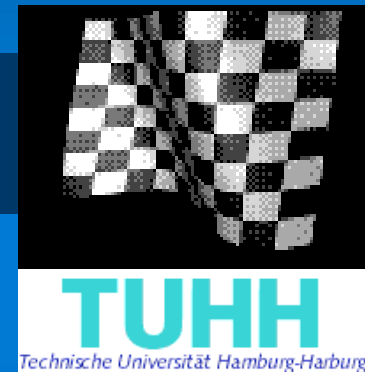


The Ins and Outs of Racer



Ralf Möller

Technical University Hamburg-Harburg

Collaboration with:

Volker Haarslev

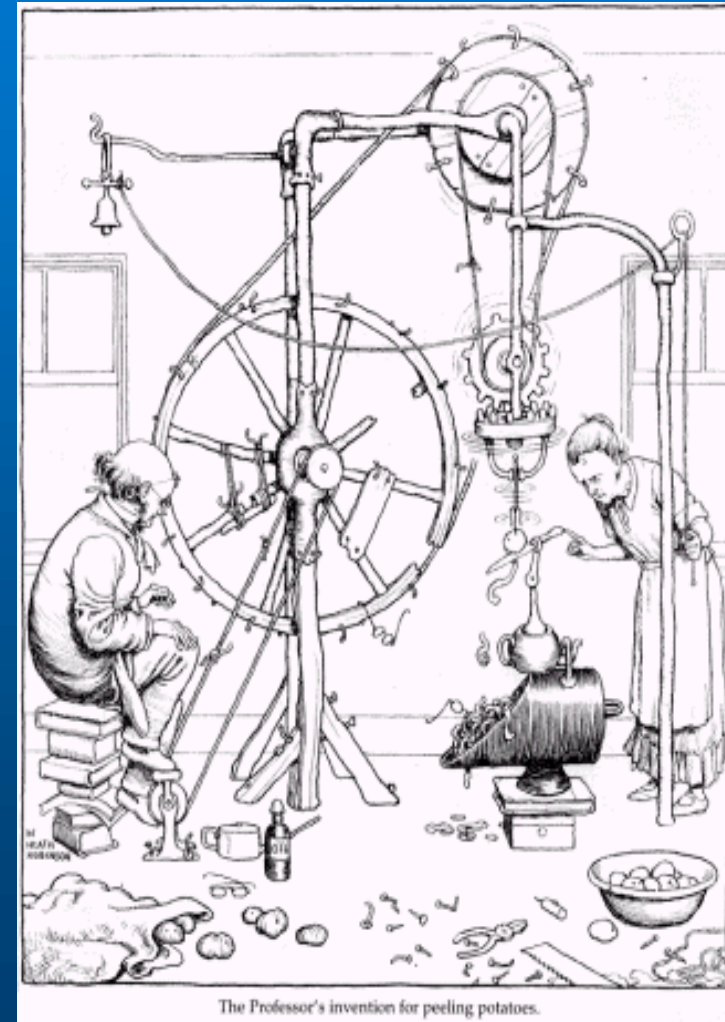
Concordia University, Montreal

Racer

- **Logic SHIQ(D_n)-**
 - ALC, transitive roles, role hierarchies, inverses, qualified number restrictions
 - Concrete domains without feature paths
- **T-boxes (with GCIs)**
- **A-boxes (each associated with a T-box)**
 - Set of assertions of the form
 - $i:C$
 - $(i, j):R$

Views on the Architecture

- Racer as a tool for ontology management (T-box)
- Racer as a testbed for AI/DB research (T-box, A-box, ...)
- Racer as part of applications (interfaces)



Inference Problems

- **T-box**

- Parents, Children, Synonyms

- **A-box**

- Consistency

- Instance test $i :? C$

- Instance retrieval $\{i \mid i \text{ mentioned in } A, i :? C\}$

Overview of the Talk

- **The engines under the hood**
- **Supporting classification**
 - E.g. for ontology development
- **Supporting instance retrieval**
 - E.g. for NL interpretation

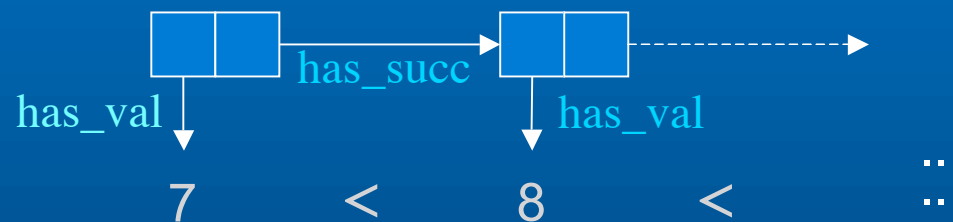
The Engines Under the Hood ...

- **Tableau prover for deciding A-box consistency for the logic SHIQ(D_n)** [Hor-Sat-Tob, BaHo, HaMo]
- **Additional provers for various algebraic structures and formalisms [...]**

\mathbb{N}	linear inequations with order constraints and integer coefficients
\mathbb{Z}	interval constraints
\mathbb{R}	linear inequations with order constraints and rational coefficients
\mathbb{C}	nonlinear multivariate inequations with integer coefficients
Strings	equality and inequality

Concrete Domains

- What cannot be expressed in Racer at the conceptual level due to syntax restrictions?



- No restrictions on A-box structures

Natural Numbers (Cardinals)

```
(define-concrete-domain-attribute year :type cardinal)
(define-concrete-domain-attribute days-in-month :type cardinal)

(implies Month (and (>= days-in-month 28) (<= days-in-month 31)))

(equivalent month-inleapyear
  (and Month
    (divisible year 4)
    (or (not-divisible year 100)
        (divisible year 400))))

(equivalent February
  (and Month
    (<= days-in-month 29)
    (or (not month-inleapyear)
        (= days-in-month 29))
    (or month-inleapyear
        (= days-in-month 28))))
```

Constraints and Queries

```
(instance feb-2003 February)
(constrained feb-2003 year-1 year)
(constrained feb-2003 days-in-feb-2003 days-in-month)
(constraints (= year-1 2003))
```

```
(instance feb-2000 February)
(constrained feb-2000 year-2 year)
(constrained feb-2000 days-in-feb-2000 days-in-month)
(constraints (= year-2 2000))
```

- (concept-instances month-in-leapyear)
- (concept-instances (not month-in-leapyear))
- (constraint-entailed? (= days-in-feb-2000 29))
- (constraint-entailed? (<=> days-in-feb-2003 29))

There's more to Racer ...

- ... than a collection of optimized (inference) engines
- For many inference problems:
- Motto: Avoid using the optimized tableaux prover (and the other provers) at all!
- So, exploit given information,
 - (implies a (and b c (some r D)))
- ... and do not compute anything twice
- Nevertheless: the tableau prover is important

Some Optimizations

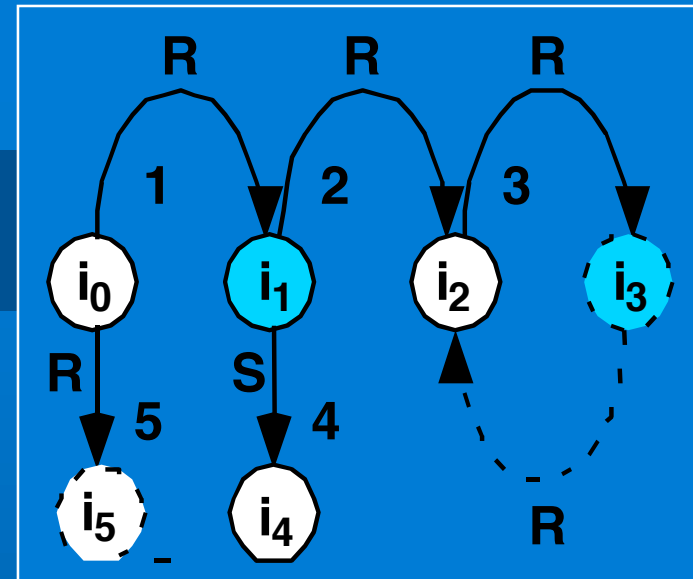
- **Inspired by FaCT [Horrocks]
(but Racer does it for A-boxes)**
 - **Dependency-directed backtracking**
 - **Semantic branching, BCP**
 - **Caching, model merging**
 - **notion of a pseudo model**
 - **Optimized blocking
(partially implemented in Racer)**

Optimizations specific to Racer

- **SAT and ASAT reasoning**
 - Language-adaptive selection of applicable optimization techniques
 - Dependency-directed backtracking (also for concrete domain reasoning)
 - Extended caching (in particular, satisfiable concepts)
 - Process qualified number restrictions with
 - Simplex procedure (concept consistency)
 - Signature calculus
 - Incremental constraint solving (mostly)

Pitfalls in Caching

- Assume a pseudo model for concept D is cached in step 7, i.e. D is satisfiable
- After step 8 concept C is marked as unsatisfiable
- The cache entry for D is still marked as satisfiable
- After backtracking to step 2, we proceed with step 9 and erroneously cache a pseudo model for concept E, i.e. E is satisfiable
- The cache entry of D depends on C and must be invalidated if C is marked as unsatisfiable



$$E \sqsubseteq (\exists R.C) \sqcup (\exists R.D)$$

$$C \sqsubseteq (\exists R.D) \sqcap (\exists S.X) \sqcap \forall S.(\neg X \sqcap A)$$

$$D \sqsubseteq \exists R.C$$

1. $\{i_0 : E\}$
2. $\{i_0 : E, i_0 : \exists R.C\} \vee \{i_0 : E, i_0 : \exists R.D\}$
3. $\{i_1 : C\}$
4. $\{i_1 : C, i_1 : \exists R.D, i_1 : \exists S.X, i_1 : \forall S. \neg X \sqcap A\}$
5. $\{i_2 : D\}$
6. $\{i_2 : D, i_2 : \exists R.C\}$
7. $\{i_3 : C\}$
8. $\{i_4 : X, i_4 : A, i_4 : \neg X\}$
9. $\{i_5 : D\}$

Dealing with Qualifying Number Restrictions

- (-> ALCQ [Baader-Hollunder])
- Additional level of nondeterminism
- Basically: for each ($\leq n R C$)
try either C or $\sim C$ for each filler of R
where $\sim C$ is the negation normal
form of (not C)
- Merge fillers as appropriate

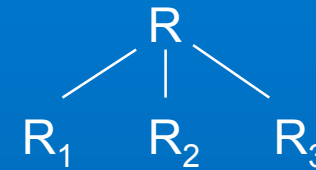
Standard Tableau Rules

- **At least rule:** $\Box_{\geq n} R.C$
 - if necessary, create n distinct R -successors which become members of C
- **At most rule:** $\Box_{\leq n} R.C$
 - if there exist more than n R -successors which are members of C , then merge two (non-distinct) of these R -successors
- **Choose rule:**
 - if there exists an R -successor whose membership for C or $\sim C$ is unknown, determine whether this successor is a member of C or $\sim C$

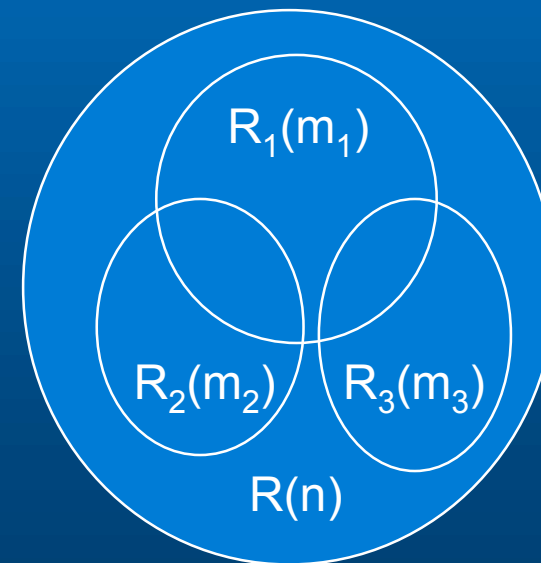
- **Treatment of numbers is highly inefficient**
- **Individual-wise computation of set cardinalities**
- **No representation of sets**
- **Blind search tries to fulfill at least and at most restrictions**

Standard Tableaux Rules: Example

- Assume $n=3$, $m_1=2$, $m_2=2$, $m_3=2$
- $(a,b_1):R_1$, $(a,b_2):R_1$, $(a,c_1):R_2$,
 $(a,c_2):R_2$, $(a,d_1):R_3$, $(a,d_2):R_3$,
 $c_1:C$, $c_2:C$, $d_1:\neg C$, $d_2:\neg C$,
 $b_1 \neq b_2$, $c_1 \neq c_2$, $d_1 \neq d_2$
- 6 R-successors, only 3 allowed
- Try to replace b_1 with c_1 , or c_2 , or
 ...
- After replacing b_1 with c_1 :
 $(a,c_1):R_1$, $(a,b_2):R_1$, $(a,c_1):R_2$,
 $(a,c_2):R_2$, $(a,d_1):R_3$, $(a,d_2):R_3$,
 $c_1:C$, $c_2:C$, $d_1:\neg C$, $d_2:\neg C$,
 $c_1 \neq b_2$, $c_1 \neq c_2$, $d_1 \neq d_2$
- ...
- Eventually the calculus recognizes the unsatisfiability



$$\exists_{<n} R \sqcap \exists_{>m_1} R_1 \sqcap \exists_{>m_2} R_2 \sqcap \exists_{\geq m_3} R_3 \sqcap \forall R_2 . C \sqcap \forall R_3 . \neg C$$



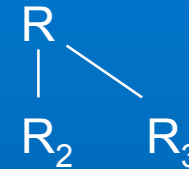
Idea: Use Gomory Algorithm

- Inspired by work of [Ohlbach]
- Compute system of linear equations
- Integrated into tableau prover for dealing with concept consistency
- Simplex procedure cannot be (naively) applied for deciding full A-box consistency
- Another idea: Use placeholders
 - > Signature Calculus

Computation with Placeholders...

- ...cannot be done in a simple way:

$$\exists_{\leq n} R \sqcap \exists_{\geq m_2} R_2 \sqcap \exists_{\geq m_3} R_3 \sqcap \forall R_2 . C \sqcap \forall R_3 . \neg C$$



- Automatic generation of required placeholders for subroles

- New kind of constraint: $(ind_1, ind_2): \langle n, \{R, S, \dots\} \rangle$

- $(i, j): \langle m_2, \{R_2\} \rangle$, $(i, k): \langle m_3, \{R_3\} \rangle$

Signature

- $j:C$, $k:\neg C$

- Signature merge rule:

- $(i, l): \langle 1, \{R_2, R_3\} \rangle$, $(i, j): \langle m_2 - 1, \{R_2\} \rangle$, $(i, k): \langle m_3 - 1, \{R_3\} \rangle$

Eliminated if zero

- or

- $(i, l): \langle p + 1, \{R_2, R_3\} \rangle$, $(i, j): \langle m_2 - 1, \{R_2\} \rangle$, $(i, k): \langle m_3 - 1, \{R_3\} \rangle$

- $l:C$, $l:\neg C$

Additional Rules

- Choose rule
- Additional clash triggers

Dealing with Inverse Roles and A-boxes

- Use of the A-box reasoning kernel to deal with inverse roles
- Blocking (-> see SHIQ-Papers [Horr-Satt-Tob])
 - dynamic blocking, pairwise blocking
- No trace technique, additional role constraints
- If something is "propagated back", all constraints are recomputed (within backtracking context!)
- Caveat: technique relies on caching
- Caching techniques still have to be devised for inverse roles

Dependencies and Pseudo Models

- If an A-box is consistent, generate a pseudo model for each individual
- If an A-box is inconsistent, collect those assertions that were involved in a clash in all branches of the search tree

Concrete domain reasoners

- **Simplex algorithm (incremental)**
 - for linear inequations with orders over the reals
 - for min/max restrictions over the integers
- **Gomory algorithm**
 - for linear inequations with orders over cardinals
- **Groebner Bases, Buchberger algorithm**
 - for multivariate non-linear inequations over complex numbers
- **String (in)equation solver**

Optimizations for Concrete Domains

- Incremental constraint solving for R
- Dependency-directed backtracking
- Keep constraint systems minimal
- Avoiding copies of constraint systems during backtracking
- Use Simplex as preprocessor for N

Overview of the Talk

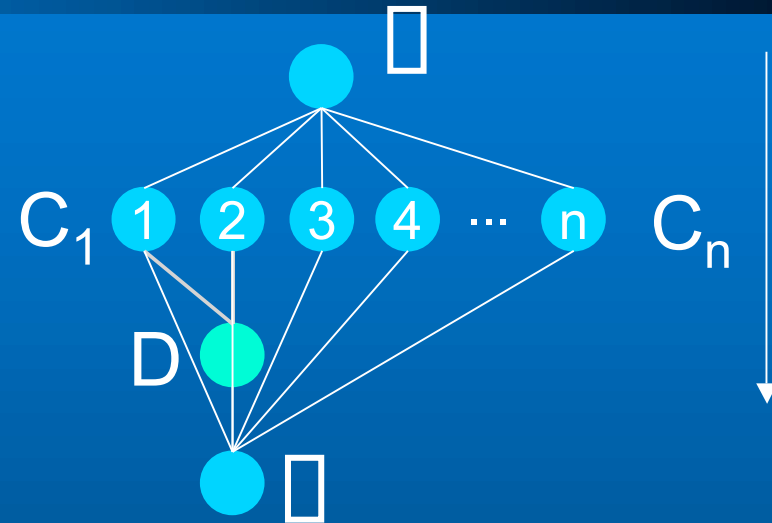
- The engines under the hood
- **Supporting classification**
 - E.g. for ontology development
 - Finding synonyms of *bottom*
- Supporting instance retrieval

Ontology Development: T-boxes

- **Optimizations of**
 - KRIS [Baader et al.] (taxonomy, lazy unfolding)
 - FaCT [Horrocks]
(GCI absorption, model merging)
- **Optimizations specific to Racer:**
 - Different transformation of general axioms
 - domain and range restrictions
 - lazy unfolding of negated atomic concepts
 - Determine classification order
(topological sorting)
 - Clustering of nodes

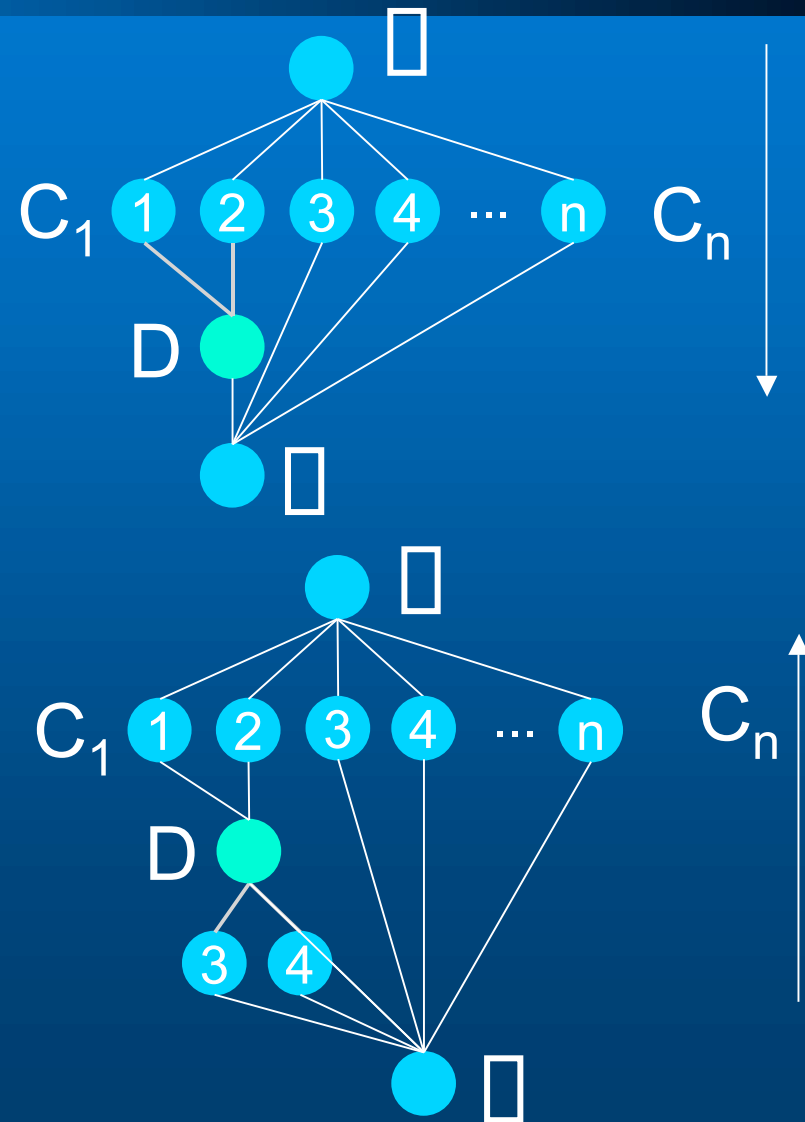
TBox Classification: Inserting a Concept

- Insert new concept D into existing taxonomy w.r.t subsumption relationship
- 1. Top-search phase
 - traverse from top
 - determine parents of D
 - C_1 and C_2
 - $\text{SAT}(\neg C_1 \sqcap D), \dots, \text{SAT}(\neg C_n \sqcap D)$
- 2. Bottom-search phase
 - traverse from bottom
 - determine children of D
 - C_3 and C_4
 - $\text{SAT}(C_1 \sqcap \neg D), \dots, \text{SAT}(C_n \sqcap \neg D)$



TBox Classification: Inserting a Concept

- Insert new concept **D** into existing taxonomy w.r.t subsumption relationship
- 1. Top-search phase
 - traverse from top
 - determine parents of **D**
 - C_1 and C_2
 - $\text{SAT}(\neg C_1 \sqcap D), \dots, \text{SAT}(\neg C_n \sqcap D)$
- 2. Bottom-search phase
 - traverse from bottom
 - determine children of **D**
 - C_3 and C_4
 - $\text{SAT}(C_1 \sqcap \neg D), \dots, \text{SAT}(C_n \sqcap \neg D)$

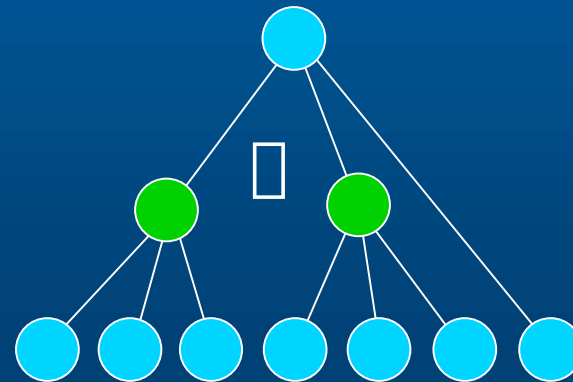
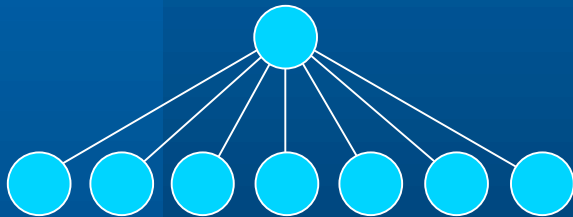


Topological Sorting

- GCI absorption yields only primitive concept definitions
- Definition order
- Bottom-search phase can be omitted
- But: Beware of cycles
- Refers-to relation
- Sets of concept names
- Quasi definition order
- Serialization of the partial order

Clustering

- Introduce virtual concepts (buckets)
- Reduce the no. of children
- $\square = 10$: max no. children without bucket
- $\square = 15$: max no. of buckets per concept
- Possibly: restructuring necessary



Domain and Range Restrictions

- $A \sqsubseteq C$ (all $x . x$: (or (not A) B))
- Lazy Unfolding: $A \sqsubseteq C, A = C$
- Generalized Lazy Unfolding:
 - (some R. top) $\sqsubseteq C$ **domain restriction**
 - top \sqsubseteq (all R. C) **range restriction**
- To be considered
 - for model merging
 - for the refers-to relation

Hard for Racer ...

... and probably other tableau reasoners

- **Qualified domain and range restrictions**
 - (some R. D) \sqsubseteq C
 - D \sqsubseteq (all R. C)
- **Cycles**
(even if no meta constraints remain)
- **Inverse roles**
(no caching,
more complex blocking condition)

Overview of the Talk

- The engines under the hood
- Supporting classification
- **Supporting instance retrieval**
 - E.g. for NL interpretation, Semantic Web, etc.

Optimizations specific to Racer

- **A-box reasoning**
 - graph transformation
 - fast non-instance test based on pseudo models for concepts and individuals
 - dependency-driven divide-and-conquer for instance checks
 - fast candidate elimination by subsumption-based caching

Instance retrieval

- Given an A-box A , a T-box T , and a concept C , find those individuals i mentioned in A for which it can be shown that $i \sqsubseteq C$ for all models I of T .
- Different types of applications
 - One single "static" A-box
lots of queries
 - Many A-boxes wrt. a single T-box,
A-boxes computed on the fly,
few queries for each A-box, result set small
(example: NL application)

Linear instance retrieval

- Collect each i mentioned in A s.t. $\text{instance?}(i,C)$
- $\text{instance?}(i,C) = \neg \text{ASAT}(A \sqcup \{i:\neg C\})$
- "Expensive" ASAT test
- Guards required
 - individual model merging possible, $\text{individual-concept}(i)$
 - $\text{subsumes?}(\neg C, \text{individual-concept}(i))$
- "Expensive" ASAT test only if guards do not indicate non-instance result for i

Binary instance retrieval

- **Observation:**
 - One individual considered at a time
- **New idea: consider sets of individuals at a time**
 - Take a set of candidates $\{i, j, k, \dots\}$ (which are assumed to be instances of C) and check the result of $ASAT(A \sqcup \{i:\neg C, j:\neg C, k:\neg C, \dots\})$
 - If "yes" \rightarrow all candidates are ruled out
 - If "no" \rightarrow partition the set of candidates (binary partition) and recurse until a singleton set of candidates is derived

Dependency-based instance retrieval

- If $ASAT(A \sqcup \{i:\neg C, j:\neg C, k:\neg C, \dots\})$ returns "no"
- and during all attempts to construct a completion, there is only one individual involved in a clash according to dependency tracking,
 - then this individual can be removed from the set of candidates. It is a member of the result set.
 - else if more than one individual is involved: partition the set of candidates and recurse until a singleton candidate set is derived

Index-based instance retrieval

- **Index:**

- **associated_inds:**

- Set of concept names $\rightarrow P(\text{Set of inds})$

```
if  $\exists N \in CN : N \in \text{synonyms}(C)$  then
  return  $\bigcup_{D \in \text{descendants}(C)} \text{associated\_inds}(D)$ 
else
  known_results :=  $\bigcup_{D \in \text{descendants}(C)} \text{associated\_inds}(D)$ 
  candidates :=  $\bigcup_{P \in \text{parents}(C)} (\bigcup_{D \in \text{descendants}(P)} \text{associated\_inds}(D))$ 
  return known_results  $\cup$  instance_retrieval(C, A, candidates \ known_results)
end if
```

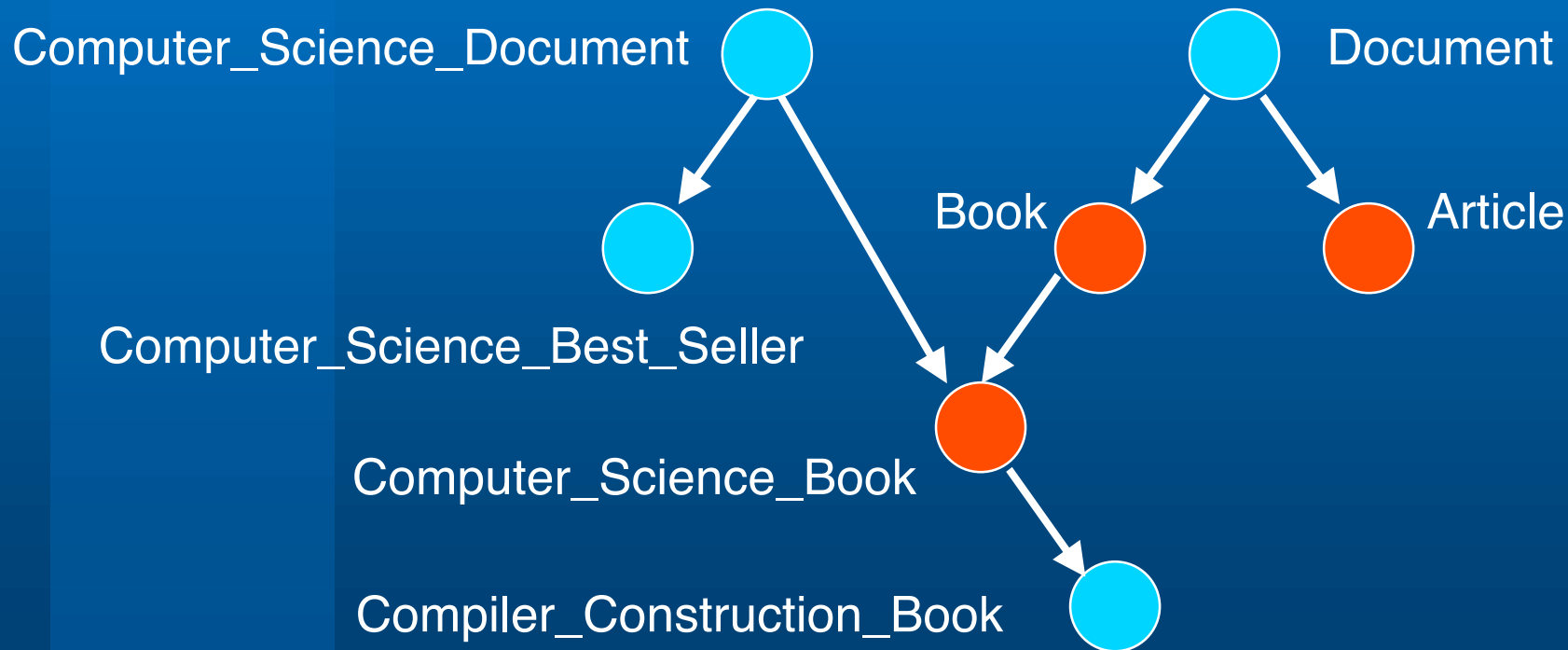
- **Index associated_inds is expensive to compute**

Index computation:

- **One-individual-at-a-time approach**
- **Sets-of-individuals-at-a-time approach**

Partial Indexing

- Racer Server running with **subsumption-based instance retrieval** enabled



Partial Indexing

- **Index computation** from
 - Initial A-box instance assertions
 - Previous instance retrieval queries
 - Previous queries for the direct types of an individual

Retrieval: Exploiting the Index

- Find **known results**
 - Collect individuals from descendants
- Find **candidates**
 - Collect individuals from ancestors
- Find **non-candidates**
 - Collect those candidates for which the direct-types are known and less specific than query concept C
 - Collect those candidates that previously did not qualify to to instances of an ancestor concept

Conclusions

- **Racer is more than a tableau prover**
- **Tableau prover returns more than "satisfiable" or "not-satisfiable"**
- **Dealing with A-boxes causes overhead for pure concept consistency**
- **Many optimizations:
No single trick does the job!**
- **Extensions become harder and harder...**

Future Work



- Optimizations for filler retrieval
- Feature chains in predicate exists restrictions (with restr. on GCIs)
- UNA optional
- Nominals
- Incremental A-box reasoning
- Output of models
- Acyclic role axioms