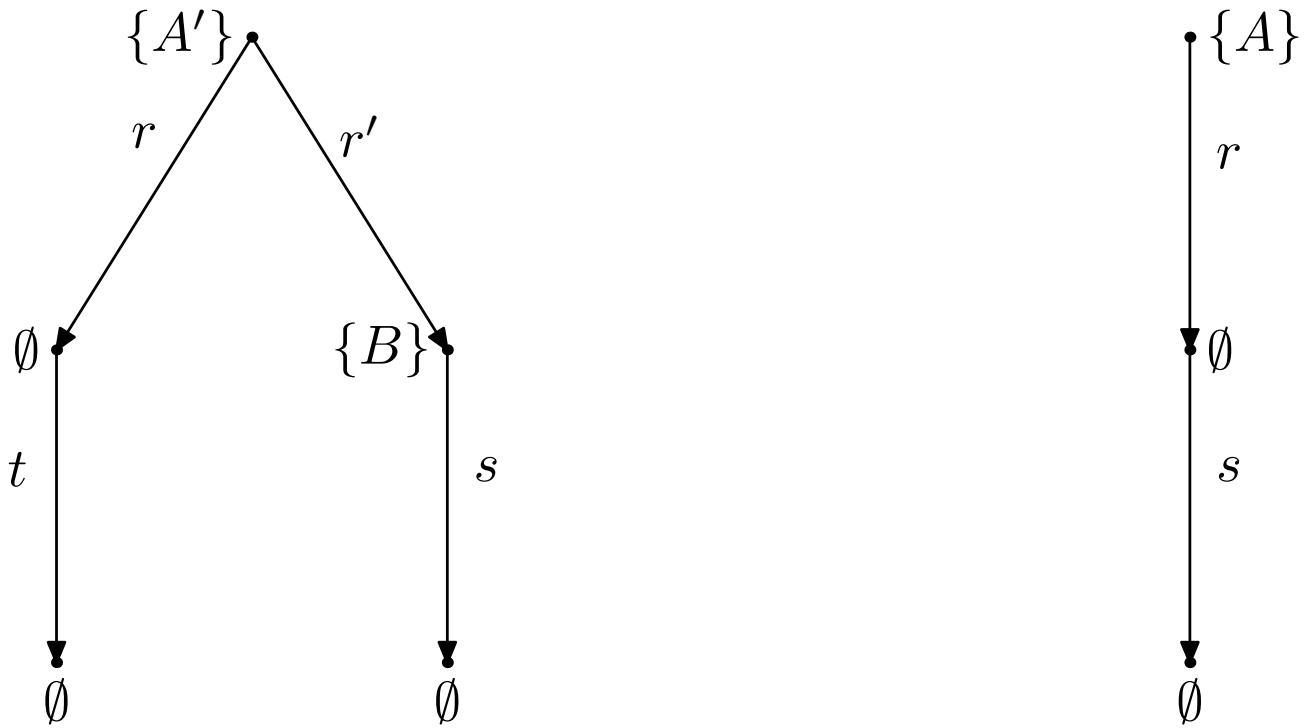
- let $\mathcal{K}$ be a knowledge base, and $t_C = (V_C, E_C, \ell_C)$, $t_D = (V_D, E_D, \ell_D)$ be $\mathcal{EL}$ trees corresponding to $\mathcal{EL}$ concepts C, D
- a relation $\mathcal{S} \subseteq V_D \times V_C$ is a simulation relation from t_D to t_C if
  - $(root(t_D), root(t_C)) \in \mathcal{S}$
  - if $(v_D, v_C) \in \mathcal{S}$ then the node v_C simulates "the behaviour" of node v_D
- can be computed in $\mathcal{O}(|V_C||V_D|)$

Theorem

There exists a simulation relation from t_D to t_C if and only if $C \sqsubseteq_{\mathcal{T}} D$.

Subsumption as Simulation between $\mathcal{EL}$ trees

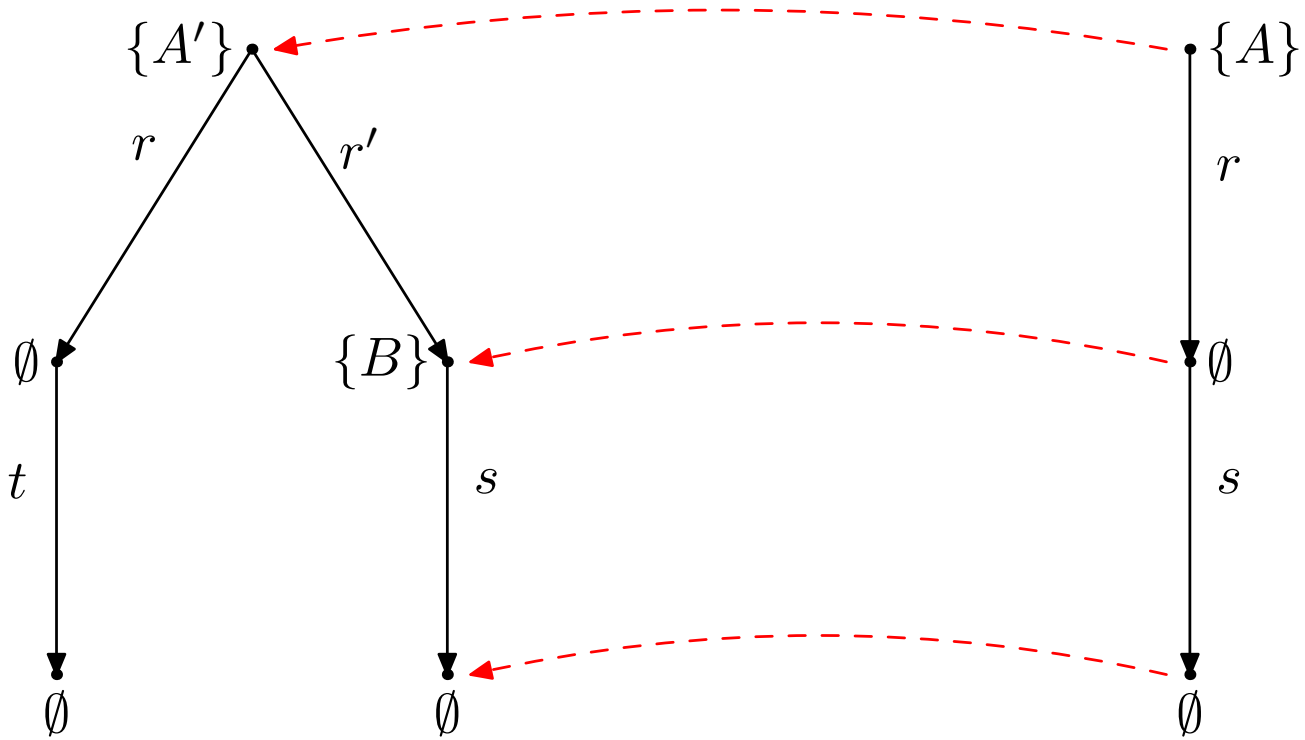
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



$$A' \sqcap \exists r.\exists t.\top \sqcap \exists r'.(B \sqcap \exists s.\top) \sqsubseteq A \sqcap \exists r.\exists s.\top$$

Subsumption as Simulation between $\mathcal{EL}$ trees

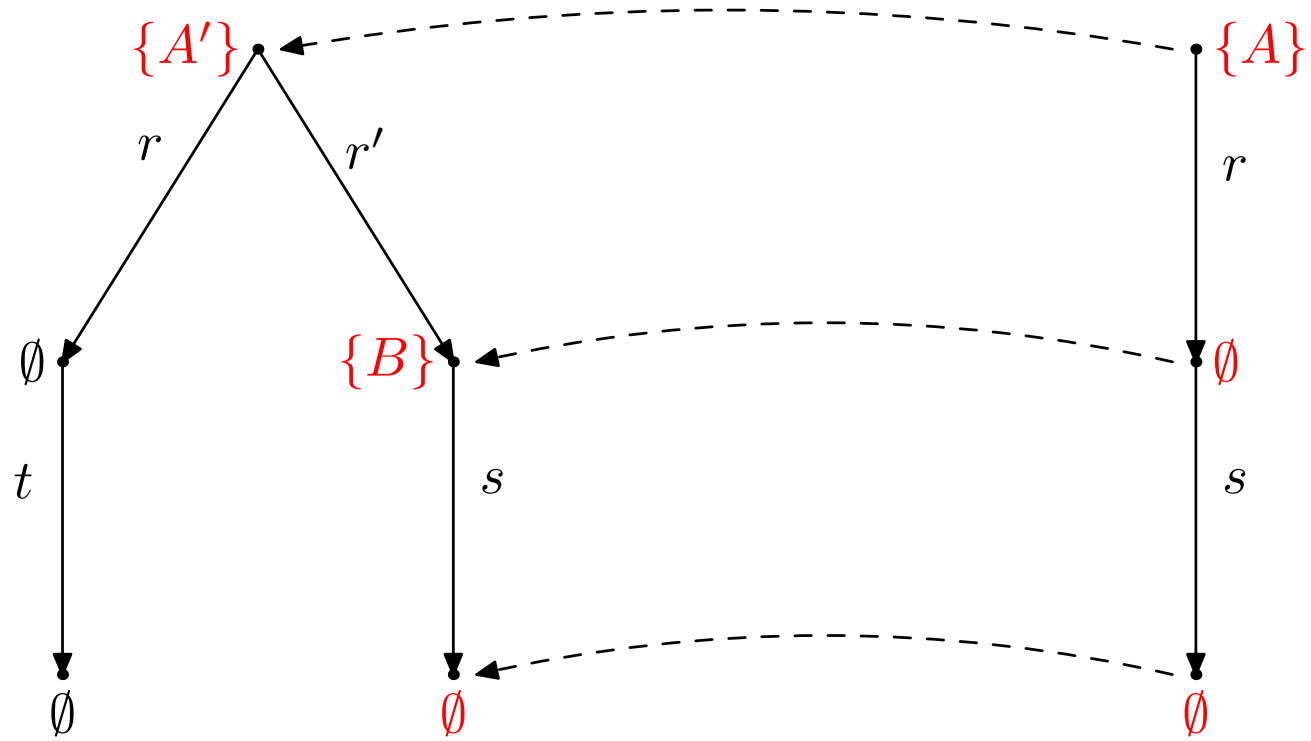
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



Simulation relation $\mathcal{S}$

Subsumption as Simulation between $\mathcal{EL}$ trees

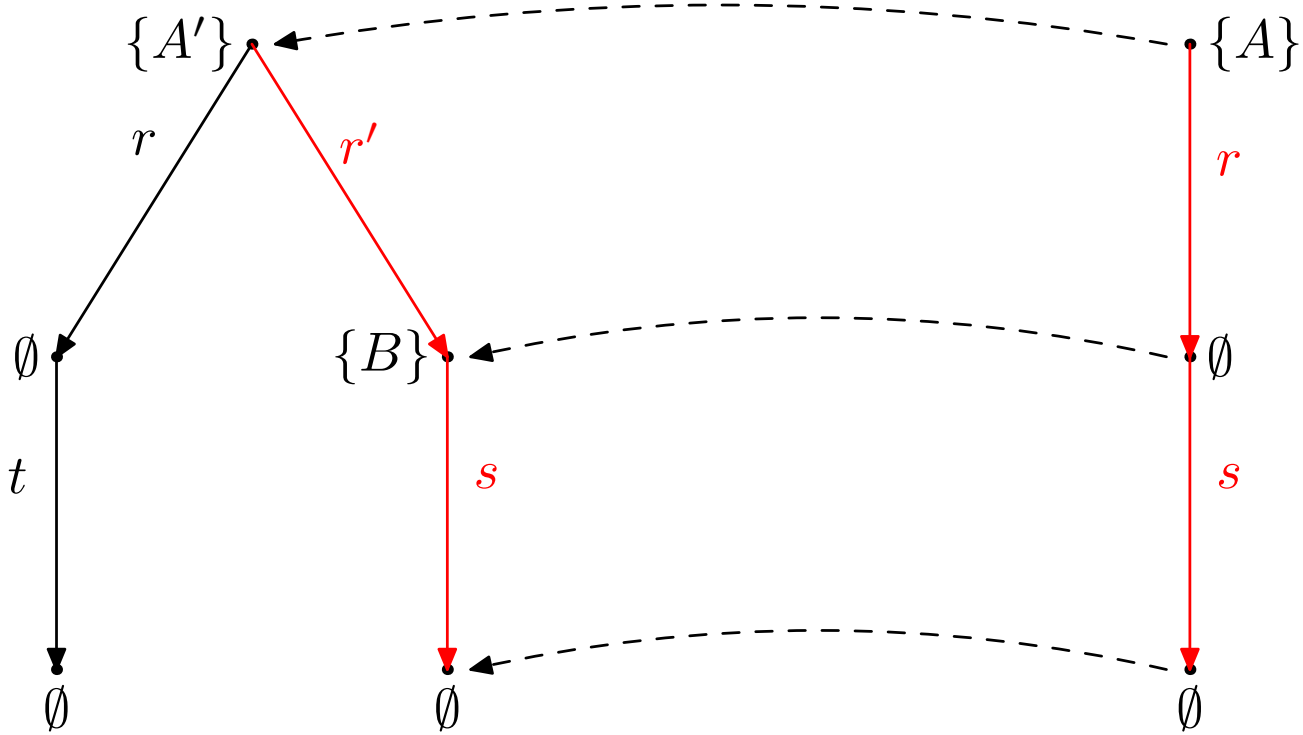
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



Cond. 1 on $(v, v') \in \mathcal{S}$: For $A \in \ell_D(v)$ there is $A' \in \ell_C(v')$ s.t. $A' \sqsubseteq_{\mathcal{T}} A$

Subsumption as Simulation between $\mathcal{EL}$ trees

$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



Cond. 2 on $(v, v') \in \mathcal{S}$: For $(v, r, w) \in E_D$ there is $(v', r', w') \in E_C$ s.t. $r' \sqsubseteq_{\mathcal{T}} r$ and $(w, w') \in \mathcal{S}$

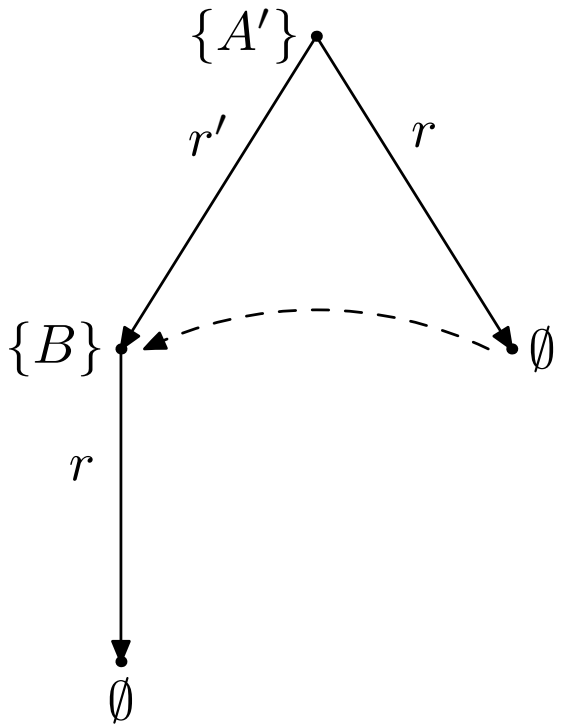
Minimal $\mathcal{EL}$ trees

- minimal $\mathcal{EL}$ trees allow for a canonical representation of a set of equivalent $\mathcal{EL}$ trees
- informally, an $\mathcal{EL}$ tree is minimal if
  - its labels cannot be further reduced
  - it does not contain redundant subtrees
- redundant subtrees can be detected by computing a simulation on a tree itself

Minimal $\mathcal{EL}$ trees

An example of a non-minimal tree

$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$

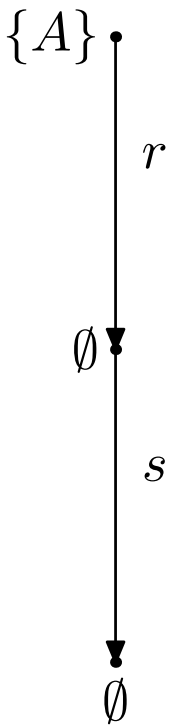


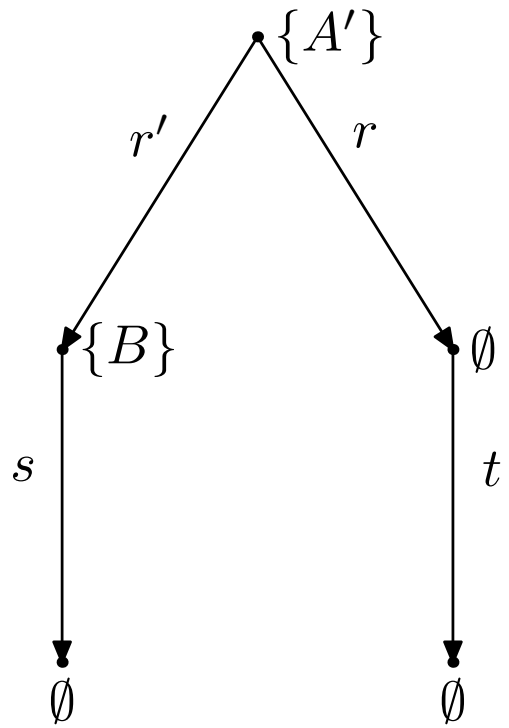
$\mathcal{EL}$ refinement operator

- maps an $\mathcal{EL}$ tree to a set of refined $\mathcal{EL}$ trees obtained by performing one of the following operations on the input tree:
  - extend label of a node
  - refine label of a node
  - refine label of an edge
  - create new subtree at a node
- checking properness of a refinement corresponds to checking minimality of the obtained tree
- if the obtained tree is not minimal it is either discarded or further refined

A refinement example

$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$

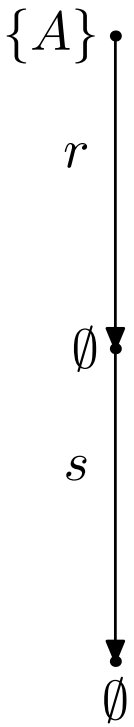




How can we reach the tree on the right starting from the tree on the left?

A refinement example

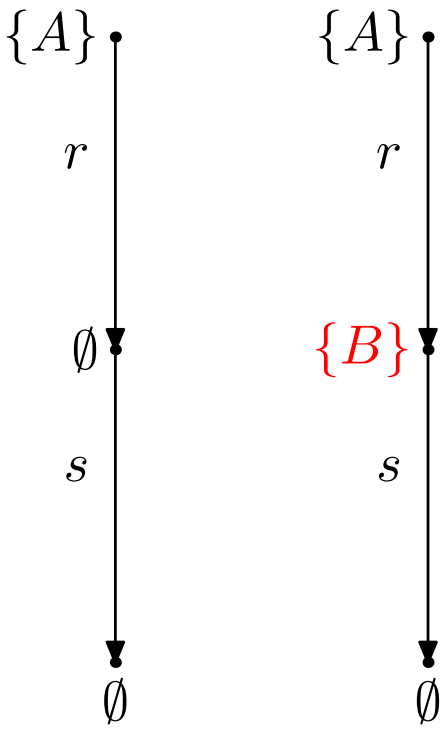
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



source tree

A refinement example

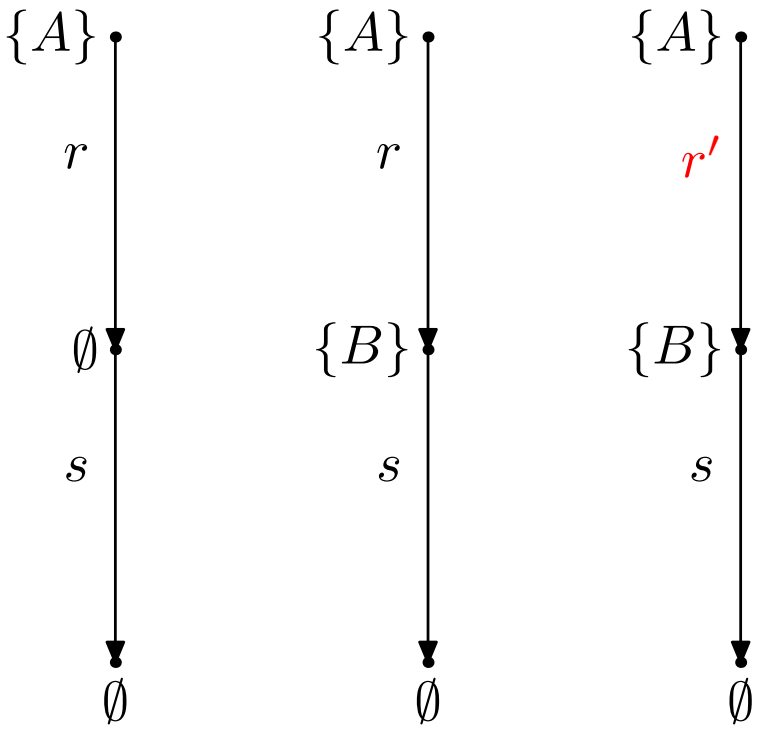
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



extend label ✓

A refinement example

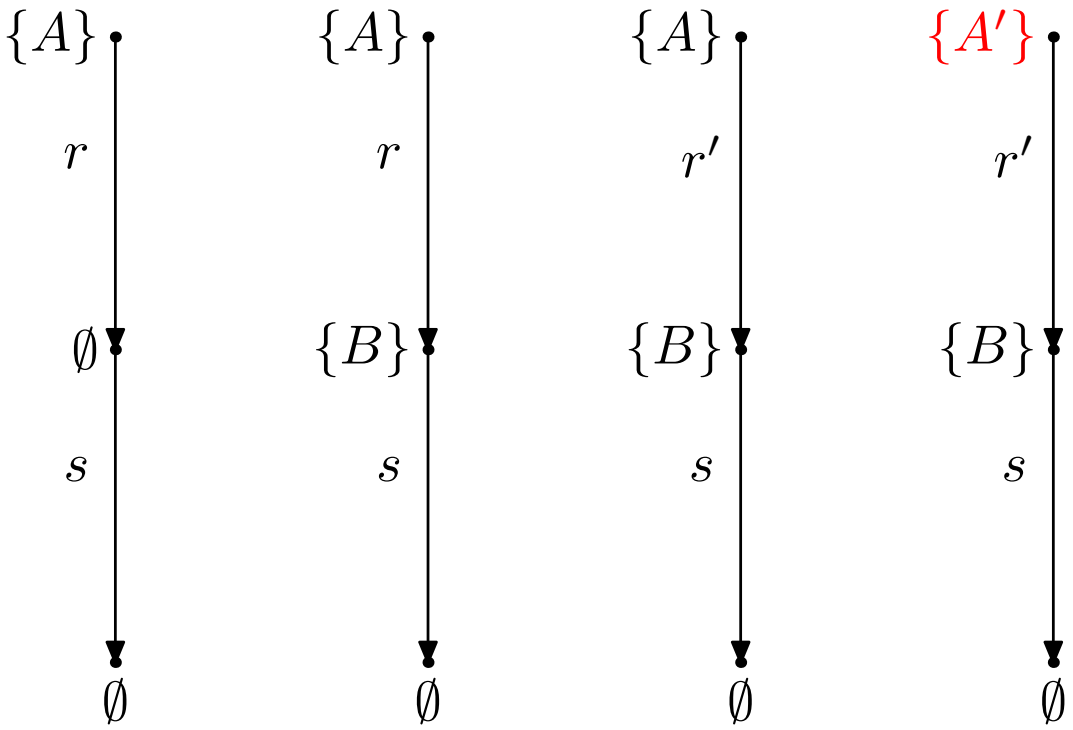
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



refine role ✓

A refinement example

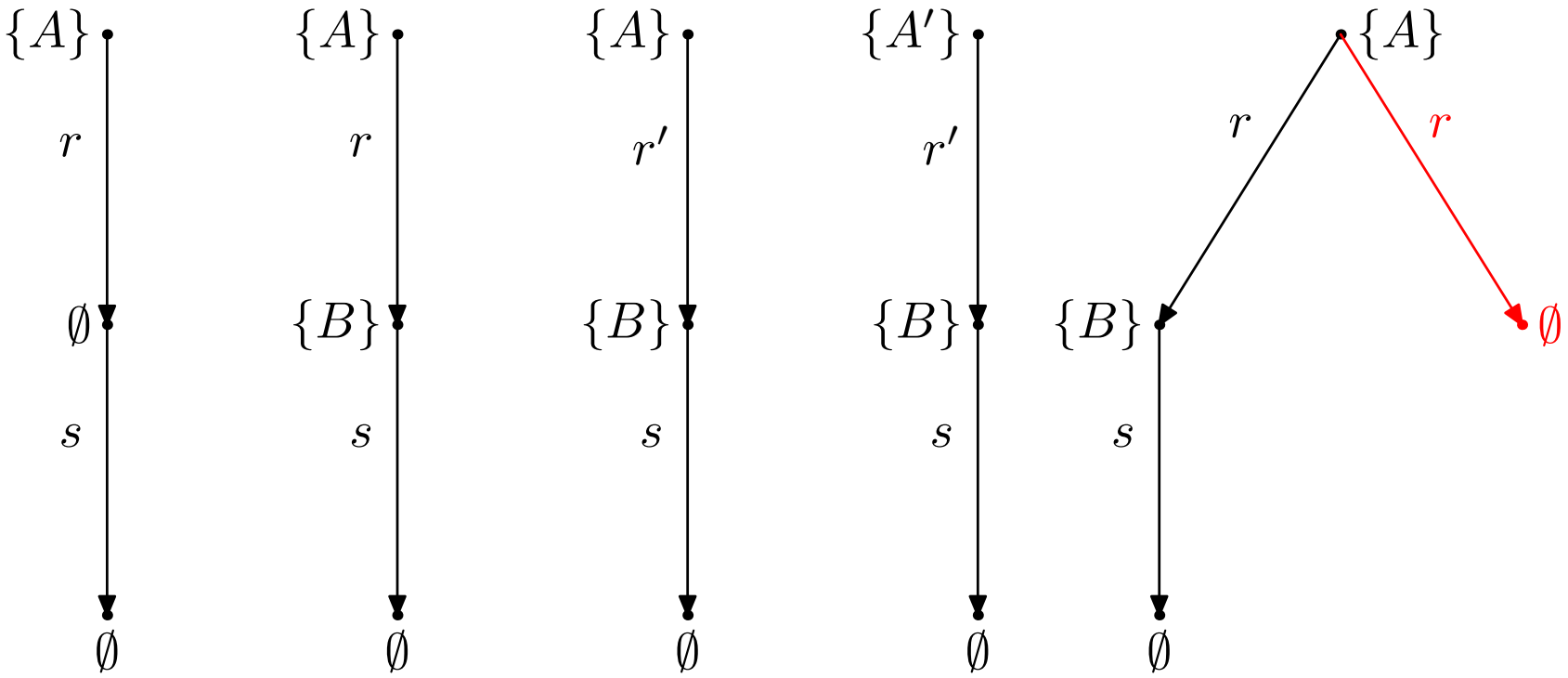
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



refine label ✓

A refinement example

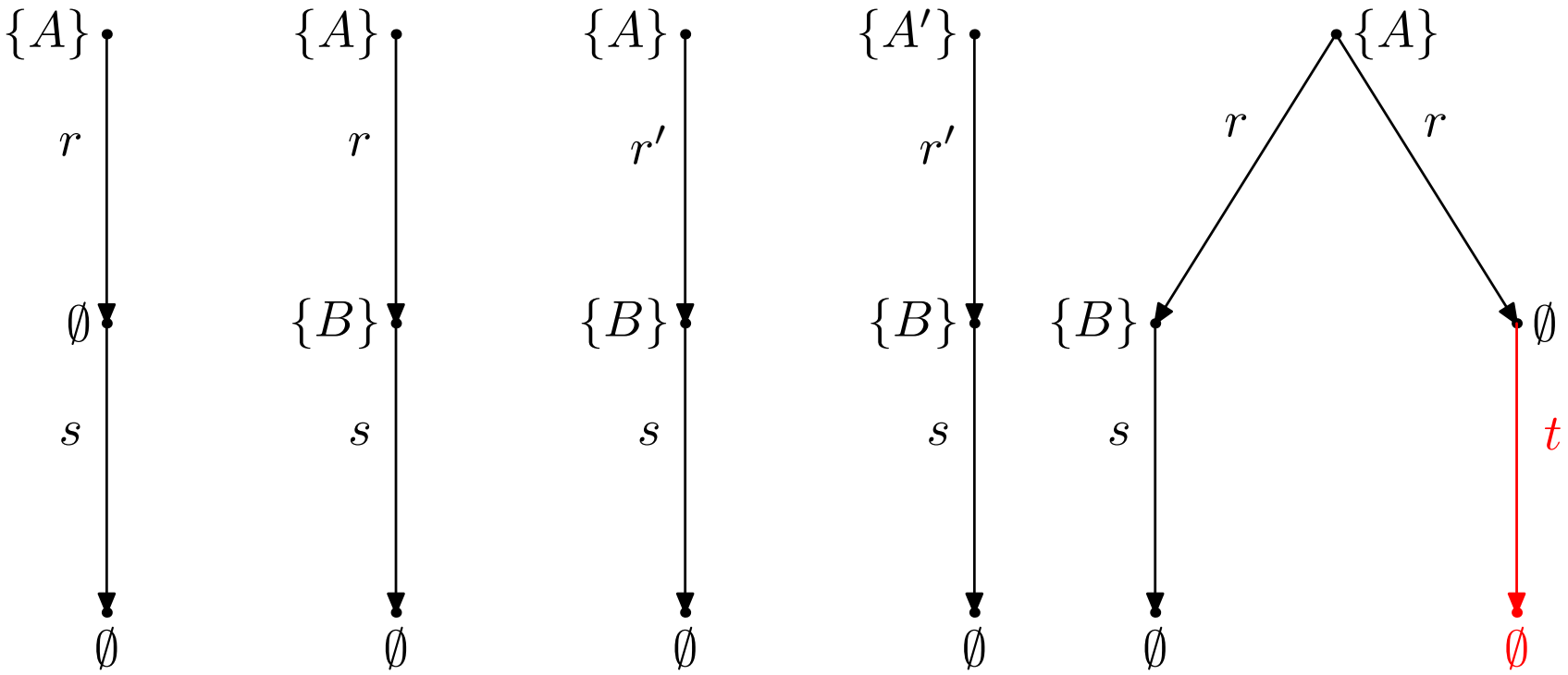
$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



attach subtree

A refinement example

$$\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$$



attach subtree ✓

$\mathcal{EL}$ operator - background knowledge

- introduced operator uses concept hierarchy and role hierarchy
- further background knowledge can be used to reduce search space (of a learning algorithm) and ensure that trees are satisfiable
- further structures supported:
  - domain of roles
  - range of roles
  - disjoint concepts
- realised by extending the four operations

Outline

Benchmark Setup

- tested performance of operator outside of a learning algorithm
- random refinement chains: starting at $\top$, select a refinement randomly and apply operator again etc.
- tested 100 random refinement chains of length 8 on 2,2 GHz dual core machine with 4 GB RAM
- diverse set of $\mathcal{EL}$-trees obtained w.r.t. their size and structure, i.e. performance results should be conclusive
- optimisations: use of local simulation updates, inference caching

Benchmark Results

name	logical axioms	ρ av. time (in ms)	ρ per ref. (in ms)	reasoning time (%)
GENE	42656	167.2	0.14	68.4
CTON	33203	76.2	0.08	5.1
GALEN	4940	3.5	0.21	37.1
PROCESS	2578	193.6	0.16	27.2
TRANSPORT	1157	164.4	0.09	5.9
EARTHREALM	931	407.4	0.17	23.2
TAMBIS	595	141.6	0.09	1.5

- benchmark results on ontologies from the TONES repository
- time per refinement below one millisecond

Outline

Conclusions

- $\mathcal{EL}$ allows for ideal refinement (but expressive DLs and OWL do not)
- development of an operator, which makes heavy use of background knowledge
- operator works efficiently even on large ontologies (e.g. through local simulation updates, caching)
- prototypical learning algorithm implementation

Future Work

- explore whether language can be extended while maintaining ideality, e.g. hasValue constructor, primitive negation, inverse roles, ...
- work on learning algorithm
- use cases:
  1. light weight (learn single $\mathcal{EL}$ tree): ontology engineering in $\mathcal{EL}$ (OWL 2 EL++) knowledge bases, "friend finder", recommendations
  2. learn disjunction of $\mathcal{EL}$ trees: many ILP problems, e.g. carcinogenesis

Thank you for your attention!

Learning Algorithm

- learning problem: given positive and negative examples (objects in $\mathcal{K}$), try to find a complex concept C covering all positives and no negatives
- learning algorithm is still preliminary work (not part of ILP09 article)
- algorithm for learning disjunctions of $\mathcal{EL}$ trees
- implementation in DL-Learner open source framework
- test on carcinogenesis problem (74000 axioms, 337 examples)

Learning Algorithm

approach/tool	accuracy
Aleph Ensembles	59.0% to 64.5%
Disjunctive $\mathcal{EL}$ trees	62.6% $\pm$ 9.0%
Boosted Weak ILP	61.1%
Weak ILP	58.7%
Aleph DTD 0.7	57.9% $\pm$ 9.8%
Aleph RRR 0.9	57.6% $\pm$ 6.4%
Aleph DTD 0.9	56.2% $\pm$ 9.0%
Aleph RRR 0.7	54.8% $\pm$ 9.0%

runtime: 5.3 $\pm$ 2.0 seconds

(caution: ongoing work)