Programming Languages
and Formal Models
Mogens Nielsen, professor

Michael Schwartzbach, assoc. prof.

Olivier Danvy, assoc. prof.

Michael Achenbach, PhD stud.

Kim Birkelund, PhD stud.

Klaus Ostermann, assoc. prof.

Anders Møller, assoc. prof.

Erik Ernst, assoc. prof.

Doina Bucur, PhD stud.

Christian Hofer, PhD stud.

Janus D. Nielsen, PhD stud.

Tillmann Rendel, PhD stud.

Johnni Winther, PhD stud.

Simon Jensen, Stud.pgm.

Mathias Schwarz, Stud.pgm.

Karl Klose, PhD stud.

Anders Bach Nielsen, PhD stud.

UNIVERSITY OF AARHUS

Department of Computer Science

Programming Languages and Formal Models

+ M.Sc. students
Languages

- Programming languages is a core topic in computer science, but..
- There are so many ways to do it
- **Design** - abstraction mechanisms, typing concepts, run-time entities
- **Implementation** - performance
- **Formalization** - proofs of properties
- We work with most of these issues
Models and Abstraction

- “Real” software systems are complex
- So we build models, i.e., abstractions
- A good model
  - focuses on selected aspects and abstracts away the rest
  - supports formal reasoning
  - exhibits desirable properties
  - connects back to reality (soundness)
Areas of ongoing projects

- Trust in computing
- Secure computation
- Functional programming
- Language design and abstractions
- Web/XML programs
Consider some animals..

- A horse is an animal, and so is a lion
- “Animals” do not exist, but they are useful to express general scenarios
Now consider the housing..

- A horse stable is animal housing, and so is a lion’s cage.
- So obviously we can express general scenarios here, too?
Trouble if we do

- The abstract scenario turns out to be dangerous
- The problem is the assumption: Any kind of animal housing can be used with any kind of animal
There is a solution..

Use-site variance
Wildcards
Existential types
Soundness

(ECOOP 2008)

..and it works!

UNIVERSITY OF AARHUS
Department of Computer Science
Programming Languages and Formal Models
Java wildcards

- Basic idea in 1999 paper from Aarhus
- Developed in 2002 from Japan & Italy
- Implemented in joint project with Sun
- Part of Java J2SE 5
- .. but until now soundness had not been proven

```
AnimalHousing<? extends Animal> ah;
ah = new HorseStable(new Horse());
ah.add(new Lion()); // Nope!
Animal a = ah.getAnimal();
```
A Model of Java Wildcards

- Joint work with Nick Cameron and Sophia Drossopoulou, Imperial College, London
- Uses existential types
- Crucial point: No ‘open’ statements, subtyping crosses existential/non-ex. boundary

\[
\begin{align*}
\text{dom}(\Delta') \cap \text{fv}(\exists X \rightarrow [B_l \ B_u] \cdot N) &= \emptyset \\
\Delta, \Delta' \vdash [T/X] B_l <: T &\quad \Delta, \Delta' \vdash T <: [T/X] B_u \\
\Delta \vdash \exists X. [T/X] N \sqsubseteq \exists X \rightarrow [B_l \ B_u] \cdot N &\quad \text{(SC-ENV)}
\end{align*}
\]
Future work

- Investigate relation to virtual classes, ownership types, ...
- Include mutable state
- Mechanize proofs
- etc.