Knowledge Representation and Processing

Florian Rabe (for a course given with Michael Kohlhase)

Computer Science, University Erlangen-Nürnberg, Germany

Summer 2020

Administrative Information

Format

Zoom

- lectures and exercises via zoom
- participants muted by default for simplicity
- interaction strongly encouraged

  We don't want to lecture —
  we want to have a conversation during which you learn

- let's try out zoom
  - use reactions to say yes no, ask for break etc.
  - feel free to annotate my slides
  - talk in the chat

Recordings

- maybe prerecorded video lectures or recorded zoom meeting
- to be decided along the way

Background

Instructors

- Prof. Dr. Michael Kohlhase
  Professor of Knowledge Representation and Processing
- PD Dr. Florian Rabe — same research group

Course

- This course is given for the first time
- Always a little bit of an experiment — cutting edge vs. unpolished
- Could become signature course of our research group — same name!

Prerequisites

Required

- basic knowledge about formal languages, context-free grammars
  but we'll do a quick revision here

Helpful

- Algorithms and Data Structures mostly as a contrast to this lecture
- Basic logic we'll revise it slightly differently here
- all other courses as examples of how knowledge pervades all of CS

General

- Curiosity this course is a bit unusual
- Interest in big picture
  this course touches on lots of things from all over CS

Examination and Grading

Suggestion

- grade determined by single exam
- written or oral — depends on number of students
- some acknowledgment for practical exercises

to be finalized next week

Exam-relevant

- anything mentioned in notes
- anything discussed in lectures

neither is a superset of the other!

Materials and Exam-Relevance

Textbook

- does not exist
- normal for research-near specialization courses

Notes

- textbook-style but not as comprehensive
- developed along the way

Slides

- not comprehensive
- used as visual aid, conversation starters

Communication

Open for questions

- open door policy in our offices — if the lockdown ever ends
- always room for questions during lectures
- for personal questions, contact me during/after lecture or by email
- forum at `https://fsi.cs.fau.de/forum/154-Wissensrepraesentation-und-Verarbeitung`

Materials

- official notes and slides as pdf: `https://kwarc.info/teaching/WuV/`
  will be updated from time to time
- watch me prepare the materials: `https://github.com/florian-rabe/Teaching/tree/master/WuV`
  pull requests and issues welcome

Exercises

Learning Goals

- Get acquainted with state of the art of practice
- Try out real tools

Homeworks

- one major project as running example
- homeworks building on each other

build one large knowledge-based system

details on later slides

Overview and Essential Concepts

Representation and Processing

Common pairs of concepts:

Representation	Processing
Static	Dynamic
Situation	Change
Be	Become
Data Structures	Algorithms
Set	Function
State	Transition
Space	Time

Data and Knowledge

2 × 2 key concepts

Syntax	Data
Semantics	Knowledge

- Data: any object that can be stored in a computer
  Example: $((49.5739143, 11.0264941), "2020-04-21T16:15:00CEST")$
- Syntax: a system of rules that describes which data is **well-formed**
  Example: "a pair of (a pair of two IEEE double precision floating point numbers) and a string encoding of a time stamp"
- Semantics: system of rules that determines the meaning of well-formed data
- Knowledge: combination of some data with its syntax and semantics

Knowledge is Elusive

Representation of key concepts

- Data: using primitive objects
  implemented as bits, bytes, strings, records, arrays, ...
- Syntax: (context-free) grammars, (context-sensitive) type systems
  implemeted as inductive data structures
- Semantics: functions for evaluation, interpretation, of well-formed data
  implemented as recursive algorithms on the syntax
- Knowledge: elusive
  emerges from applying and interacting with the semantics

Semantics as Translation

- Knowledge can be captured by a higher layer of syntax
- Then semantics is translation into syntax

Data syntax	Semantics function	Knowledge syntax
SPARQL query	evaluation	result set
SQL query	evaluation	result table
program	compiler	binary code
program expression	interpreter	result value
logical formula	interpretation in a model	mathematical object
HTML document	rendering	graphics context

Heterogeneity of Data and Knowledge

- Capturing knowledge is difficult
- Many different approaches to semantics
  - fundamental formal and methodological differences
  - often captured in different fields, conferences, courses, languages, tools
- Data formats equally heterogeneous
  - ontologies
  - programs
  - logical proofs
  - databases
  - documents

Challenges of Heterogeneity

Challenges

- collaboration across communities
- translation across languages
- conversion between data formats
- interoperability across tools

Sources of problems

- interoperability across formats/tools major source of
  - complexity
  - bugs
- friction in project team due to differing preferences, expertise
- difficult choice between languages/tools with competing advantages
  - reverting choices difficult, costly
  - maintaining legacy choices increases complexity

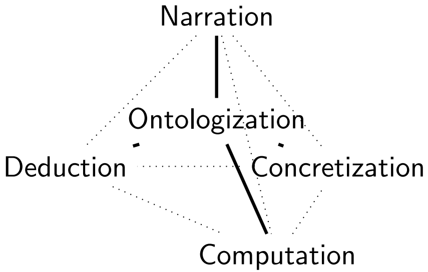
Aspects of Knowledge

- Tetrapod model of knowledge   active research by our group
- classifies approaches to knowledge into five aspects

Aspect	KRLs (examples)
ontologization	ontology languages (OWL), description logics (ALC)
concretization	relational databases (SQL, JSON)
computation	programming languages (C)
deduction	logics (HOL)
narration	document languages (HTML, LaTeX)

Relations between the Aspects

Ontology is distinguished: capture the knowledge that the other four aspects share



Complementary Advantages of the Aspects

Aspect	objects	characteristic		
		advantage	joint advantage of the other aspects	application
ded.	formal proofs	correctness	ease of use	verification
comp.	programs	efficiency	well-definedness	execution
concr.	concrete objects	tangibility	abstraction	storage/retrieval
narr.	texts	flexibility	formal semantics	human understanding

Aspect pair	characteristic advantage
ded./comp.	rich meta-theory
narr./conc.	simple languages
ded./narr.	theorems and proofs
comp./conc.	normalization
ded./conc.	decidable well-definedness
comp./narr.	Turing completeness

Structure of the Course

Aspect-independent parts

- general methods that are shared among the aspects
- to be discussed as they come up

Aspects-specific parts

- one part (about 2 weeks) for each aspect
- high-level overview of state of the art
- focus on comparison/evaluation of the aspect-specific results

Structure of the Exercises

One major project

- representative for a project that a CS graduate might be put in charge of
- challenging heterogeneous data and knowledge
- requires integrating/combining different languages, tools

unique opportunity in this course because knowledge is everywhere

Concrete project

- develop a univis-style system for a university
- lots of heterogeneous knowledge
  - course and program descriptions
  - legal texts
  - websites
  - grade tables
  - transcript generation code
- build a completely functional system applying the lessons of the course

Ontological Knowledge

Components of an Ontology

8 main declarations

- **individual** — concrete objects that exist in the real world, e.g., "Florian Rabe" or "WuV"
- **concept** — abstract groups of individuals, e.g., "instructor" or "course"
- **relation** — binary relations between two individuals, e.g., "teaches"
- **properties** — binary relations between an individuals and a concrete value (a number, a date, etc.), e.g., "has-credits"
- **concept assertions** — the statement that a particular individual is an instance of a particular concept
- **relation assertions** — the statement that a particular relation holds about two individuals
- **property assertions** — the statement that a particular individual has a particular value for a particular property
- **axioms** — statements about relations between concepts, e.g., "instructor" $\sqsubseteq$ "person"

Divisions of an Ontology

Abstract vs. concrete

- TBox: concepts, relations, properties, axioms
  everything that does not use individuals
- ABox: individuals and assertions

Named vs. unnamed

- Signature: individuals, concepts, relations, properties
  together called entities or resources
- Theory: assertions, axioms

Comparison of Terminology

Here	OWL	Description logics	ER model	UML	semantics via logics
individual	instance	individual	entity	object, instance	constant
concept	class	concept	entity-type	class	unary predicate
relation	object property	role	role	association	binary predicate
property	data property	(not common)	attribute	field of base type	binary predicate

domain	individual	concept
type theory, logic	constant, term	type
set theory	element	set
database	row	table
philosophy[1]	object	property
grammar	proper noun	common noun

[1] as in `https://plato.stanford.edu/entries/object/`

Ontologies as Sets of Triples

Assertion	Triple		
	Subject	Predicate	Object
concept assertion	"Florian Rabe"	`is-a`	"instructor"
relation assertion	"Florian Rabe"	"teaches"	"WuV"
property assertion	"WuV"	"has credits"	7.5

Efficient representation of ontologies using RDF and RDFS standardized special entities.

Special Entities

RDF and RDFS define special entities for use in ontologies:

- "rdfs:Resource": concept of which all individuals are an instance and thus of which every concept is a subconcept
- "rdf:type": relates an entity to its type:
  - an individual to its concept (corresponding to `is-a` above)
  - other entities to their special type (see below)
- "rdfs:Class": special class for the type of classes
- "rdf:Property": special class for the type of properties
- "rdfs:subClassOf": a special relation that relates a subconcept to a superconcept
- "rdfs:domain": a special relation that relates a relation to the concepts of its subjects
- "rdfs:range": a special relation that relates a relation/property to the concept/type of its objects

Goal/effect: capture as many parts as possible as RDF triples.

Declarations as Triples using Special Entities

Assertion	Triple		
	Subject	Predicate	Object
individual	individual	"rdf:type"	"rdfs:Resource"
concept	concept	"rdf:type"	"rdf:Class"
relation	relation	"rdf:type"	"rdf:Property"
property	property	"rdf:type"	"rdf:Property"
concept assertion	individual	"rdf:type"	concept
relation assertion	individual	relation	individual
property assertion	individual	property	value
for special forms of axioms			
$c \sqsubseteq d$	c	"rdfs:subClassOf"	d
$\mathrm{dom}\, r \equiv c$	r	"rdfs:domain"	c
$\mathrm{rng}\, r \equiv c$	r	"rdfs:range"	c

An Example Ontology Language

see syntax of BOL in the lecture notes

Semantics as Translation

Example: Syntax of Arithmetic Language

Syntax: represented as formal grammar

$\boxed{\text{Numbers}}$

$N ::= 0 \mid 1$	literals
$\mid \quad N + N$	sum
$\mid \quad N * N$	product

$\boxed{\text{Formulas}}$

$F ::= N \doteq N$	equality
$\mid \quad N \leq N$	ordering by size

Implementation as inductive data type

Example: Semantics of Arithmetic Language

Semantics: represented as translation into known language

Problem: Need to choose a known language first
Here: unary numbers represented as strings
Built-in data (strings and booleans):

$$
\begin{array}{lcll}
S & ::= & \varepsilon & \text{empty} \\
 & | & \texttt{(Unicode)} & \text{characters} \\
B & ::= & \texttt{true} & \text{truth} \\
 & | & \texttt{false} & \text{falsity}
\end{array}
$$

Built-in operations to work on the data:

- concatenation of strings $S ::= \texttt{conc}(S, S)$
- replacing all occurrences of c in S_1 with S_2
  $S ::= \texttt{replace}(S_1, c, S_2)$
- equality test: $B ::= S_1 == S_2$
- prefix test: $B ::= \texttt{startsWith}(S_1, S_2)$

Example: Semantics of Arithmetic Language

Represented as function from syntax to semantics

- mutually recursive, inductive functions for each non-terminal symbol
- compositional: recursive call on immediate subterms of argument

For numbers n: semantics $[\![n]\!]$ is a string

- $[\![0]\!] = \varepsilon$
- $[\![1]\!] = "|"$
- $[\![m+n]\!] = \mathtt{conc}([\![m]\!], [\![n]\!])$
- $[\![m*n]\!] = \mathtt{replace}([\![m]\!], "|", [\![n]\!])$

For formulas f: semantics $[\![f]\!]$ is a boolean

- $[\![m \doteq n]\!] = [\![m]\!] == [\![n]\!]$
- $[\![m \leq n]\!] = \mathtt{startsWith}([\![n]\!], [\![m]\!])$

Semantics of BOL

Aspect	kind of semantic language	semantic language
deduction	logic	SFOL
concretization	database language	SQL
computation	programming language	Scala
narration	natural language	English

see details of each translation in the lecture notes

General Definition

A semantics by translation consists of

- syntax: a formal language I
- semantic language: a formal language L

  different or same aspect as I
- semantic prefix: a theory P in L

  formalizes fundamentals that are needed to represent I-objects
- interpretation: translates every I-theory T to an L-theory $P, [\![T]\!]$

Common Principles

Properties shared by all semantics of BOL

not part of formal definition, but best practices

- l-declaration translated to L-declaration for the same name
- ontologies translated declaration-wise
- one inductive function for every kind of complex l-expression
  - individuals, concepts, relations, properties, formulas
  - maps l-expressions to L-expressions
- atomic cases (base cases): l-identifier translated to L-identifier of the same name — or something very similar
- complex cases (step cases): compositional

Compositionality

Case for operator $*$ in interpretation function compositional iff interpretation of $*(e_1, \dots, e_n)$ only depends on on the interpretation of the e_i

$$[\![*(e_1, \dots, e_n)]\!] = [\![*]\!]([\![e_1]\!], \dots, [\![e_n]\!])$$

for some function $[\![*]\!]$

Example: ;-operator of BOL in translation to FOL

- translation: $[\![R_1; R_2]\!] = \exists m : \iota.[\![R_1]\!](x, m) \wedge [\![R_2]\!](m, y)$
- special case of the above via
  - $* =;$
  - $n = 2$
  - $[\![;]\!] = (p_1, p_2) \mapsto \exists m : \iota.p_1(x, m) \wedge p_2(m, y)$
- Indeed, we have $[\![R_1; R_2]\!] = [\![;]\!]([\![R_1]\!], [\![R_2]\!])$

Compositionality (2)

Translation compositional iff

- one translation function for each non-terminal all written $[\![-]\!]$
- each defined by one induction on syntax

  i.e., one case for production

  mutually recursive
- all cases compositional

Substitution theorem: a compositional translation satisfies

$$[\![E(e_1, \ldots, e_n)]\!] = [\![E]\!]([\![e_1]\!], \ldots, [\![e_n]\!])$$

for

- every expression $E(N_1, \ldots, N_n)$ with non-terminals N_i
- some function $[\![E]\!]$ that only depends on E

Compositionality (3)

$$[\![E(e_1,\ldots,e_n)]\!] = [\![E]\!]([\![e_1]\!],\ldots,[\![e_n]\!])$$

for every expression $E(N_1,\ldots,N_n)$ with non-terminals N_i

Now think of

- variable x_i of type N_i instead of non-terminal N_i
- $E(x_1,\ldots,x_n)$ as expression with free variables x_i of type N_i
- expressions e derived from N as expressions of type N
- $E(e_1,\ldots,e_n)$ as result of substituting e_i for x_i
- $[\![E]\!](x_1,\ldots,x_n)$ as (semantic) expression with free variables x_i

Then both sides of equations act on $E(x_1,\ldots,x_n)$:

- left side yields $[\![E(e_1,\ldots,e_n)]\!]$ by
  - first substitution e_i for x_i
  - then semantics $[\![-]\!]$ of the whole
- right side yields $[\![E]\!]([\![e_1]\!],\ldots,[\![e_n]\!])$ by
  - first semantics $[\![-]\!]$ of all parts
  - then substitution $[\![e_i]\!]$ for x_i

semantics commutes with substitution

Non-Compositionality

Examples

- deduction: cut elimination, translation from natural deduction to Hilbert calculus
- computation: optimizing compiler, e.g., loop unrolling
- concretization: query optimization, e.g., turning a WHERE of a join into a join of WHEREs,
- narration: ambiguous words are translated based on context

Typical sources

- subcases in a case of translation function
  - based on inspecting the arguments, e.g., subinduction
  - based on context
- custom-built semantic prefix

Type Systems

Breakout Question

Is this an improvement over BOL?

$\boxed{\text{Declarations}}$

D ::=	**individual** $\mathtt{ID} : C$	typed atomic individual
|	**concept** $\mathtt{ID}$	atomic concept
|	**relation** $\mathtt{ID} \subseteq C \times C$	typed atomic relation
|	**property** $\mathtt{ID} \subseteq C \times T$	typed atomic property

rest as before

Church vs. Curry Typing

	intrinsic	extrinsic
λ-calculus by	Church	Curry
type is	carried by object	given by environment
typing is a	function objects $\to$ types	relation objects $\times$ types
objects have	unique type	any number of types
types interpreted as	disjoint sets	unary predicates
type given by	part of declaration	additional axiom
example	**individual** "WuV":"course"	**individual** "Wuv",
		"WuV" `is-a` "course"
examples	SFOL, SQL	OWL, Scala, English
	most logics, functional PLs	ontology, OO,
		natural languages
	many type theories	set theories

Type Checking

	intrinsic	extrinsic
type is	carried by object	given by environment
typing is a	function objects $\to$ types	relation objects $\times$ types
objects have	unique type	any number of types
type given by	part of declaration	additional axiom
example	**individual** "WuV":"course"	**individual** "Wuv", "WuV" `is-a` "course"
type inference for x	uniquely infer A from x	find minimal A with $x : A$
type checking	inferred=expected	prove $x : A$
subtyping $A <: B$	cast from A to B	$x : A$ implies $x : B$
typing decidable	yes unless too expressive	no unless restricted
typing errors	static (compile-time)	dynamic (run-time)
advantages	easy unique type inference	flexible allows subtyping

Curry Typing in BOL

language	objects	types	typing relation
Syntax	individuals	concepts	i `is-a` c
Semantics in			
FOL	type ι	predicates $c \subseteq \iota$	$c(i)$ true
SQL	table Individuals	tables containing ids	id of i in table c
Scala	String	hash sets of strings	c.contains(i)
English	proper nouns	common nouns	"i is a c" is true

Subtyping

Subtyping works best with Curry Typing

- explicit subtyping as in $\mathbb{N} <: \mathbb{Z}$
- comprehension/refinement as in $\{x : \mathbb{N} | x \neq 0\}$
- operations like union and intersection on types
- inheritance between classes, in which case subclass = subtype
- anonymous record types as in $\{x : \mathbb{N}, y : \mathbb{Z}\} <: \{x : \mathbb{N}\}$

General Definition

A **type system** consists of

- a collection, whose elements are called **objects**,
- a collection, whose elements are called **intrinsic types**,
- a function assigning to every object x its **intrinsic type** I, in which case we write $x : I$,
- for some intrinsic types I
  - an intrinsic type E_I
  - a relation $\in_I$ between objects with intrinsic types I and E_I, called the **extrinsic typing** relation for I.

Examples

System	intrinsic types	E_I	$\in_I$
FOL	one per type	none	none
Scala	*AnyRef*, *Class*	$E_{Any} = Class$	$\in_{Any}=$ `isInstanceOf`
BOL	*Ind*, *Conc*	$E_{Ind} = Conc$	$\in_{Ind}= \mathtt{is} - \mathtt{a}$
set theory	*Set*, *Prop*	$E_{Set} = Set$	$\in_{Set}=\in$