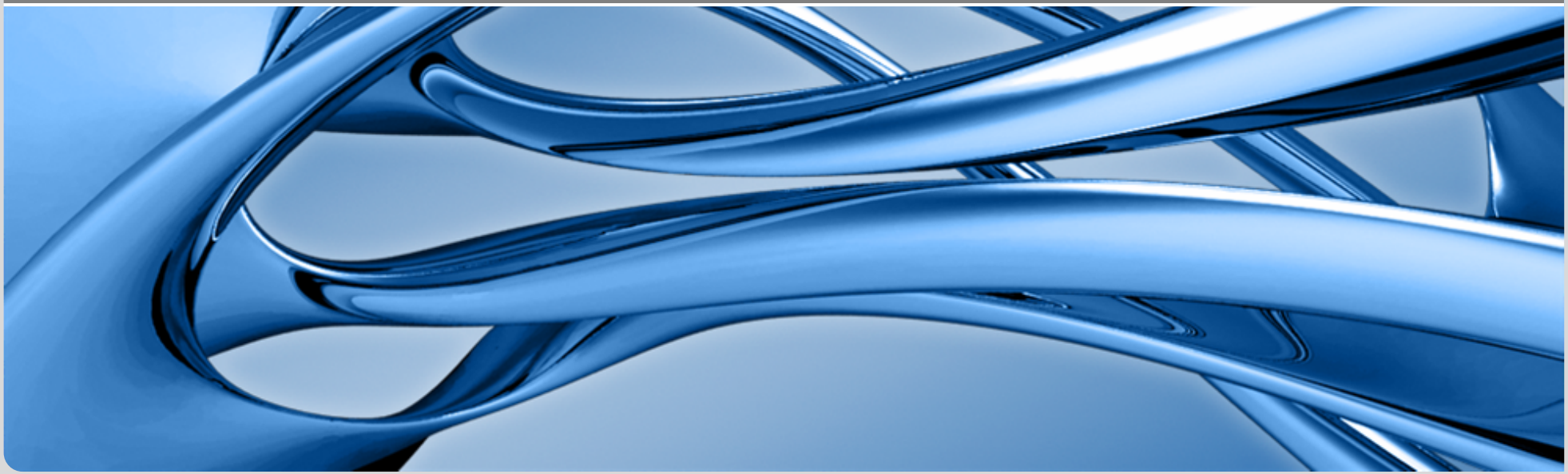
Karlsruhe Institute of Technology

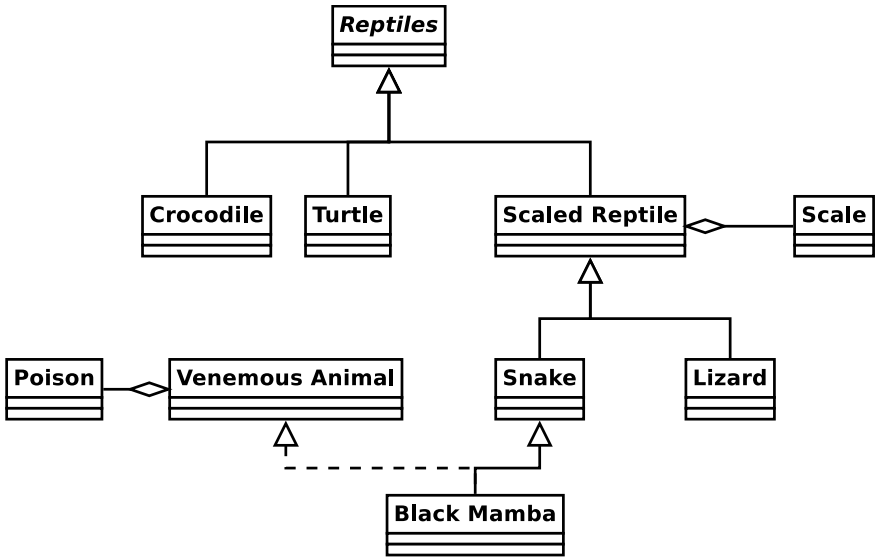
Semantic Objectoriented Programming (SOOP)

Florian Weber, Andreas Bihlmaier, Heinz Wörn

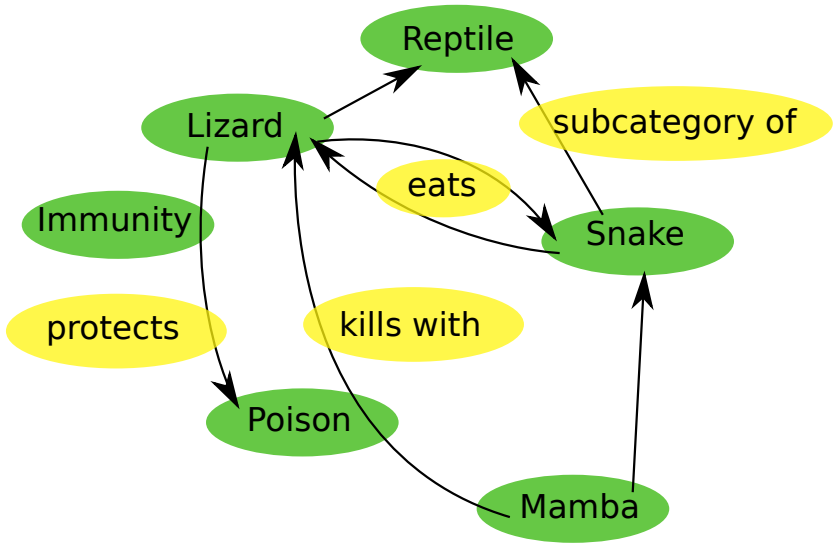
Institute for Anthropomatics and Robotics - Intelligent Process Control and Robotics (IPR)



Motivation



VS



Introduction

- OOP: Everything is an object, functionality resides in methods.
- Declarative Programming: Describe properties of a solution, let the runtime find one.
- Ontology: Specification of a conceptualization. [Gruber 92]

State of the Art

- Mapping Ontology $\rightarrow$ programming language: Different Approaches with inherent problems.
  - Direct vs. indirect modelling

State of the Art = indirect Modelling (OWL-API)

```
// Source: https://dior.ics.muni.cz/~makub/owl/
OWLObjectProperty isEmployedAtProperty =
    factory.getOWLObjectProperty(":isEmployedAt", pm);

for (OWLLiteral email : reasoner.getDataPropertyValues(
        martin, hasEmailProperty)) {
    System.out.println("Martin has email: "
        + email.getLiteral());
}

for (OWLNamedIndividual ind :
        reasoner.getObjectPropertyValues(
        martin, isEmployedAtProperty).getFlattened()) {
    System.out.println("Martin is employed at: "
        + renderer.render(ind));
}
```

State of the Art

- Mapping Ontology $\rightarrow$ programming language: Different Approaches with inherent problems.
  - Direct vs. indirect modelling
- Mapping programming language $\rightarrow$ Ontology: No known approaches.

Basic Idea of our Work

- Linking classical C++-types with ontology-entities
- Not just a programmable editor

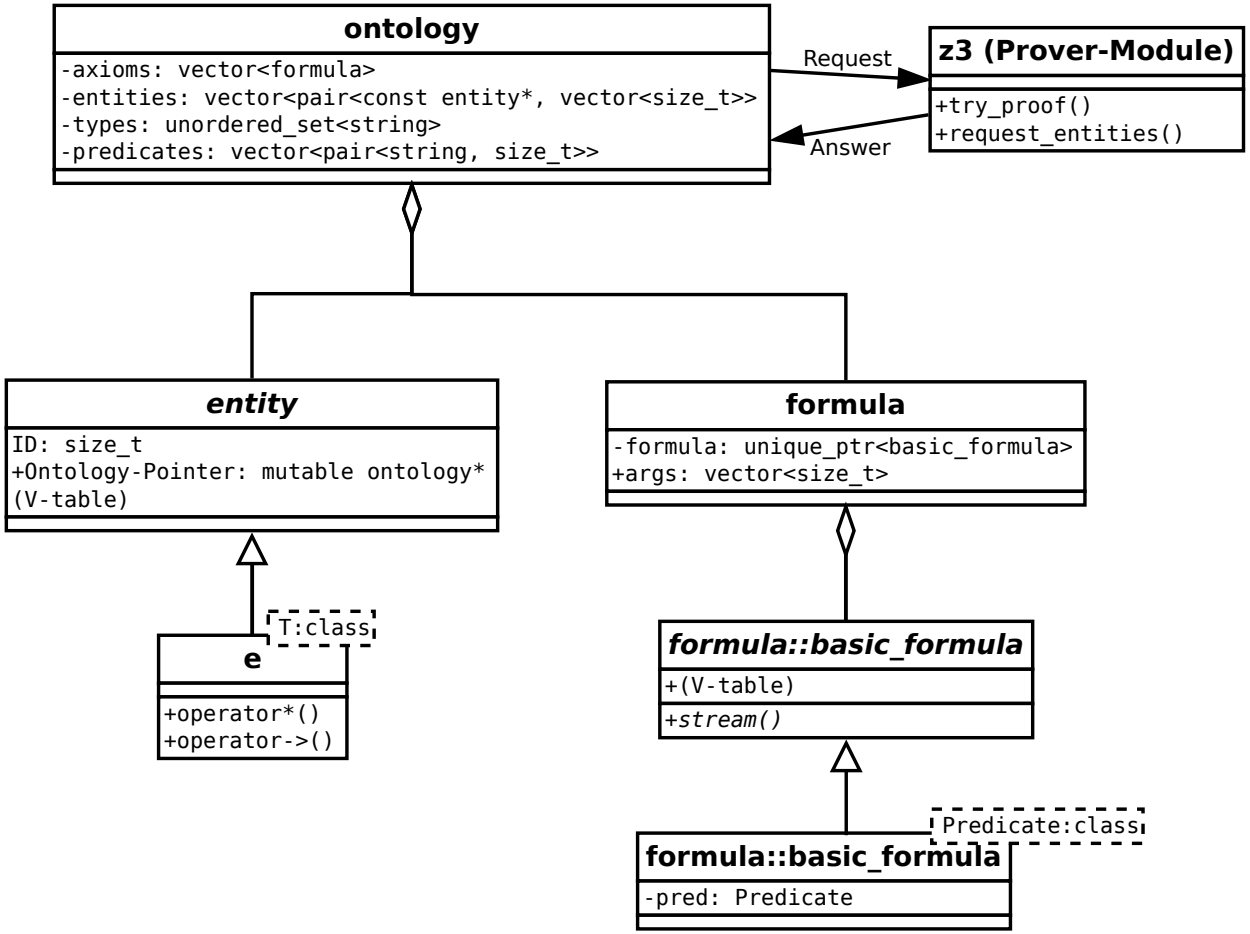
Definitions

- Class: General description of things of one kind in C++
- Object: Instance of a class
- Entity: Objects and classes about which semantic statements are being made.
- Predicate: A Relation that contains entities and returns a boolean value.
- Formula: Combination of entities and predicates.

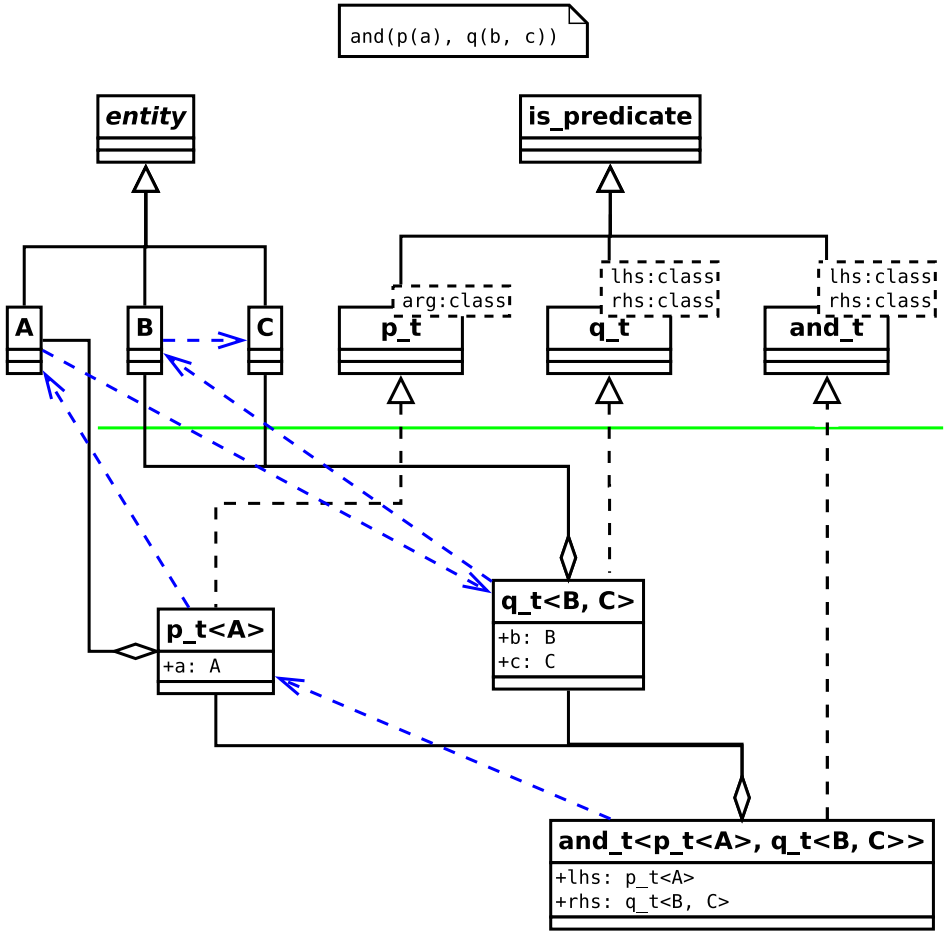
SOOP

- Static formulas with dynamic values.
- Semantic data retention.

Architecture

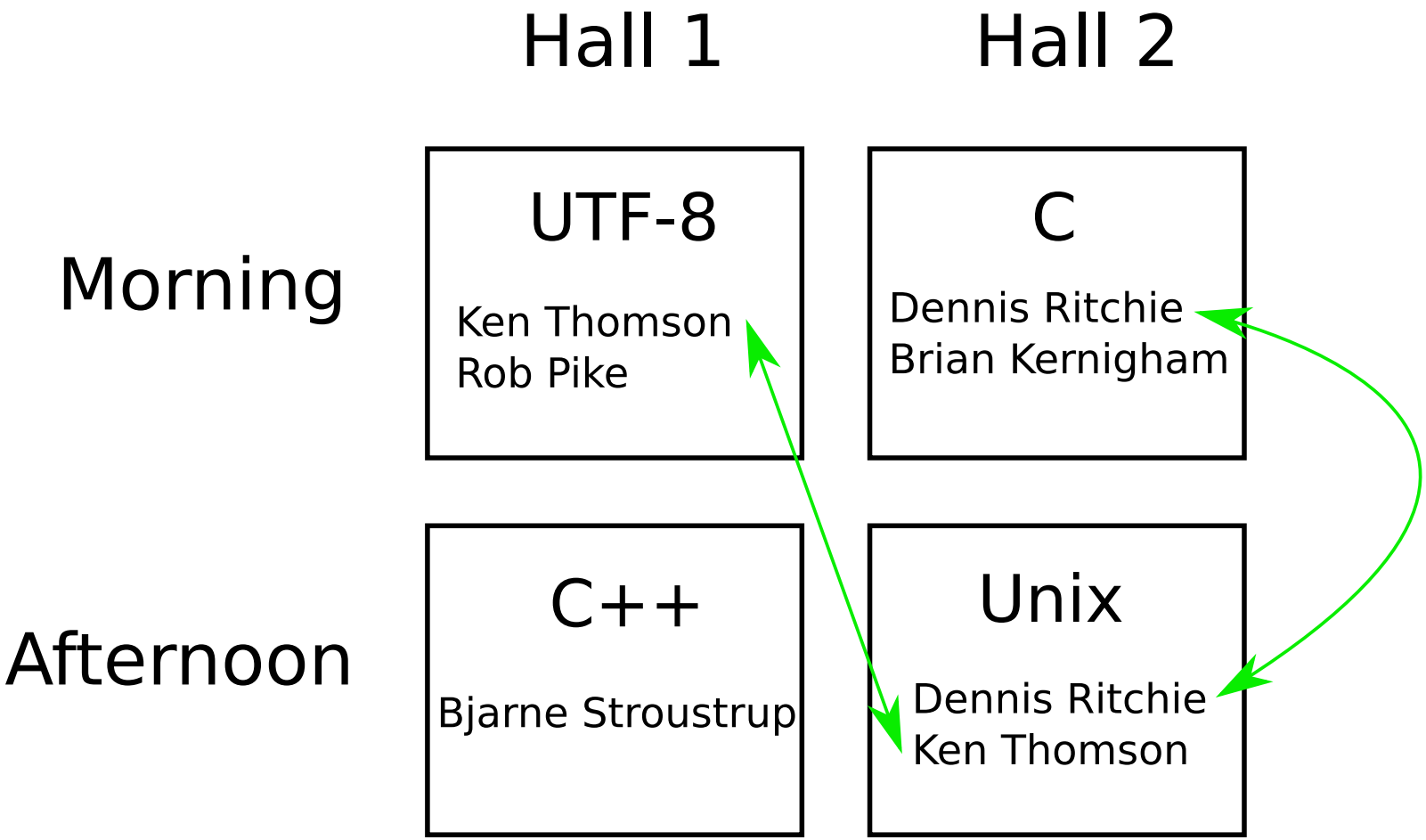


Predicate-instantiation



Application

- Conference scheduler: Talks, Speakers, Rooms, Slots
- Assignment is NP-hard ($\rightarrow$ 3COL)



Application - Relationships

```
#include <iostream>
#include <string>

#include <soop/onto.hpp>

class person {
public:
    person(std::string name): name{std::move(name)} {}
    std::string name;
};
using person_e = soop::e<person>;

SOOP_MAKE_TYPECHECKED_PREDICATE(parent_of, 2,
                                person_e, person_e)
```

Application - Relationships

```
int main() {
    using namespace preds;
    using namespace soop::preds;
    soop::ontology o{};
    o.add_type<person_e>();
    o.add_predicate<parent_of_t>();
    soop::variable<'p'> p;
    soop::variable<'c'> c;
    // parents and children are persons:
    o.add_axiom(forall({p, c}, implies(parent_of(p, c), and_(
                instance_of(p, soop::type<person_e>),
                instance_of(c, soop::type<person_e>)))));
    // parents are not their childrens' children:
    o.add_axiom(forall({p, c}, implies(parent_of(p, c),
                                       not_(parent_of(c, p)))));
```

Application - Relationships

```
    person_e max{o, "Max Mustermann"};
    person_e erika{o, "Erika Mustermann"};
    o.add_axiom(preds::parent_of(max, erika));

    soop::variable<'s'> s;
    const auto& parent = std::get<0>(
        o.request_entities<person_e>(
            exists({c}, parent_of(s, c)), s));
    std::cout << parent->name << " is a parent.\n";
}
```

Output

Max Mustermann is a parent.

Discussion

- More heuristics possible than in Prolog

Discussion

- More heuristics possible than in Prolog
- First investigation of mapping-direction

Discussion

- More heuristics possible than in Prolog
- First investigation of mapping-direction
- Demo-application demonstrates feasability in principal.

Discussion

- More heuristics possible than in Prolog
- First investigation of mapping-direction
- Demo-application demonstrates feasability in principal.
- Use with multiple different libraries is easy:

```
soop::e<lib1::tire> t;
soop::e<lib2::vehicle> v;
onto.add_axiom(preds::is_tire_of(t, v));
```

Outlook

- Keeping state of the prover
- Semantics of copies

Summary

- Ontologies can be generated from data in static languages.
- Doing so appears to be feasable in practice
- the code needed for doing this can blend in well with the programming-language

Thank you for your attention

Questions?

Backup: Big Axioms

```
forall({t1,t2,s1,s2,r1,r2,sl1,sl2},
        implies(
            and_(
                talk_assignment(t1,s1,r1,sl1),
                talk_assignment(t2,s2,r2,sl2)),
            equal(
                equal(t1,t2),
                and_(equal(r1,r2), equal(sl1,sl2)))));
```

Backup: Query for multiple Entities

```
auto solution = o.request_entities<room, slot>(
    talk_assignment(talk, speaker, r, sl), r, sl);
```