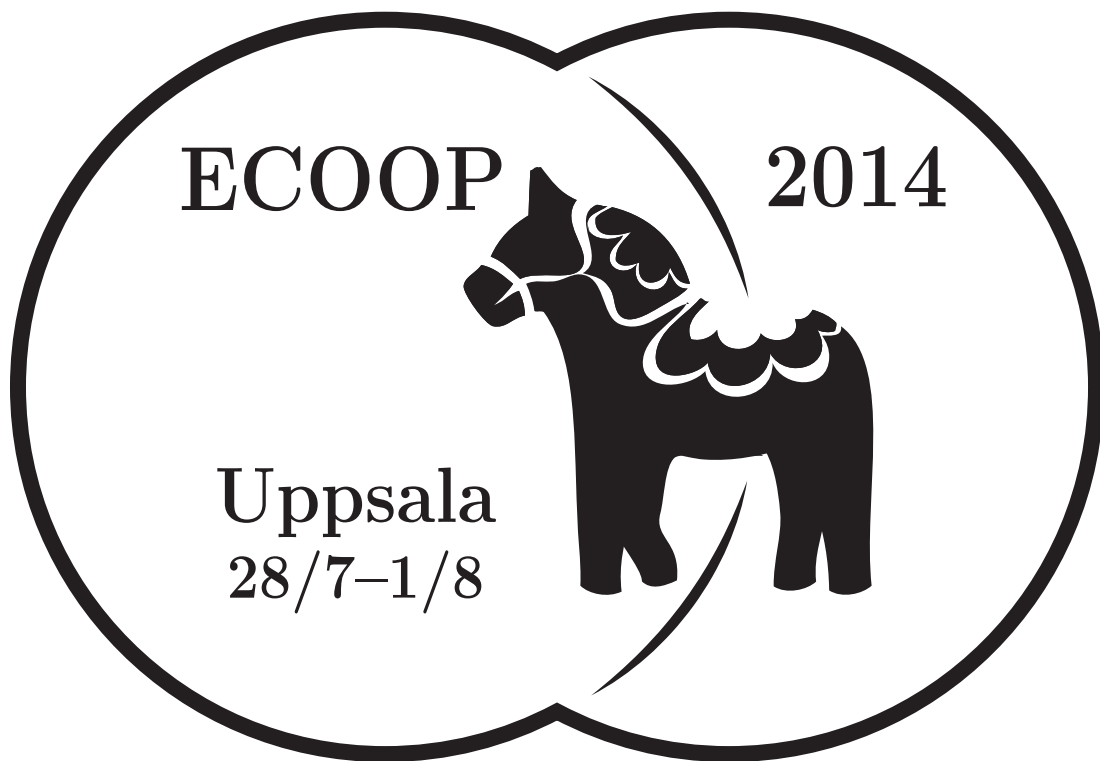


## ECOOP'14 Programme



Association Internationale  
pour les Technologies Objets

The official sponsor of ECOOP

## Locations

Also see map at the back.

### *Workshops 28/7*

What	Room	Explanation
FTfJP	Lecture Hall IV	Formal Techniques for Java-Like Programs
ICOOOLPS	Lecture Hall XI	Implementation, Compilation, Optimization of OO Languages, Programs and Systems
JSTools	Lecture Hall II	Tools for JavaScript Analysis
PhD Symp	Lecture Hall I	Doctoral Symposium (open only to participating students)
PLE	Lecture Hall VIII	Programming Language Evolution
Scala	Lecture Hall X	Fifth annual Scala Workshop
UPMARC	Lecture Hall IX	UPMARC Summer School

### *Workshops 29/7*

What	Room	Explanation
COP	Lecture Hall VIII	Context Oriented Programming
IWACO	Lecture Hall IV	International Workshop on Aliasing, Capabilities, and Ownership
PLAS	Lecture Hall XI	Programming Language and Analysis for Security
Scala	Lecture Hall X	Fifth annual Scala Workshop
UPMARC	Lecture Hall IX	UPMARC Summer School

### *ECOOP Conference 30/7–1/8*

What	Room	
Keynotes	Lecture Hall X	Main lecture hall
Technical Papers	Lecture Hall X	Main lecture hall
Summer School	Lecture Hall IX	
Demonstrations	Lecture Hall VIII	
Tutorials	Lecture Hall VIII	

*We thank the IT department at Uppsala University, and Dick Elfström, for printing this programme for ECOOP'14.*

# Contents

<b>1</b>	<b>General Information</b>	<b>1</b>
1.1	Welcome to Uppsala—The Ground Zero of OO . . . . .	1
1.2	Organisation . . . . .	1
1.2.1	Programme Chair . . . . .	1
1.2.2	Programme Committee . . . . .	1
1.2.3	External Review Committee . . . . .	2
1.2.4	Artifact Evaluation Committee Chairs . . . . .	3
1.2.5	Artifact Evaluation Committee . . . . .	3
1.2.6	Organising Committee . . . . .	3
1.2.7	Student Volunteers . . . . .	4
1.2.8	Official Sponsors of ECOOP 2014 . . . . .	4
1.3	Useful Information . . . . .	6
1.3.1	Conference Address . . . . .	6
1.3.2	Registration Desk and Dates . . . . .	6
1.3.3	Participation Identification (Badge Policy) . . . . .	6
1.3.4	Internet Access . . . . .	6
1.3.5	Access to On-line Proceedings . . . . .	6
1.3.6	Physical Copies of the Proceedings . . . . .	6
1.3.7	Lunches . . . . .	8
1.3.8	Coffee Breaks . . . . .	8
1.3.9	Award Ceremonies . . . . .	9
1.3.10	Paper Awards . . . . .	9
1.3.11	Electricity . . . . .	10
1.3.12	Restaurants . . . . .	10
1.3.13	Phone Numbers of Taxi Companies . . . . .	11
1.3.14	Getting to and from Uppsala . . . . .	11
1.3.15	Pharmacies . . . . .	12
1.3.16	<b>Emergency Numbers</b> . . . . .	12
1.3.17	Public Transport and Tickets . . . . .	13
1.3.18	Liability . . . . .	13
1.4	Social Programme . . . . .	13
1.4.1	Workshop Reception (28/7 17.30–20.00) . . . . .	15
1.4.2	Banquet (31/7 19.00–00.00) . . . . .	15
<b>2</b>	<b>Main Conference</b>	<b>17</b>
2.1	Keynotes . . . . .	17

2.2	Technical Papers . . . . .	20
2.3	Demos and Tutorials . . . . .	32
2.4	Posters . . . . .	34
<b>3</b>	<b>Satellite Events</b>	<b>37</b>
3.1	UPMARC Summer School . . . . .	37
3.1.1	Abstracts . . . . .	38
3.2	ECOOP Summer School . . . . .	39
3.2.1	Abstracts . . . . .	39
3.3	PhD Symposium . . . . .	44
3.3.1	Main Organiser . . . . .	44
3.3.2	Student Panel . . . . .	44
3.3.3	Academic Panel . . . . .	44
3.4	Workshops . . . . .	44
3.4.1	FTfJP: Formal Techniques for Java-Like Programs . . . . .	44
3.4.2	JSTools: Third Annual Workshop on Tools for JavaScript Analysis . . . . .	45
3.4.3	ICOOOLPS: 9th workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems) . . . . .	46
3.4.4	PLE: Workshop on Programming Language Evolution . . . . .	48
3.4.5	Scala: Fifth Workshop on Scala . . . . .	49
3.4.6	COP: 6th International Workshop on Context-Oriented Programming . . . . .	53
3.4.7	IWACO: International Workshop on Aliasing, Capabilities, and Ownership . . . . .	54
3.4.8	PLAS: ACM Ninth Workshop on Programming Languages and Analysis for Security	56
<b>4</b>	<b>Map of Downtown Uppsala</b>	<b>59</b>
<b>5</b>	<b>Schedule Overview &amp; Venue Map</b>	<b>61</b>

# 1 | General Information

This instalment of ECOOP is the 28th in ECOOP’s history and the first ECOOP in Sweden. The ECOOP conferences are organised locally and coordinated by AITO, a non-profit organisation dedicated to the advancement of object technology.

## 1.1 Welcome to Uppsala—The Ground Zero of OO

On behalf of the local organising team, I would like to welcome you to Uppsala and ECOOP 2014. As I write these words, Swedish summer is in full force: the temperature is somewhere between 25 and 30 degrees centigrade and there is certainly enough sun to go around (both day and night). Hopefully, the weather will be as nice during the conference, with a hint of rain in-between the breaks and lunches.

Uppsala—pronounced “oopsla”—is Sweden’s fourth largest city with a population of 200 000 and the home to Sweden’s oldest university, founded in 1477. Back then, there were twelve staff members, that somehow managed to be split into three different faculties. The University’s first documented PhD degree was awarded in year 1600. (Today, a Swedish PhD is 5 years.)

Carolus Linnaeus became a professor in medicine here in 1741. Whether you are a fan of Self, platypuses in general, Smalltalk or Java, there seems to be an agreement on the fundamental influence of Linnaeus’ work on taxonomies on the organisational principles of object-oriented programming. It’s on that note that I’d like to take the opportunity to welcome you all “home”.



Tobias Wrigstad/Organising Chair

## 1.2 Organisation

### 1.2.1 Programme Chair

Richard Jones, University of Kent

### 1.2.2 Programme Committee

Davide Ancona, DIBRIS, Università di Genova (Italy)

Sven Apel, University of Passau (Germany)

Walter Binder, University of Lugano (Switzerland)

Steve Blackburn, Australian National University (Australia)

Ana Cavalcanti, University of York (UK)  
Satish Chandra, Samsung Electronics (US)  
Dave Clarke, Katholieke Universiteit Leuven (Belgium)/Uppsala University (Sweden)  
Wolfgang De Meuter, Vrije Universiteit Brussel (Belgium)  
Isil Dillig, University of Texas at Austin (US)  
Amer Diwan, Google (USA)  
Lieven Eeckhout, Ghent University (Belgium)  
Robby Findler, Northwestern University (USA)  
Irene Finocchi, Sapienza University of Rome (Italy)  
Christian Hammer, Saarland University (Germany)  
Laurie Hendren, McGill University (Canada)  
Atsushi Igarashi, Kyoto University (Japan)  
Tomas Kalibera, Purdue University (USA)  
Doug Lea, SUNY Oswego (USA)  
Yu David Liu, SUNY Binghamton (USA)  
Cristina Lopes, University of California, Irvine (USA)  
Ana Milanova, Rensselaer Polytechnic Institute (USA)  
Nick Mitchell, IBM Research (USA)  
Eliot Moss, University of Massachusetts, Amherst (USA)  
Jens Palsberg, UCLA (USA)  
Matthew Parkinson, Microsoft Research (UK)  
Arnd Poetzsch-Heffter, University of Kaiserslautern (Germany)  
Dirk Riehle, Friedrich-Alexander-Universität Erlangen-Nürnberg (Germany)  
Yannis Smaragdakis, University of Athens (Greece)  
Arie van Deursen, Delft University of Technology (The Netherlands)  
Hongseok Yang, University of Oxford (UK)

### 1.2.3 External Review Committee

Vikram Adve, University of Illinois at Urbana-Champaign (USA)  
Jonathan Aldrich, Carnegie Mellon University (USA)  
Ioana Baldini, IBM Research (USA)  
Eric Bodden, Fraunhofer SIT and TU Darmstadt (Germany)  
Sebastian Burckhardt, Microsoft Research (USA)  
Shigeru Chiba, University of Tokyo (Japan)  
Ferruccio Damiani, Università di Torino (Italy)  
Werner Dietl, University of Waterloo (Canada)  
Sophia Drossopolou, Imperial College London (UK)  
Erik Ernst, Aarhus Universitet (Denmark)  
Matthew Flatt, University of Utah (USA)  
Michael Franz, University of California, Irvine (USA)  
Kathryn E Gray, University of Cambridge (UK)  
Sam Guyer, Tufts University (USA)  
Matthias Hauswirth, University of Lugano (Switzerland)  
Einar Broch Johnsen, University of Oslo (Norway)  
Christian Kästner, Carnegie Mellon University (USA)  
Jörg Kienzle, McGill University (Canada)

Ondrej Lhoták, University of Waterloo (Canada)  
 Hidehiko Masuhara, Tokyo Institute of Technology (Japan)  
 Romain Robbes, University of Chile (Chile)  
 Sukyoung Ryu, KAIST (South Korea)  
 Mooly Sagiv, Tel Aviv University (Israel)  
 Ina Schaefer, TU Braunschweig (Germany)  
 Friedrich Steimann, Fernuniversität in Hagen (Germany)  
 Alexander J. Summers, ETH Zurich (Switzerland)  
 Frank Tip, University of Waterloo (Canada)  
 Laurence Tratt, King's College London (UK)  
 Greta Yorsh, Queen Mary University of London (UK)

#### 1.2.4 Artifact Evaluation Committee Chairs

Camil Demetrescu, Sapienza University of Rome, Italy  
 Erik Ernst, Aarhus University, Denmark

#### 1.2.5 Artifact Evaluation Committee

Adriana E. Chis, University College Dublin (Ireland)  
 Alberto Bacchelli, Delft University of Technology (The Netherlands)  
 Carl Ritson, University of Kent (UK)  
 Dominic Orchard, University of Cambridge (UK)  
 Dominique Devriese, KU Leuven (Belgium)  
 Emilio Coppa, Sapienza University of Rome (Italy)  
 George Kastrinis, University of Athens (Greece)  
 Georgios Gousios, Delft University of Technology (The Netherlands)  
 Ilya Sergey, IMDEA Software Institute (Spain)  
 Karim Ali, University of Waterloo (Canada)  
 Mahdi Eslamimehr, Viewpoints Research Institute (USA)  
 Mike Rainey, INRIA-Rocquencourt (France)  
 Oscar E. A. Callaú, PLEIAD, University of Chile (Chile)  
 Valentin Wüstholtz, ETH Zurich (Switzerland)  
 Valerio Panzica La Manna, Politecnico di Milano (Italy)  
 Veselin Raychev, ETH Zurich (Switzerland)  
 Wei Huang, Rensselaer Polytechnic Institute (USA)

#### 1.2.6 Organising Committee

##### *Organising Chair*

Tobias Wrigstad, Uppsala Universitet (Sweden)

##### *Workshop Chair*

Nate Nystrom, University of Lugano (Switzerland)

##### *Poster and Demo Chair*

Wolfgang Ahrendt, Chalmers University of Technology (Sweden)

##### *Publicity Chair*

Werner Dietl, University of Waterloo (Canada)

*Student Volunteer Chair*

Jürgen Börstler, Blekinge Institute of Technology (Sweden)

*Summer School Chairs*

Jan Vitek, Purdue University (US)

James Noble, Victoria University of Wellington (New Zealand)

*Sponsor Co-Chairs*

Einar Broch Johnsen, University of Oslo (Norway)

Erik Ernst, Aarhus University (Denmark)

*Wine Chair*

Reiner Hähnle, Darmstadt University (Germany)

*Local Organising Co-Chair*

Johannes Borgström, Uppsala University (Sweden)

Kostis Sagonas, Uppsala University (Sweden)

Lars-Henrik Eriksson, Uppsala University (Sweden)

*Indispensable Organisational Memory and AITO Liaison*

Annabel Satin, Petit Canard Kitchen (UK)

*Local Student Aides*

Stephan Brandauer, Uppsala University (Sweden)

Elias Castegren, Uppsala University (Sweden)

Johan Östlund, Uppsala University (Sweden)

*Webmaster*

Stephan Brandauer, Uppsala University (Sweden)

**1.2.7 Student Volunteers**

Darya Kurilova, Carnegie Mellon (USA)

Gabor Kozar, Eotvos Lorand Univ (Hungary)

Eric Skoglund, Stockholm University (Sweden)

Krishna Narasimhan, Univ of Frankfurt (Germany)

Marianna Rapoport, University of Waterloo (Canada)

Paley Li, Victoria Univ of Wellington (New Zealand)

Timothy Jones, Victoria Univ of Wellington (New Zealand)

Jens Nicolay, Vrije Universiteit Brussels (Belgium)

**1.2.8 Official Sponsors of ECOOP 2014****Diamond Sponsor**



**Gold Sponsors**



**Silver Sponsors**



**Bronze Sponsors**



**Other Partners**



And: Underscore Consulting LLC.

## 1.3 Useful Information

This section gathers information about the conference, and about health and safety. A map of downtown Uppsala can be found on Page 59.

### 1.3.1 Conference Address

Universitetshuset (University Main Building), Biskopsgatan 3, Uppsala. See #1 on the map on Page 59.

The University Main Building is situated in the centre of the town, close to the cathedral. It was built in the 1880s and today it is used for lectures, conferences, concerts and academic festivities.

### 1.3.2 Registration Desk and Dates

The registration desk is inside the conference venue and is open every day from 8:00 until 16:00.

### 1.3.3 Participation Identification (Badge Policy)

The conference venue is a public building in Uppsala and a place on the tourist map. Thus, we must enforce a strict badge policy for ECOOP'14—badges should be worn at all times during the conference, and badges will be checked at the doors.

### 1.3.4 Internet Access

Personal login instructions for access to the conference venue WIFI can be found in the conference bags. If you have trouble accessing the network, **please make sure that you have chosen to login as a guest.**

Note that EDUROAM works in the conference venue, in the Uppsala and Stockholm train stations and at the Arlanda airport.

### 1.3.5 Access to On-line Proceedings

Springer provides complimentary access to the on-line proceedings of ECOOP via SpringerLink. Details on how to access the proceedings—including the association code/access token—have been sent to you in email, and are also available on a separate printed sheet in your conference bag.

There is a small song-and-dance routine attached to getting on-line access in the creation of a SpringerLink account at <http://link.springer.com/>.

### 1.3.6 Physical Copies of the Proceedings

A limited number are available for purchase at the registration desk at an affordable price.



Figure 1.1: All ECOOP activities take place inside the old Main University Building which dates from the late 1800's.

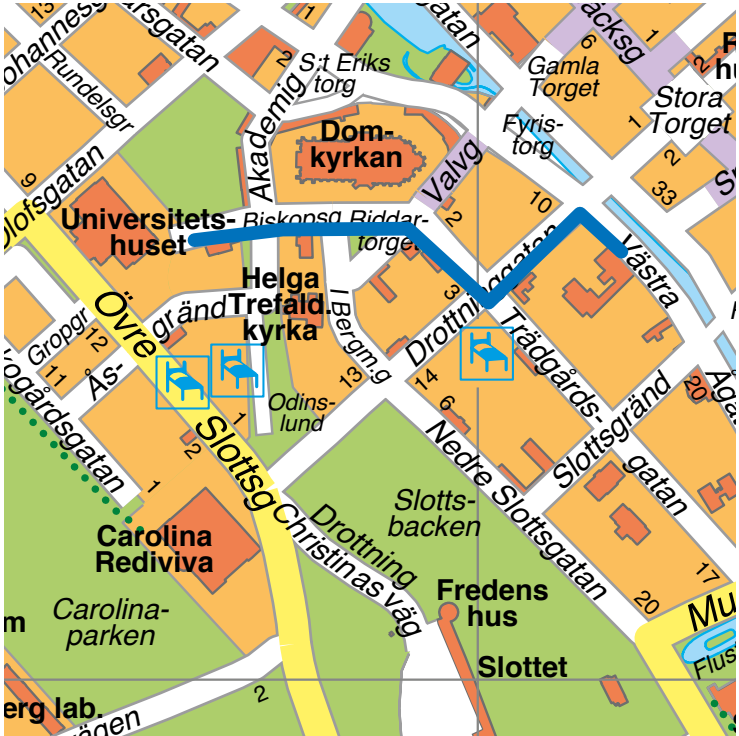


Figure 1.2: Walk from the conference venue (Universitetshuset) to the lunch venue (Norrland’s nation).

1.3.7 Lunches

Lunches are served 12:00–13:30 at “Norrland’s nation”. This is a short walk (5–10 minutes depending on your speed/mood). See Figure 1.2 for details.

Menu

Monday	Lamb patties with rich ratatouille, bred and marinade olives
Tuesday	Smoked salmon with sourish bulgur salad and cloudberry yogurt sauce
Wednesday	Salt lightly codfish with new potatoes, brown butter, shrimps and horseradish
Thursday	Sirloin steak, potato gratin, red wine sauce and fried onions
Friday	Caesar chicken sallad

“Norrland’s nation” is a student union for students from Northern parts of Sweden formed in 1827 through a merger of two other unions dating back to the 1600’s.

1.3.8 Coffee Breaks

On Monday and Tuesday (workshop days), coffee breaks are 10:00–10:30 and 15:00–15:30.

Wednesday–Friday (conference days), coffee breaks are 9:30–10:00 and 15:00–15:30.

Coffee is served *outside* of Lecture Hall X. Coffee is *not* allowed in lecture halls.





Figure 1.3: Lunches are served at Norrland's nation.

### 1.3.9 Award Ceremonies

Dahl–Nygaard awards are handed out in conjunction with keynotes on Thursday and Friday.

In 2004, AITO established an annual prize in the name of the Ole-Johan Dahl and Kristen Nygaard to honour their pioneering work on object-orientation.

This year AITO has awarded two Senior Prizes. The Senior Prize is awarded to *Robert France* for his research on adding formal semantics to object-oriented modelling notations, and to *William Cook* for his contributions to the theory and practice of object-oriented programming. The Junior Prize is awarded to *Tudor Gîrba* for his work on modelling and visualisation of evolution and interplay of large numbers of objects.

The Members of the 2014 Dahl–Nygaard Award Committee were: Awais Rashid (Chair), Walter Cazzola, Erik Ernst, Theo D'Hondt, and James Noble.

### 1.3.10 Paper Awards

The programme includes 27 excellent papers. Thanks to Springer's generous support, the ECOOP Programme Committee was able to recognise two papers as worthy of Distinguished Paper Awards. The winners will be announced at the conference banquet.

### 1.3.11 Electricity

The voltage in Sweden is 230v and the plugs/outlets are the standard round two-prong Europlug (type C and F) used in most Western European countries (not UK or Ireland).

One convenient place to purchase an adapter is Teknikmagasinet, (<http://www.teknikmagasinet.se>) Kungsängsgatan 1, Uppsala. Their opening hours are 10–19 on weekdays, 10–17 on Saturdays and 12–16 on Sundays.

### 1.3.12 Restaurants

There is a thriving food culture in Uppsala and you will find trendy new bistros alongside classic, familiar restaurants. Organic and locally produced, eccentric crossovers, finger food in its various forms, or old-fashioned Swedish home cooking; there is something for all tastes and moods. If you are interested in food you should try at least one of the eight restaurants that are listed in the 2014 White Guide.

Uppsala's many cafés are popular no matter what the time of day. Enjoy traditional Swedish coffee (quite strong) and refreshments in an old, traditional café or while watching the world go by at a sidewalk café.

If you like vibrant nightlife, there are also many nice bars and clubs.

#### Restaurants in the White Guide

Uppsala boasts several restaurants in this year's edition of Sweden's leading restaurant guide, the White Guide. Whether you are an enthusiast or if you just want to eat well, the White Guide is an invaluable guide and reference point if you want to try the best food offered. (Numbers match map on Page 59.)

8. Aaltos—Italian grill inside a piece of lovely architecture built by renowned Finnish architect Alvar Aalto. (Address: Sysslomansgatan 14.)
9. Dryck—little pub with just 17 seats focusing on drinks. Each week, a new drink menu is prepared accompanied by three complementing dishes. (Address: Olof Palmes Plats 4.)
10. Hambergs Fisk—Seafood restaurant with genuine interest in marine life. (Address: Fyristorg 8.)
11. Stationen—Continental dining inspired by Paris. (Address: Olof Palmes Plats 6.)
12. Villa Anna—Rustic Swedish cuisine mostly from local produce and high-quality organic produce. Quite pricey, but excellent. (Address: Odinslund 3.)

#### Swedish Food Not just fish.

13. Amandas Mat & Vinstudio. Run by one of Uppsala's best accomplished restaurant keepers. (Address: Fyrislundsgatan 81.)
14. Borgen. Good, classic Swedish food. (Address: Orphei Drängars plats 1.)
15. Domtrappkällaren. Normally great food. Huge outdoor eating area when the weather is good. (Address: S:t Eriks gränd 15.)
16. Lingon. Chefs serve you directly. (Address: Svartbäcksgatan 30.)
17. Smultron. Outdoor seating in beautiful gardens. (Address: Svartbäcksgatan 27.)

### 1.3.13 Phone Numbers of Taxi Companies

A large number of taxis usually linger at the Uppsala Central station, see #7 on the map on Page 59.

- Uppsala Taxi, +46 (0)18-100000, <http://www.uppsalataxi.se>
- Taxi Kurir, +46 (0)18-123456, <http://www.taxikurir.se>
- Taxi 020, +46 (0)20-202020, <http://www.taxi020.se>

### 1.3.14 Getting to and from Uppsala

Uppsala is located 40 minutes north of Stockholm. Uppsala can easily be reached from the four airports in the area. The organisers cannot take responsibility for late schedule changes or price changes. For updates, please check the airport websites. You can reach Uppsala from the four different airports as follows:

#### Arrival at Stockholm-Arlanda International Airport

*You want to go straight from Arlanda to Uppsala. Do not go via Stockholm as it is in the opposite direction.*

Stockholm-Arlanda International Airport is located between Uppsala and Stockholm, 36 km south of Uppsala.

**Trains** leave Arlanda for Uppsala Central Station directly from Sky City. Frequency: please consult the timetables online [www.resplus.se](http://www.resplus.se). The journey time is about 15-20 minutes and the cost is 90-185 SEK (approximately EUR 10-19) if ticket is purchased in advance. Ticket Desk: SJ Resebutik/Travel Office, in Sky City (next to Terminal 5). Purchasing a ticket on the train is expensive.

**Bus 801** runs between Arlanda and central Uppsala once every hour between 1 a.m and 4 a.m, and twice every hour between 4 a.m and 1 a.m. The journey takes about 45 minutes and costs SEK 100 (approximately EUR 11). You can buy the ticket with credit card on the bus (Amex and Diners are not accepted). *You cannot pay the ticket cash on the bus.* There are vending machines at Arlanda airport terminals 2, 4 and 5 where you can buy a one way ticket for the bus. Credit card only. There is a small reduction of the price if you buy it from the vending machine in advance of the journey, compared to buying the ticket on board the bus (SEK 90 appr EUR 10).

**Taxis** leave from outside the arrival hall. The journey time is about 30 minutes and the cost is about SEK 450 (approx. EUR 50). Ask the driver for a fixed price (“fast pris”) in advance. All taxis have that from Arlanda to Uppsala. You can pay by credit card or by cash. Advance reservations are possible but not necessary unless you arrive in really strange hours. See Section 1.3.13 for numbers. You can ask for the driver to wait for you in the terminal.

#### Arrival at Stockholm-Bromma Airport

Stockholm-Bromma airport is located 9 km west of Stockholm Central Station and 70 km south of Uppsala. The airport bus runs between Bromma airport and Stockholm city bus terminal, next to the central station. The timetable runs to connect with flight arrivals and departures. A single ticket costs SEK 79 (approximately EUR 8) and a return ticket SEK 150 (approximately EUR 15). The journey time to Stockholm is 20 minutes (depending on traffic). You buy your ticket from the ticket-vending machines or on-line on the [flygbussarna.se](http://flygbussarna.se) website. E-ticket purchased on-line is valid for 12 months from the purchase date.

You then go from Stockholm Central to Uppsala Central station. (See below.)

### Arrival at Stockholm-Skavsta Airport, city of Nyköping

Stockholm-Skavsta airport is located 110 km south of Stockholm Central Station and 177 km south of Uppsala. Airport shuttle busses run between Skavsta Airport and Stockholm Central Station. Airport busses (Flygbussarna): Frequency: about 30 minutes after every flight arrival. The bus waits for delayed flights. Timetable online: [www.flygbussarna.se](http://www.flygbussarna.se) The journey time to Stockholm is about 80 minutes and the cost is SEK 149 (approximately EUR 16) single ticket, SEK 259 (approximately EUR 28) round trip. You buy your ticket from the ticket-vending machines or on-line on the [www.flygbussarna.se](http://www.flygbussarna.se) website.

### Getting from Stockholm Central to Uppsala

*Beware: two train companies with incompatible tickets run this line.*

The SJ trains that the fast-running trains with one or two stops and a travel time of 40 minutes. A ticket costs about 10 EUR. Most SJ trains leave from platform 3 in Stockholm Central station and from platform 4 at Uppsala Central station. There is 1–2 trains per hour. You can get your tickets in vending machines close to the tracks at both stations.

The SL trains are commuter trains which stop frequently and are much less comfortable. The travel time is about 60 minutes and the cost is the same. 2–4 trains per hour run this line.

#### 1.3.15 Pharmacies

The closest pharmacy from the venue is “S:t Per”, located at S:t Persgatan 10. They are open 9:00–19:00 on weekdays, 10:00–17:00 on Saturdays, and 12:00–16:00 on Sundays. See #6 on the map on Page 59.

*Samariten* is open daily 8:00–22:00 and is located downtown at Kålsängsgränd 10.

#### 1.3.16 Emergency Numbers

112 is the emergency number that you can call from a landline or mobile phone anywhere in Sweden. The number 114 14 can be used to call the nearest police agency in case of a non-emergency.

- Call 112 if you need immediate help
- Call 112 if you or somebody in your vicinity falls ill or has a serious accident.
- Call 112 if there is a fire or any other situation requiring ambulance, fire or police services.

Your call will be answered by an operator at one of the SOS Alarm Centres who will ask you the following questions:

- What has happened?
- Where did it happen?
- What phone number are you calling from?

### National Telephone Number for Health Care Advice

1177 is a national telephone number for health care advice. You can call it 24 hours a day, wherever you are in the country. The nurses who answer calls to 1177 Healthcare Guide can answer your questions, assess your need for care, give you advice and refer you to the right care centre if needed.



**Emergency Ward**

CityAkuten, Dragarbrunnsgatan 70, B entrance. Call 1177 and ask to be directed there.

Open 07.00 to 23.00 every day, year round.

**Emergency Dental Care**

Public Dental Service in Uppsala County, Vretgränd 9A, +46 (0)18– 611 63 50

**1.3.17 Public Transport and Tickets**

Once you are in Uppsala, you have easy access around the city by public transport, in a taxi, on foot, or by bike. The compact city centre makes it easy to get around in Uppsala on foot. There are many nice parks, and a stroll by the river can be very pleasant in good weather. *All ECOOP activities and recommended hotels are within walking distance from each other. Fewer busses run in the summer, so unless weather is bad, or you want to rent a bike, walking is usually the best choice.*

The oddly named *Ski Total Bike Rental* rents bikes and helmets. They are located close to the Central Station at Kungsängsgatan 5A. (See map on Page 59.)

Some information about busses:

- You can pay by credit card on the busses. The fare is about 20 SEK and the ticket is valid for a return within 60 minutes.
- The two closest bus stops for the main conference venue are “Slottsbacken” (busses 4, 6, 7, 11, and 20), and “Universitetet” (busses 22 and 41). Lines 20 and 22 stop at the Central Station. The other lines stop at “Stadshuset” which is two blocks north-east of the Central Station.
- To go from the Central Station to the main Conference Venue (Universitetshuset), take bus 22 or bus 20 from stop A1. The ride is about 5 minutes.
- To go from the main Conference Venue (Universitetshuset) to the Central station, the best bet is to go to Slottsbacken and take any bus going in the downtown direction. Bus 20 stops at the Central Station, the other busses stop at Stadshuset, which is two blocks north-east of the Central Station.
- For more information and trip planning, see [www.ul.se](http://www.ul.se).

**1.3.18 Liability**

The organisers are not liable for damages and/or losses of any kind which may be incurred by the conference delegates or by any other individuals accompanying them, both during the official activities as well as going to/from the conference. Delegates are responsible for their own safety and belongings.

**1.4 Social Programme**

The ECOOP social programme contains a workshop reception and a banquet. Due to kind contributions from Spotify, we are happy to announce that **the banquet will be open to all students.**



Figure 1.4: The workshop reception is held in Uppsala's famous Botanical Gardens.



Figure 1.5: Walk from the conference venue (Universitetshuset) to the workshop reception venue (Botanical gardens). The walk is about 8–10 minutes.

### 1.4.1 Workshop Reception (28/7 17.30–20.00)

The Workshop Reception is held on Monday 28/7 in the Botanical Gardens of Uppsala, see #3 on the map on Page 59. The reception opens at 17.30 and includes a lighter meal, some complimentary wine, with the option of topping up at the bar which takes cash and credit cards.

Depending on how hungry you are, you might want to wander off to a restaurant in downtown Uppsala following the reception. In either case, here is the menu. Alternatives will be offered to people who listed food sensitivities or allergies.

<i>Menu</i>
Skagen with crayfish tails, dill and lavaret roe, served on sunflower bread
Flatbread roller with smoked reindeer, horseradish cream and lingonberries sweetened with honey
Brie with fig jelly and roasted pumpkin seeds, served on dried fruit bread
Vegetable platter with green beans, carrots, radishes and lettuce served with mango chutney dressing

### 1.4.2 Banquet (31/7 19.00–00.00)

The banquet will be held in the Spotify headquarters in downtown Stockholm. Busses will be picking us up outside the conference venue between, and drive back on a flexible schedule later in the evening. Snacks and drinks will be served already on the bus, to prevent you from starving or dehydration. **See separate sheet in conference bag for details.**

#### Banquet Speech: Category Theory As A Digestif: A Tribute to Joachim Lambek (1922–2014)

*Erik Meijer, Applied Duality Inc./TU Delft*

Banquet Speech, Spotify HQ, Stockholm

**Abstract** Category Theory is the Mathematicians’ interpretation of interface-based design, so whenever you hack together a new API in your favourite OO language, it is always a smart idea to ask (XXX→“What would XXX do?”) applied to the Category Theorist that worked on your same problem already decades ago.

Since lambda expressions are the new hot topic all across programming language land these days, we will invoke our question of conscience with “Joachim Lambek” and learn that Java 8 lambdas and method references are simply a Cartesian Closed Category, proving yet again that interfaces are the OO developers’ interpretation of Category Theory.

**Bio** “There are three Erik Meijers on wikipedia; a handsome ex-soccer player, a socialist politician, and a hacker. The latter is me.”



## 2 | Main Conference

**NOTE:** All technical papers and keynotes are held in Lecture Hall X.

For an overview of the main conference schedule, see the end of this programme.

### 2.1 Keynotes

#### **Keynote: Molecular Programming**

*Luca Cardelli, Microsoft Research Cambridge, UK and University of Oxford, UK*

Wednesday 8.30, Lecture Hall X

**Abstract** Nucleic acids (DNA/RNA) encode information digitally, and are currently the only truly ‘user-programmable’ entities at the molecular scale. They can be used to manufacture nano-scale structures, to produce physical forces, to act as sensors and actuators, and to do computation in between. Eventually we will be able to use them to produce nanomaterials at the bottom end of Moore’s Law, and to interface them with biological machinery to detect and cure diseases at the cellular level under program control. Recently, computational schemes have been developed that are autonomous (run on their own power) and involve only short, easily producible, DNA strands with no other complex molecules. While simple in mechanism, these schemes are highly combinatorial and concurrent.

Understanding and programming systems of this new kind requires new software technologies. Computer science has developed a large body of techniques for analyzing (modeling) and developing (engineering) complex programmable systems. Many of those techniques have a degree of mathematical generality that makes them suitable for applications to new domains. This is where we can make critical contributions: in developing and applying programming techniques (in a broad sense) that are unique to computing to other areas of science and engineering, and in particular at the interface between biology and nanotechnology.

**Bio** Luca Cardelli is a Principal Researcher at Microsoft Research Cambridge. He heads the Programming Principles and Tools Group and his main interests are type theory and operational semantics, mostly for applications to language design and distributed computing. Currently he is working in Computational Systems Biology.

Luca implemented the first compiler for ML and one of the earliest direct-manipulation user-interface editors. He was a member of the Modula-3 design committee and has designed a few experimental languages, of which the latest are Obliq, a distributed higher-order scripting language, and Polyphonic

C#, an object-oriented language with modern concurrency abstractions. His more protracted research activity has been in establishing the semantic and type-theoretic foundations of object-oriented languages.

Luca was born in Monecatini, Italy, and studied at the University of Pisa until 1978; he has a PhD in computer science from the University of Edinburgh (1982). He worked at Bell Labs, Murray Hill, from 1982 to 1985, and at DEC Systems Research Center in Palo Alto from 1985 to 1997 before joining Microsoft Research in Cambridge, UK. He is a Research Professor and Fellow of The Royal Society, an ACM Fellow, and an Elected Member of Academia Europaea and AITO. He was awarded the AITO Dahl-Nygaard Senior Prize in 2007.

### **Keynote: Software Environmentalism**

*Tudor Gîrba, CompuGroup Medical Schweiz AG*

Thursday, 8.30, Lecture Hall X

**Abstract** As software grows in size, complexity and age, software engineering becomes as much about dealing with existing systems as about building systems. Despite several decades of research into reverse engineering and program comprehension, the state of practice is still that engineers largely rely on manual code reading as the preferred means to understand systems. The main reason is that most existing approaches tend to be generic and ignore the context of systems. We cannot continue to let systems loose in the wild without any concern for how we will deal with them at a later time.

Two decades ago, Richard Gabriel coined the idea of software habitability. We go further and introduce the concept of software environmentalism as a systematic discipline to pursue and achieve habitability. Engineers have the right to build upon assessable systems and have the responsibility of producing assessable systems. For example, even if code has often a text shape, it is not text. The same applies to logs and anything else related to a software system. It's all data, and data is best dealt with through tools. No system should get away without dedicated tools that help us take it apart and recycle it effectively. Every significant object in a system should be allowed to have dedicated inspectors to reveal its various facets and interactions, and every significant library should come with dedicated debugging possibilities. We need to go back to the drawing board to (1) construct moldable development environments that help us drill into the context of systems effectively, (2) reinvent our underlying languages and technologies so that we can build assessable systems all the way down, and (3) re-educate our perception of what software engineering is.

**Bio** Tudor Gîrba attained his PhD from the University of Berne (2005), and now works as team and innovation lead at CompuGroup Medical Schweiz, and as an independent consultant. He leads the work on Moose, a smart open-source platform for software and data analysis and he is part of the board of Pharo, the new cool kid on the object-oriented languages arena.

He advocates that software assessment must be recognised as a critical software engineering activity. He developed the humane assessment method, and he is helping companies to rethink the way they manage complex software systems and data sets. To demystify innovation, he developed the demo-driven innovation method as a combination of design thinking, idea prototyping and storytelling. He is applying it in multiple contexts varying from research labs to engineering companies.

**Keynote: How do you like your software models?****Towards empathetic design of software modeling methods and tools**

*Robert France, Colorado State University, USA and INRIA, France*

Thursday, 16.15, Lecture Hall X

**Abstract** The terms Model Driven Development/Engineering (MDD/E) are typically used to describe software development approaches in which models of software systems play a pivotal role. In the past I have argued that good support for software modeling is essential to bringing software development closer to an engineering endeavour. As in other engineering disciplines, modeling should be an integral part of software processes that tackle the very challenging problems associated with the creation and evolution of complex software-based systems. While MDD/E research targets important software development problems, the results have not yet led to widespread effective use of software modeling practices. While the wicked problems associated with the development of complex systems is a factor, another is a lack of attention to the issue of fitness-for-purpose with respect to modeling methods and tools. The state-of-the-art leaves some practitioners with the impression that modeling techniques add significant accidental complexity to the software development process. In this talk, I argue that there is a need to take a more empathetic approach to the design of tools and methods. In *empathetic design*, methodologists and tool developers actively consider and evaluate how their tools and methods fit with how modeling practitioners across a wide skill spectrum (expert, average, novice modellers) work. This should lead to methods and tools that are fit-for-purpose, and open the door for more widespread use of software modeling techniques.

**Bio** Robert France was born in Jamaica and received his B.Sc. in Natural Sciences with First Class Honours from the University of the West Indies in Trinidad and Tobago. He attended Massey University in New Zealand under a Commonwealth Scholarship, where he graduated with a Ph.D. in Computer Science (1990). He is currently a Professor in the Department of Computer Science at Colorado State University. His research on model-driven software development focuses on providing software developers with mathematically-based software modeling languages and supporting analysis tools that they can use to specify and analyse critical software properties (e.g. behavioural and security properties). He is also actively engaged in research on domain-specific modeling languages, model-based approaches to software product line engineering, and models@runtime.

Robert is a founding editor-in-chief of the Springer journal on Software and Systems Modeling, and a founding steering committee member of the international conference series on Model Driven Engineering Languages and Systems (MODELS). In 2008 he and his co-authors (Andy Evans, Kevin Lano, Bernard Runpe) were awarded the MODELS 2008 Ten Year Most Influential Paper Award for the paper The UML as a Formal Modeling Notation. In 2013 he was awarded a 5-year International Chair at INRIA, the French National Institute for research in Computational Sciences. He was awarded a Colorado State University, College of Natural Sciences Professor Laureate in 2014.

**Keynote: A View on the Past, Present and Future of Objects**

*William Cook, University of Texas at Austin, USA*

Friday, 8.30, Lecture Hall X


**Abstract** Object-oriented programming has always been somewhat mysterious. It has been realised in a fairly pure form in several ways, in Smalltalk, Beta, COM, and SELF. There are several theories (three in

Pierce’s Types and Programming Languages, and more given by Abadi and Cardelli, Bruce and others). Many partial and failed theories have been published. Most programming languages today are hybrids of objects with other styles of programming. Yet many programming language researchers believe that objects are somehow evil. And still we are experimenting with different forms and inventing new ideas on top of objects. Objects have ‘won’ as far as I am concerned, or at least objects have won a place at the table.

So where do we go from here? While there are many low-level improvements that can be made, it is a reasonable time to consider the big picture. One of the original views of objects was as a form of modeling. Modeling has taken on a life of its own, but has not been as successful as objects were. In this talk I will sketch out a path forward for objects and modeling to work together.

**Bio** William Cook is an Associate Professor in the Department of Computer Sciences at the University of Texas at Austin. His research is focused on object-oriented programming, programming languages, modeling languages, and the interface between programming languages and databases. Prior to joining UT in 2003, Dr. Cook was Chief Technology Officer and co-founder of Allegis Corporation. He was chief architect for several award-winning products, including the eBusiness Suite at Allegis, the Writer’s Solution for Prentice Hall, and the AppleScript language at Apple Computer. At HP Labs his research focused on the foundations of object-oriented languages, including formal models of mixins, inheritance, and typed models of object-oriented languages. He completed his Ph.D. in Computer Science at Brown University in 1989.

## 2.2 Technical Papers

For an overview of the schedule, see the very end of the programme. The “artifact badge”  indicates that a paper is accompanied by an evaluated artifact.

---

### Session I—Analysis

Wednesday, 10.00–12.00, Lecture Hall X | Chair: Tomas Kalibera

#### State-sensitive Points-to Analysis for the Dynamic Behavior of JavaScript Objects

*Shiyi Wei, Virginia Tech, United States*

*Barbara G. Ryder, Virginia Tech, United States*

**Abstract** JavaScript object behavior is dynamic and adheres to prototype-based inheritance. The behavior of a JavaScript object can be changed by adding and removing properties at runtime. Points-to analysis calculates the set of values a reference property or variable may have during execution. We present a novel, partially flow-sensitive, context-sensitive points-to algorithm that accurately models dynamic changes in object behavior. The algorithm represents objects by their creation sites and local property names; it tracks property updates via a new control-flow graph representation. The calling context comprises the receiver object, its local properties and prototype chain. We compare the new points-to algorithm with an existing JavaScript points-to algorithm in terms of their respective performance and accuracy on a client application. The experimental results on real JavaScript websites show that the new points-to analysis significantly improves precision, uniquely resolving on average 11% more property lookup statements.



**Self-Inferencing Reflection Resolution for Java**

*Yue Li, School of Computer Science and Engineering, UNSW, Australia*

*Tian Tan, School of Computer Science and Engineering, UNSW, Australia*

*Yulei Sui, School of Computer Science and Engineering, UNSW, Australia*

*Jingling Xue, School of Computer Science and Engineering, UNSW, Australia*

**Abstract** Reflection has always been an obstacle both for sound and for effective under-approximate pointer analysis for Java applications. In pointer analysis tools, reflection is either ignored or handled partially, resulting in missed, important behaviors. This paper presents our findings on reflection usage in Java applications. Guided by these findings, we introduce a static reflection analysis, ELF, by exploiting a self-inferencing property inherent in reflective calls. Given a reflective call, the basic idea behind ELF is to automatically infer its targets based on the dynamic types of the arguments of its target calls and the downcasts on their returned values, if its targets cannot be already obtained from the metaobjects on which the reflective call is made. We evaluate ELF against DOOP’s state-of-the-art reflection analysis using all 11 DaCapo benchmarks and two applications. ELF can make a disciplined tradeoff among soundness, precision and scalability while also discovering usually more reflective targets.

**Constructing Call Graphs of Scala Programs**

*Karim Ali, University of Waterloo, Canada*

*Marianna Rapoport, University of Waterloo, Canada*

*Ondřej Lhoták, University of Waterloo, Canada*

*Julian Dolby, IBM T.J. Watson Research Center, United States*

*Frank Tip, University of Waterloo, Canada*

**Abstract** As Scala gains popularity, there is growing interest in programming tools for it. Such tools often require call graphs. However, call graph construction algorithms in the literature do not handle Scala features, such as traits and abstract type members. Applying existing call graph construction algorithms to the JVM bytecodes generated by the Scala compiler produces very imprecise results due to type information being lost during compilation. We adapt existing call graph construction algorithms, Name-Based Resolution (RA) and Rapid Type Analysis (RTA), for Scala, and present a formalization based on Featherweight Scala. We evaluate our algorithms on a collection of Scala programs. Our results show that careful handling of complex Scala constructs greatly helps precision and that our most precise analysis generates call graphs with 1.1-3.7 times fewer nodes and 1.5-18.7 times fewer edges than a bytecode-based RTA analysis.

**Finding Reference-Counting Errors in Python/C Programs with Affine Analysis**

*Siliang Li, Lehigh University, United States*

*Gang Tan, Lehigh University, United States*

**Abstract** Python is a popular programming language that uses reference counting to manage heap objects. Python also has a Foreign Function Interface (FFI) that allows Python extension modules to be written in native code such as C and C++. Native code, however, is outside Python’s system of memory management; therefore extension programmers are responsible for making sure these objects are reference counted correctly. This is an error prone process when code becomes complex. In this paper, we propose

Pungi, a system that statically checks whether Python objects' reference counts are adjusted correctly in Python/C interface code. Pungi transforms Python/C interface code into affine programs with respect to our proposed abstractions of reference counts. Our system performs static analysis on transformed affine programs and reports possible reference counting errors. Our prototype implementation found over 150 errors in a set of Python/C programs.

---

## Session II—Design

Wednesday, 13.30–15.00, Lecture Hall X | Chair: Erik Ernst

### Safely Composable Type-Specific Languages

*Cyrus Omar, Carnegie Mellon University, United States*

*Darya Kurilova, Carnegie Mellon University, United States*

*Ligia Nistor, Carnegie Mellon University, United States*

*Benjamin Chung, Carnegie Mellon University, United States*

*Alex Potanin, Victoria University of Wellington, New Zealand*

*Jonathan Aldrich, Carnegie Mellon University, United States*

**Abstract** Programming languages often include specialized syntax for common datatypes and some also build in support for specific specialized datatypes (e.g. regular expressions), but user-defined types must use general-purpose syntax. Frustration with this causes developers to turn to strings, leading to correctness, performance, security, and usability issues. Allowing library providers to extend a language with new syntax could help address these issues. Unfortunately, prior mechanisms either limit expressiveness or are not safely composable: individually unambiguous extensions can lead to ambiguities when used together. We introduce type-specific languages (TSLs): logic associated with a type that determines how the bodies of generic literals, able to contain arbitrary syntax, are parsed and elaborated, hygienically. A TSL is invoked only when a literal appears where a term of its type is expected, guaranteeing non-interference. We give evidence supporting the applicability of TSLs and formally specify them with a bidirectionally typed elaboration semantics for Wyvern.

### Graceful Dialects



*Michael Homer, Victoria University of Wellington, New Zealand*

*Timothy Jones, Victoria University of Wellington, New Zealand*

*James Noble, Victoria University of Wellington, New Zealand*

*Kim B. Bruce, Pomona College, United States*

*Andrew P. Black, Portland State University, United States*

**Abstract** Programming languages are enormously diverse, both in their essential concepts and in their accidental aspects. This creates a problem when teaching programming. To let students experience the diversity of essential concepts, the students must also be exposed to an overwhelming variety of accidental and irrelevant detail: the accidental differences between the languages are likely to obscure the teaching point. The dialect system of the Grace programming language allows instructors to tailor and vary the language to suit their courses, while staying within the same stylistic, syntactic and semantic framework, as well as permitting authors to define advanced internal domain-specific languages. The

dialect system achieves this power through a combination of well-known language features: lexical nesting, lambda expressions, multi-part method names, optional typing, and pluggable checkers. Grace’s approach to dialects is validated by a series of case studies, including both extensions and restrictions of the base language.

### Structuring Documentation to Support State Search: A Laboratory Experiment about Protocol Programming

*Joshua Sunshine, Carnegie Mellon University, United States*

*James D. Herbsleb, Carnegie Mellon University, United States*

*Jonathan Aldrich, Carnegie Mellon University, United States*

**Abstract** Application Programming Interfaces (APIs) often define object protocols. Objects with protocols have a finite number of states and in each state a different set of method calls is valid. Many researchers have developed protocol verification tools because protocols are notoriously difficult to follow correctly. However, recent research suggests that a major challenge for API protocol programmers is effectively searching the state space. Verification is an ineffective guide for this kind of search. In this paper we instead propose Plaiddoc, which is like Javadoc except it organizes methods by state instead of by class and it includes explicit state transitions, state-based type specifications, and rich state relationships. We compare Plaiddoc to a Javadoc control in a between-subjects laboratory experiment. We find that Plaiddoc participants complete state search tasks in significantly less time and with significantly fewer errors than Javadoc participants.

## Session III—Concurrency

Wednesday, 15.30–17.00, Lecture Hall X | Chair: Doug Lea

### Reusable Concurrent Data Types

*Vincent Gramoli, NICTA and University of Sydney, Australia*

*Rachid Guerraoui, EPFL, Switzerland*

**Abstract** This paper contributes to address the fundamental challenge of building Concurrent Data Types (CDT) that are reusable and scalable at the same time. We do so by proposing the abstraction of Polymorphic Transactions (PT): a new programming abstraction that offers different compatible transactions that can run concurrently in the same application. We outline the commonality of the problem in various object-oriented languages and implement PT and a reusable package in Java. With PT, annotating sequential ADTs guarantee novice programmers to obtain an atomic and deadlock-free CDT and let an advanced programmer leverage the application semantics to get higher performance. We compare our polymorphic synchronization against transaction-based, lock-based and lock-free synchronizations on SPARC and x86-64 architectures and we integrate our methodology to a travel reservation benchmark. Although our reusable CDTs are sometimes less efficient than non-composable handcrafted CDTs from the JDK, they outperform all reusable Java CDTs.

### TaDA: A Logic for Time and Data Abstraction

*Pedro da Rocha Pinto, Imperial College London, United Kingdom*

*Thomas Dinsdal-Young, Aarhus University, Denmark*

*Philippa Gardner, Imperial College London, United Kingdom*

**Abstract** To avoid data races, concurrent operations should either be at distinct times or on distinct data. Atomicity is the abstraction that an operation takes effect at a single, discrete instant in time, with linearisability being a well-known correctness condition which asserts that concurrent operations appear to behave atomically. Disjointness is the abstraction that operations act on distinct data resource, with concurrent separation logics enabling reasoning about threads that appear to operate independently on disjoint resources. We present TaDA, a program logic that combines the benefits of abstract atomicity and abstract disjointness. Our key contribution is the introduction of atomic triples, which offer an expressive approach to specifying program modules. By building up examples, we show that TaDA supports elegant modular reasoning in a way that was not previously possible.

### Infrastructure-Free Logging and Replay of Concurrent Execution on Multiple Cores

*Kyu Hyung Lee, Purdue University, United States*

*Dohyeong Kim, Purdue University, United States*

*Xiangyu Zhang, Purdue University, United States*

**Abstract** We develop a logging and replay technique for real concurrent execution on multiple cores. Our technique directly works on binaries and does not require any hardware or complex software infrastructure support. We focus on minimizing logging overhead as it only logs a subset of system calls and thread spawns. Replay is on a single core. During replay, our technique first tries to follow only the event order in the log. However, due to schedule differences, replay may fail. An exploration process is then triggered to search for a schedule that allows the replay to make progress. Exploration is performed within a window preceding the point of replay failure. During exploration, our technique first tries to reorder synchronized blocks. If that does not lead to progress, it further reorders shared variable accesses. The exploration is facilitated by a sophisticated caching mechanism. Our experiments on real world programs and real workload show that the proposed technique has very low logging overhead (2.6% on average) and fast schedule reconstruction.

---

## Session IV—Types

Thursday, 10.00–12.00, Lecture Hall X | Chair: Atsushi Igarashi

### Understanding TypeScript

*Gavin Bierman, Oracle, United Kingdom*

*Martín Abadi, Microsoft Research, United States*

*Mads Torgersen, Microsoft, United States*

**Abstract** TypeScript is an extension of JavaScript intended to enable easier development of large-scale JavaScript applications. While every JavaScript program is a TypeScript program, TypeScript offers a module system, classes, interfaces, and a rich gradual type system. The intention is that TypeScript provides a smooth transition for JavaScript programmers—well-established JavaScript programming idioms are supported without any major rewriting or annotations. One interesting consequence is that the TypeScript type system is not statically sound by design. The goal of this paper is to capture the essence of TypeScript by giving a precise definition of this type system on a core set of constructs of the language.

Our main contribution, beyond the familiar advantages of a robust, mathematical formalization, is a refactoring into a safe inner fragment and an additional layer of unsafe rules.

### Sound and Complete Subtyping between Coinductive Types for Object-Oriented Languages



*Davide Ancona, Università di Genova, Italy*

*Andrea Corradi, Università di Genova, Italy*

**Abstract** Structural subtyping is an important notion for effective static type analysis; it can be defined either axiomatically by a collection of subtyping rules, or by means of set inclusion between type interpretations, following the more intuitive approach of semantic subtyping, which allows simpler proofs of the expected properties of the subtyping relation. In object-oriented programming, recursive types are typically interpreted inductively; however, cyclic objects can be represented more precisely by coinductive types. We study semantic subtyping between coinductive types with records and unions, which are particularly interesting for object-oriented programming, and develop and implement a sound and complete top-down direct and effective algorithm for deciding it. To our knowledge, this is the first proposal for a sound and complete top-down direct algorithm for semantic subtyping between coinductive types.

### Spores: A Type-Based Foundation for Closures in the Age of Concurrency and Distribution

*Heather Miller, EPFL, Switzerland*

*Philipp Haller, Typesafe, Inc., Switzerland*

*Martin Odersky, EPFL, Switzerland*

**Abstract** Functional programming is regularly touted as the way forward for bringing parallel, concurrent, and distributed programming to the mainstream. The popularity of the rationale behind this viewpoint (immutable data transformed by function application) has even lead to a number of object-oriented programming languages adopting functional features such as lambdas and thereby function closures. However, despite this established viewpoint of FP as an enabler, reliably distributing closures over a network, or using them in concurrent environments nonetheless remains a challenge across FP and OO languages. This paper takes a step towards more principled distributed and concurrent programming by introducing a new closure-like abstraction and type system, called spores, that can guarantee closures to be serializable, thread-safe, or even have custom user-defined properties. Crucially, our system is based on the principle of encoding type information corresponding to captured variables in the type of a spore. We prove our type system sound, implement our approach for Scala, and show the power of these guarantees through a case analysis of real-world distributed/concurrent frameworks that this safe foundation for migratable closures facilitates.

### Rely-Guarantee Protocols

*Filipe Militão, Carnegie Mellon University & Universidade Nova Lisboa, Portugal*

*Jonathan Aldrich, Carnegie Mellon University, United States*

*Luís Caires, Universidade Nova Lisboa, Portugal*

**Abstract** The use of shared mutable state, commonly seen in object-oriented systems, is often problematic due to the potential conflicting interactions between aliases to the same state. We present a substructural type system outfitted with a novel lightweight interference control mechanism, rely-guarantee protocols, that enables controlled aliasing of shared resources. By assigning each alias separate roles, encoded

in a novel protocol abstraction in the spirit of rely-guarantee reasoning, our type system ensures that challenging uses of shared state will never interfere in an unsafe fashion. In particular, rely-guarantee protocols ensure that each alias will never observe an unexpected value, or type, when inspecting shared memory regardless of how the changes to that shared state (originating from potentially unknown program contexts) are interleaved at run-time.

---

## Session V—Implementation

Thursday, 13.30–15.00, Lecture Hall X | Chair: Wolfgang De Meuter

### Stream Processing with a Spreadsheet

*Mandana Vaziri, IBM T.J. Watson Research Center, United States*

*Olivier Tardieu, IBM T.J. Watson Research Center, United States*

*Rodric Rabbah, IBM T.J. Watson Research Center, United States*

*Philippe Suter, IBM T.J. Watson Research Center, United States*

*Martin Hirzel, IBM T.J. Watson Research Center, United States*

**Abstract** Continuous data streams are ubiquitous and represent such a high volume of data that they cannot be stored to disk, yet it is often crucial for them to be analyzed in real-time. Stream processing is a programming paradigm that processes these immediately, and enables continuous analytics. Our objective is to make it easier for analysts, with little programming experience, to develop continuous analytics applications directly. We propose enhancing a spreadsheet, a pervasive tool, to obtain a programming platform for stream processing. We present the design and implementation of an enhanced spreadsheet that enables visualizing live streams, live programming to compute new streams, and exporting computations to be run on a server where they can be shared with other users, and persisted beyond the life of the spreadsheet. We formalize our core language, and present case studies that cover a range of stream processing applications.

### Implicit Staging of EDSL Expressions: A Bridge between Shallow and Deep Embedding

*Maximilian Scherr, The University of Tokyo, Japan*

*Shigeru Chiba, The University of Tokyo, Japan*

**Abstract** Common implementation approaches for embedding DSLs in general-purpose host languages force developers to choose between a shallow (single-staged) embedding which offers seamless usage, but limits DSL developers, or a deep (multi-staged) embedding which offers freedom to optimize at will, but is less seamless to use and incurs additional runtime overhead. We propose a metaprogrammatic approach for extracting domain-specific programs from user programs for custom processing. This allows for similar optimization options as deep embedding, while still allowing for seamless embedded usage. We have implemented a simplified instance of this approach in a prototype framework for Java-embedded EDSL expressions, which relies on load-time reflection for improved deployability and usability.

### Babelsberg/JS - A Browser-based Implementation of an Object Constraint Language



*Tim Felgentreff, Hasso Plattner Institute, Germany*

*Alan Borning, University of Washington, United States*

*Robert Hirschfeld, Hasso Plattner Institute, Germany*

*Jens Lincke, Hasso Plattner Institute, Germany*

*Yoshiki Ohshima, Viewpoints Research Institute, Japan*

*Bert Freudenberg, Viewpoints Research Institute, Germany*

*Robert Krahn, Communications Design Group, SAP Labs, United States*

**Abstract** Constraints provide a useful technique for ensuring that desired properties hold in an application. They have been used in a wide range of applications, including graphical layout, simulation, and scheduling. We describe the design and implementation of an Object Constraint Programming language, a language that cleanly integrates constraints with an object-oriented language in a way that respects encapsulation and OO programming techniques, and that runs in browser-based applications. Prior work on OCP has relied on modifying the underlying VM, but that is not an option for web-based applications, which are increasingly prominent. In this paper, we demonstrate a language, Babelsberg/JS, a number of its applications, and provide performance measurements. Programs without constraints run at the same speed as pure JavaScript, while programs with constraints still run efficiently. Our design and implementation incorporate incremental re-solving as well as a cooperating solvers architecture that allows multiple solvers to work together.

---

### Special Session—Evaluated Artifacts

Thursday 31/7 15:30–16:15, Lecture Hall X | Chairs: Camil Demetrescu and Erik Ernst

In this session the evaluated artifacts are presented, and Distinguished Artifacts awarded. The Artifact Evaluation (AE) process at ECOOP 2014 is a continuation of the AE process at ECOOP 2013, OOPSLA 2013, and ESEC/FSE 2011. It enables artifacts (tools, data, models, videos, etc.) to be an integrated part of the publication, providing crucial and detailed evidence for the quality of the results that the associated paper offers, supporting repeatability of experiments, aiding creation of derived work, and enabling deeper comparisons with alternative approaches. The AE process was run by the Artifact Evaluation Committee (AEC) whose task was to assess how the artifacts support the work described in the papers. The AEC has reviewed the submitted artifacts and accepted 11 of them, designated as Evaluated Artifacts and marked as such with a badge in the proceedings and elsewhere. This session features a brief presentation of each of the Evaluated Artifacts; a Distinguished Artifact Award is given to the creators of especially meritorious artifacts at the end of the session.

---

### Session VI—Refactoring

Friday, 10.00–12.00, Lecture Hall X | Chair: Irene Finocchi

#### Automated Multi-Language Artifact Binding and Rename Refactoring between Java and DSLs used by Java Frameworks



*Philip Mayer, Programming & Software Engineering Group, LMU Munich, Germany*

*Andreas Schroeder, Programming & Software Engineering Group, LMU Munich, Germany*

**Abstract** Developing non-trivial software applications involves using multiple programming languages. Although each language is used to describe a particular aspect of the system, artifacts defined inside those languages reference each other across language boundaries; such references are often only resolved

at runtime. However, it is important for developers to be aware of these references during development time for programming understanding, bug prevention, and refactoring. In this work, we report on a) an approach and tool for automatically identifying multi-language relevant artifacts, finding references between artifacts in different languages, and (rename-) refactoring them, and b) on an experimental evaluation of the approach on seven open-source case studies which use a total of six languages found in three frameworks. As our main result, we provide insights into the incidence of multi-language bindings in the case studies as well as the feasibility of automated multi-language rename refactorings.

### Retargetting Legacy Browser Extensions to Modern Extension Frameworks

*Rezwana Karim, Rutgers University, United States*

*Mohan Dhawan, IBM Research, Delhi, India, India*

*Vinod Ganapathy, Rutgers University, United States*

**Abstract** Most modern Web browsers expose a rich API that allows third-party extensions to access privileged browser objects. However, this API can also be misused by attacks directed against vulnerable extensions. Web browser vendors have therefore recently developed new frameworks aimed at better isolating extensions while still allowing access to privileged browser state. Examples of such frameworks include the Google Chrome extension architecture and the Mozilla Jetpack extension framework. We present Morpheus, a tool to port legacy browser extensions to these new frameworks. Specifically, Morpheus targets legacy extensions for the Mozilla Firefox browser, and ports them to the Jetpack framework. We describe the key techniques used by Morpheus to analyze and transform legacy extensions so that they conform to the constraints imposed by Jetpack and simplify runtime policy enforcement. Finally, we present an experimental evaluation of Morpheus by applying it to port 52 legacy Firefox extensions to the Jetpack framework.

### Capture-Avoiding and Hygienic Program Transformations



*Sebastian Erdweg, TU Darmstadt, Germany*

*Tijs van der Storm, CWI, Amsterdam, Netherlands*

*Yi Dai, University of Marburg, Germany*

**Abstract** Program transformations in terms of abstract syntax trees compromise referential integrity by introducing variable capture. Variable capture occurs when in the generated program a variable declaration accidentally shadows the intended target of a variable reference. Existing transformation systems either do not guarantee the avoidance of variable capture or impair the implementation of transformations. We present an algorithm called name-fix that automatically eliminates variable capture from a generated program by systematically renaming variables. name-fix is guided by a graph representation of the binding structure of a program, and requires name-resolution algorithms for the source language and the target language of a transformation. name-fix is generic and works for arbitrary transformations in any transformation system that supports origin tracking for names. We verify the correctness of name-fix and identify an interesting class of transformations for which name-fix provides hygiene. We demonstrate the applicability of name-fix for implementing capture-avoiding substitution, inlining, lambda lifting, and compilers for two domain-specific languages.



## Converting Parallel Code from Low-Level Abstractions to Higher-Level Abstractions

*Semih Okur, University of Illinois at Urbana-Champaign, United States*

*Cansu Erdogan, University of Illinois at Urbana-Champaign, United States*

*Danny Dig, Oregon State University, United States*

**Abstract** Parallel libraries continuously evolve from low-level to higher-level abstractions. However, developers are not up-to-date with these higher-level abstractions, thus their parallel code might be hard to read, slow, and unscalable. Using a corpus of 880 open-source C# applications, we found that developers still use the old Thread and ThreadPool abstractions in 62% of the cases when they use parallel abstractions. Converting code to higher-level abstractions is (i) tedious and (ii) error-prone. We present two automated migration tools, Taskifier and Simplifier that work for C# code. The first tool transforms old style Thread and ThreadPool abstractions to Task abstractions. The second tool transforms code with Task abstractions into higher-level design patterns. Using our code corpus, we have applied these tools 3026 and 405 times, respectively. Our empirical evaluation shows that the tools (i) are highly applicable, (ii) reduce the code bloat, (iii) are much safer than manual transformations.

## Session VII—Javascript, PHP and Frameworks

Friday, 13.30–15.00, Lecture Hall X | Chair: Frank Tip

### Portable and Efficient Run-time Monitoring of JavaScript Applications using Virtual Machine Layering



*Erick Lavoie, McGill University, Canada*

*Bruno Dufour, Université de Montréal, Canada*

*Marc Feeley, Université de Montréal, Canada*

**Abstract** Run-time monitoring of JavaScript applications is typically achieved either by instrumenting a browser's virtual machine, usually degrading performance to the level of a simple interpreter, or through complex ad hoc source-to-source transformations. This paper reports on an experiment in layering a portable JS VM on the host VM to expose implementation-level operations that can then be redefined at run-time to monitor an application execution. Our prototype, Photon, exposes object operations and function calls through a meta-object protocol. In order to limit the performance overhead, a dynamic translation of the client program selectively modifies source elements and run-time feedback optimizes monitoring operations. Photon introduces a 4.7 to 191 slowdown when executing benchmarks on popular web browsers. Compared to the Firefox interpreter, it is between 5.5 slower and 7 faster, showing the layering approach is competitive with the instrumentation of a browser VM while being faster and simpler than other source-to-source transformations.

### An Executable Formal Semantics of PHP



*Daniele Filaretti, Imperial College London, United Kingdom*

*Sergio Maffei, Imperial College London, United Kingdom*

**Abstract** PHP is among the most used languages for server-side scripting. Although substantial effort has been spent on the problem of automatically analysing PHP code, vulnerabilities remain pervasive in

web applications, and analysis tools do not provide any formal guarantees of soundness or coverage. This is partly due to the lack of a precise specification of the language, which is highly dynamic and often exhibits subtle behaviour. We present the first formal semantics for a substantial core of PHP, based on the official documentation and experiments with the Zend reference implementation. Our semantics is executable, and is validated by testing it against the Zend test suite. We define the semantics of PHP in a term-rewriting framework which supports LTL model checking and symbolic execution. As a demonstration, we extend LTL with predicates for the verification of PHP programs, and analyse two common PHP functions.

### Identifying Mandatory Code for Framework Use via a Single Application Trace

*Naoya Nitta, Graduate School of Natural Science Konan University, Japan*

*Izuru Kume, Graduate School of Information Science, Nara Institute of Science and Technology, Japan*

*Yasuhiro Takemura, Department of Character Creative Arts, Osaka University of Arts, Japan*

**Abstract** Application frameworks allow application developers to effectively reuse both designs and implementations which frequently appear in their intended domains. However, when using a framework with large scale APIs, its usage to implement an application-specific behavior tends to be complicated. Thus, in practice, application developers use existing sample application code as references for their development, but the task to locate the parts which are related to their application usually becomes a burden. To address this problem, in this paper, we characterize the problem as a kind of dynamic flow analysis problem, and based on the characterization, we present a method to automatically identify the mandatory code for the framework use using only a single sample application's trace. We have conducted case studies with several real-world frameworks to validate our method and the results indicate that the method is suitable to extract the mandatory framework usage.

---

## Session VIII—Parallelism

Friday, 15.30–17.00, Lecture Hall X | Chair: Dave Clarke

### Cooperative Scheduling of Parallel Tasks with General Synchronization Patterns

*Shams Imam, Rice University, United States*

*Vivek Sarkar, Rice University, United States*

**Abstract** In this paper, we address the problem of scheduling parallel tasks with general synchronization patterns using a cooperative runtime. Current implementations for task-parallel programming models provide efficient support for fork-join parallelism, but are unable to efficiently support more general synchronization patterns such as locks, futures, barriers and phasers. We propose a novel approach to addressing this challenge based on cooperative scheduling with one-shot delimited continuations (OSDeConts) and event-driven controls (EDCs). The use of OSDeConts enables the runtime to suspend a task at any point (thereby enabling the task's worker to switch to another task) whereas other runtimes may have forced the task's worker to be blocked. The use of EDCs ensures that identification of suspended tasks that are ready to be resumed can be performed efficiently. Furthermore, our approach is more efficient than schedulers that spawn additional worker threads to compensate for blocked worker threads. We have implemented our cooperative runtime in Habanero-Java (HJ), an explicitly parallel language with a large variety of synchronization patterns. The OSDeCont and EDC primitives are used to implement a wide range of synchronization constructs, including those where a task may trigger the enablement of

multiple suspended tasks (as in futures, barriers and phasers). In contrast, current task-parallel runtimes and schedulers for the fork-join model (including schedulers for the Cilk language) focus on the case where only one continuation is enabled by an event (typically, the termination of the last child/descendant task in a join scope). Our experimental results show that the HJ cooperative runtime delivers significant improvements in performance and memory utilization on various benchmarks using future and phaser constructs, relative to a thread-blocking runtime system while using the same underlying work-stealing task scheduler.

### MiCA: A Compositional Architecture for Gossip Protocols



*Lonnie Princehouse, Cornell University, United States*

*Rakesh Chenchu, Cornell University, United States*

*Zhefu Jiang, Cornell University, United States*

*Kenneth Birman, Cornell University, United States*

*Nate Foster, Cornell University, United States*

*Robert Soulé, University of Lugano, Switzerland*

**Abstract** The developers of today’s cloud computing systems are expected to not only create applications that scale well, but also to create management services that will monitor run-time conditions and intervene to address problems as conditions evolve. Management tasks are generally not performance intensive, but robustness is critical: when a large system becomes unstable, the management infrastructure must remain reliable, predictable, and fault-tolerant. A wide range of management tasks can be expressed as gossip protocols, where nodes in the system periodically interact with random peers and exchange information about their respective states. MiCA is a new system for building gossip-based management tools that are highly resistant to disruptions and make efficient use of system resources. MiCA provides abstractions custom-tailored for gossip, a collection of composition operators that facilitates constructing sophisticated protocols in a modular style, and automatic optimizations that can often greatly reduce the number and size of messages used.

### Semantics of (Resilient) X10



*Silvia Crafa, University of Padova, Italy*

*David Cunningham, Google, Inc, United States*

*Vijay Saraswat, IBM TJ Watson Research Center, United States*

*Avraham Shinnar, IBM TJ Watson Research Center, United States*

*Olivier Tardieu, IBM TJ Watson Research Center, United States*

**Abstract** We present a formal small-step structural operational semantics for a large fragment of X10, unifying past work. The fragment covers multiple places, mutable objects on the heap, sequencing, try/catch, async, finish, and at constructs. This model accurately captures the behavior of a large class of concurrent, multi-place X10 programs. Further, we introduce a formal model of resilience in X10. During execution of an X10 program, a place may fail for many reasons. Resilient X10 permits the program to continue executing, losing the data at the failed place, and most of the control state, and repairing the global control state in such a way that key semantic principles hold, the Happens Before Invariance Principle, and the Exception Masking Principle. These principles permit an X10 programmer to write clean code that continues to work in the presence of place failure. The given semantics have additionally been mechanized in Coq.

## 2.3 Demos and Tutorials

This year ECOOP, hosts four demonstrations and one tutorial directly after the main track on Wednesday and Thursday, in parallel with the last session on the ECOOP summer school. All demonstrations and tutorials take place in Lecture Hall VIII.

### Schedule Wednesday 30/7

- |             |  |
|-------------|--|
| 17.15–17.45 | The GTInspector: A moldable object inspector<br><i>Andrei Chis, Tudor Gîrba and Oscar Nierstrasz</i> |
| 17.45–18.15 | Yin-Yang: Concealing the Deep Embedding of DSLs<br><i>Vojin Jovanovic and Amir Shaikhha</i>          |
| 18.15–18.45 | An MDE Tool-Chain for Pattern-Based S&D Embedded System Engineering<br><i>Brahim Hamid</i>           |

### Schedule Thursday 31/7

- |             |   |
|-------------|---|
| 17.20–17.50 | Automated and Sound Variable Renaming to Avoid Variable Capture in Generated Code<br><i>Sebastian Erdweg, Tijs van der Storm and Yi Dai</i> |
| 17.50–18.45 | Tutorial: Rust – bare metal programming without the sharp edges<br><i>Nicholas Cameron, Mozilla</i>   |

#### Demo: The GTInspector: A moldable object inspector

*Andrei Chis, University of Bern (Switzerland)*  
*Tudor Gîrba, CompuGroup Medical Schweiz AG*  
*Oscar Nierstrasz, University of Bern (Switzerland)*

**Abstract** Answering run-time questions in object-oriented systems involves reasoning about and exploring connections between multiple objects. Traditional object inspectors favor a generic view that focuses on the low level details of the state of single objects. However, different developer questions exercise different aspects of an object and require different kinds of interactions depending on the development context. The GTInspector is a novel moldable object inspector that allows developers to look at objects using multiple interchangeable presentations. These presentations are arranged in a workflow in which multiple levels of connecting objects can be seen together. In this tool demo we show how the GTInspector improves the inspection process by applying it to several use-cases, such as interacting with the file system or inspecting results from a database query. We also show how the inspector can be extended with new presentations. More information about the GTInspector can be found at: [scg.unibe.ch/research/moldableinspector](http://scg.unibe.ch/research/moldableinspector).

**Demo: Yin-Yang: Concealing the Deep Embedding of DSLs***Vojin Jovanovic, EPFL (Switzerland)**Amir Shaikhha, EPFL (Switzerland)*

**Abstract** Deeply embedded domain-specific languages (EDSLs) inherently compromise programmer experience for improved program performance. Shallow EDSLs complement them by trading program performance for good programmer experience. We present Yin-Yang, a framework for DSL embedding that uses Scala reflection to reliably translate shallow EDSL programs to the corresponding deep EDSL programs. The translation allows program prototyping and debugging in the user friendly shallow embedding, while the corresponding deep embedding is used in production—where performance is important. The reliability of the translation completely conceals the deep embedding from the DSL user. For the DSL author, Yin-Yang generates the deep DSL embeddings from their shallow counterparts by reusing the core translation. This obviates the need for code duplication and assures that the implementations of the two embeddings are always synchronised.

**Demo: An MDE Tool-Chain for Pattern-Based S&D Embedded System Engineering***Brahim Hamid, Université de Toulouse (France)*

**Abstract** In our work, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). This video tutorial presents the SEMCO MDE Tool Suite called TERESA to support PBSE in the domain of assistance to the secure and dependable embedded system engineering. We provide guidelines on how to use it to build and to store reusable artefacts (S&D patterns and property models) into a model-based repository. Once the repository is available, it serves an underlying trust engineering process. We propose a Connected Data Objects (CDO) based implementation of the repository of patterns and a set of EMF tree-based editors to create patterns and the required libraries.

**Demo: Automated and Sound Variable Renaming to Avoid Variable Capture in Generated Code***Sebastian Erdweg, TU Darmstadt, Germany**Tijs van der Storm, CWI (The Netherlands) / INRIA Lille (France)**Yi Dai, University of Marburg (Germany)*

**Abstract** Program transformations in terms of abstract syntax trees compromise referential integrity by introducing variable capture. Variable capture occurs when in the generated program a variable declaration accidentally shadows the intended target of a variable reference. Existing transformation systems either do not guarantee the avoidance of variable capture or impair the implementation of transformations. We present an algorithm called name-fix that automatically eliminates variable capture from a generated program by systematically renaming variables. name-fix is guided by a graph representation of the binding structure of a program, and requires name-resolution algorithms for the source language and the target language of a transformation. name-fix is generic and works for arbitrary transformations in any transformation system that supports origin tracking for names. We verify the correctness of name-fix and identify an interesting class of transformations for which name-fix provides hygiene. We demonstrate the applicability of name-fix for implementing capture-avoiding substitution, inlining, lambda lifting, and compilers for two domain-specific languages. This demonstration accompanies our ECOOP'14 paper "Capture-Avoiding and Hygienic Program Transformations".

**Tutorial: Rust – bare metal programming without the sharp edges***Nicholas Cameron, Mozilla*

**Abstract** Rust is a new language for bare metal (systems and low-level) programming. It provides strong guarantees about memory safety without the overhead or unpredictability of garbage collection. It is designed to support concurrency and parallelism in building platforms that take full advantage of modern hardware. Rust combines powerful and flexible modern programming constructs with explicit control of memory. This control is balanced with the absolute requirement of safety: Rust’s type system and runtime guarantee the absence of data races, buffer overflow, stack overflow, or access to uninitialized or deallocated memory. In this tutorial we’ll work through a series of examples showing how Rust allows for safer and more efficient programming in the systems domain. We’ll see how Rust uses affine and region types to avoid memory leaks and dangling pointers; type parameterisation and type classes for a flexible and performant object system, and how techniques from the functional paradigm can be used in a systems language.

**2.4 Posters**

Posters will be on display close to the coffee break area throughout the main conference. Posters will be manned during the afternoon coffee breaks.

**Component-Based Specification of Software Product Line Architecture**

*Amina Guendouz, Saad Dahlab University (Algeria)*  
*Djamal Bennouar, Akli Mohand OulHadj University (Algeria)*

**An Executable Formal Semantics for PHP**

*Daniele Filaretti, Imperial College London (UK)*  
*Sergio Maffeis, Imperial College London (UK)*

**Safely Composable Type-Specific Languages**

*Cyrus Omar, Carnegie Mellon University, United States*  
*Darya Kurilova, Carnegie Mellon University, United States*  
*Ligia Nistor, Carnegie Mellon University, United States*  
*Benjamin Chung, Carnegie Mellon University, United States*  
*Alex Potanin, Victoria University of Wellington, New Zealand*  
*Jonathan Aldrich, Carnegie Mellon University, United States*

**Using JavaScript and WebCL for Numerical Computations: A Comparative Study of Native and Web Technologies**

*Faiz Khan, McGill University (Canada)*  
*Sujay Kathrotia, McGill University (Canada)*  
*Vincent Foley-Bourgon, McGill University (Canada)*  
*Erick Lavoie, McGill University (Canada)*  
*Laurie Hendren, McGill University (Canada)*

**A Methodology for Developing Retrofitted Type Systems**

*Oscar E. A. Callaú, PLEIAD, University of Chile (Chile)*

*Romain Robbes, University of Chile (Chile)*

*Éric Tanter, University of Chile (Chile)*

**Two recipes for effective static type analysis of dynamic object oriented languages**

*Davide Ancona, Università di Genova (Italy)*

*Andrea Corradi, Università di Genova (Italy)*

**Enforcing Security and Dependability Patterns in Embedded Systems Design (Secure and dependable System by design)**

*Brahim Hamid, Université de Toulouse (France)*

**Data-driven Development of Graphical Programming Tools**

*Marcel Taeumel, University of Potsdam (Germany)*

*Tim Felgentreff, University of Potsdam (Germany)*

*Robert Hirschfeld, University of Potsdam (Germany)*

**Queue Delegation Locking**

*David Klaftenegger, Uppsala University*

*Kostis Sagonas, Uppsala University*

*Kjell Winblad, Uppsala University*

**Capable: Capabilities for Scalability**

*Elias Castegren, Uppsala University*

*Tobias Wrigstad, Uppsala University*





## 3 | Satellite Events

### 3.1 UPMARC Summer School

UPMARC is a centre of excellence at Uppsala, whose mission is to “identify and address the fundamental challenges that will enable all programmers to leverage the potential performance from the ongoing shift to universal parallel computing.” The objective of the summer school is to offer foundational tutorials accompanied by a selection of exiting new emerging technologies and industrial applications in the areas covered by UPMARC, all given by leading scientific and industrial experts of the community.

Schedule at a glance. Note that all UPMARC summer school talks are held in Lecture Hall IX.

#### Schedule Monday 28/7

08.30–10.00	Program logics for relaxed memory consistency I <i>Viktor Vafeiadis</i>
10.00–10.30	Coffee break – coffee is served outside Lecture Hall X
10.00–12.00	An introduction to many-core parallel computing with OpenCL I <i>Simon McIntosh-Smith</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13.30–15.00	Analysis Techniques to Detect Concurrency Errors I <i>Cormac Flanagan and Stephen Freund</i>
15.00–15.30	Coffee break – coffee is served outside Lecture Hall X
15.30–17.00	Short talks and posters by PhD students

#### Schedule Tuesday 29/7

08.30–10.00	Analysis Techniques to Detect Concurrency Errors II <i>Cormac Flanagan and Stephen Freund</i>
10.00–10.30	Coffee break – coffee is served outside Lecture Hall X
10.00–12.00	Program logics for relaxed memory consistency II <i>Viktor Vafeiadis</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13.30–15.00	An introduction to many-core parallel computing with OpenCL II <i>Simon McIntosh-Smith</i>
15.00–15.30	Coffee break – coffee is served outside Lecture Hall X
15.30–17.00	Dr. Streamlove or: How I Learned to Stop Worrying and Love the Flow <i>Viktor Klang</i>

### 3.1.1 Abstracts

#### Analysis Techniques to Detect Concurrency Errors

*Cormac Flanagan, University of California, Santa Cruz, US*

*Stephen Freund, Williams College, US*

**Abstract** Multithreaded software systems effectively exploit multi-core and multi-processor machines. However developing reliable multithreaded software is extremely difficult, due to problems caused by unexpected interference between concurrent threads. Given their non-deterministic, scheduler-dependent nature, bugs caused by unintended thread interference are notoriously difficult to detect, reproduce, and eliminate. At the same time, their presence can have severe consequences.

This tutorial explores analysis techniques to find concurrency errors in large-scale software systems. We cover the theoretical underpinnings, implementation techniques, and reusable infrastructure used to build state-of-the-art analysis tools for verifying a variety of important concurrency properties, such as data race freedom, atomicity or serializability, cooperability, and determinism.

#### An Introduction to Many-Core Parallel Computing with OpenCL

*Simon McIntosh Smith, University of Bristol, UK*

**Abstract** OpenCL is a relatively new open standard for programming heterogeneous parallel computers composed of CPUs, GPUs and other processors. OpenCL consists of a framework to manipulate the host CPU and one or more compute devices plus a C-based programming language for writing programs for the compute devices. Using OpenCL, a programmer can write parallel programs that harness all of the resources of a heterogeneous computer. OpenCL is widely supported by all the main parallel hardware vendors, and so OpenCL programs can be run on almost any CPU or GPU.

In this introductory series of lectures, we will present OpenCL by way of its user friendly C++ API. We will describe OpenCL's model of parallel computation, explaining the key concepts by walking through a series of examples. At the end of the lecture series the attendee should be able to start to write their own OpenCL programs. Pointers will be given to useful online material for anyone interested to continue with OpenCL after the lectures.

#### Program Logics for Relaxed Memory Consistency

*Viktor Vafeiadis, Max Planck Institute for Software Systems*

**Abstract** Traditionally, formal methods for verifying shared-memory concurrent programs have all assumed that executing a multithreaded program corresponds to interleaving the memory accesses performed by the threads.

This assumption, known as Sequential Consistency, is however invalidated by multicore hardware and standard compiler optimisations. As a result, modern language standards (such as C/C++11 and Java) allow more program behaviours than simply thread interleaving—a model known as relaxed or weak memory consistency.

This course will introduce relaxed memory consistency and explain how program logics (such as extensions of separation logic) can be used to understand, debug, and reason about the C/C++ weak memory model.

**Dr. Streamlove or: How I Learned to Stop Worrying and Love the Flow***Viktor Klang, Typesafe*

**Abstract** Stream-based programming holds the promise of unifying IO with traditional Functional Programming manipulation of collections. In this session we will introduce you to Akka Streams—an asynchronous Stream-based programming library with non-blocking back pressure implemented on top of Akka—an implementation of the Actor Model. We’ll demonstrate the Flow API—a lifted representation—as well as a pluggable way of transforming the lifted representation into the execution representation—Flow Materialization. We will also discuss the implications of non-blocking back pressure for fan-in (many producers to single consumer) as well as fan-out (single producer to many consumers) as well as discuss dynamic runtime optimization as well as compile time optimization.

**3.2 ECOOP Summer School**
 Official support for the ECOOP Summer School comes from **Oracle** Labs

Schedule at a glance. Note that all ECOOP summer school talks are held in Lecture Hall IX.

	<b>Wednesday</b>	<b>Thursday</b>	<b>Friday</b>
10:00–12:00	Captain Teach and Pyret: In-Flow Peer-Review of Programming Assignments, <i>Joe Gibbs Politz and Shriram Krishnamurthi</i>	Programming for Social Scientists, <i>Ben Livshits</i>	How to Grow a Language for “Big” Applications, <i>Philipp Haller</i>
13:30–15:00	Build your own Functional Operating System, <i>Anil Madhavapeddy and Richard Mortier</i>	Exercises in Programming Style, <i>Cristina Videira Lopes</i>	Principles of Reactive Programming, <i>Erik Meijer</i>
15:30–17:00	Towards Language Composition, <i>Laurence Tratt</i>	Programming the Network with Frenetic, <i>Nate Foster and Arjun Guha</i>	Hack: Type-Checking at Scale, <i>Julien Verlaquet</i>
17:30–19:00	Benchmarking Computer Systems, <i>Tomas Kalibera</i>	Linearizing the Heap: Thoughts on Communication, Sharing, and the Pervasive Use of Accelerators, <i>Nick Mitchell</i>	Implementing Ruby Using Truffle and Graal, <i>Chris Seaton</i>

**3.2.1 Abstracts****Captain Teach and Pyret: In-Flow Peer-Review of Programming Assignments***Shriram Krishnamurthi, Brown (US)**Joe Gibbs Politz, Brown (US)*

**Abstract** Code review is an integral part of building large-scale systems, and deciding which answer to use from StackOverflow exercises both program comprehension and judgement. Yet students rarely get much practice reading and reviewing code. Captain Teach is a new learning environment for programming that supports peer review even as the assignment is in progress. Assignments are broken up into reviewable chunks, and after submitting each piece, students are asked to review the work of a few other students before continuing. In contrast to peer grading, which happens after the assignment is done, in-flow peer-review gives students a chance to apply what they learn both from others' solutions and from feedback they receive by others. This also naturally increases motivation to perform code review well. Captain Teach is language-agnostic, and several universities are already investigating its use next year. Come learn about how you can make peer-review a part of your educational process, including assignment design and grading implications. We intend for this to be an open discussion rather than just a presentation. If time permits we will briefly show Pyret, a new programming language that happens to have features that make it especially amenable for use in in-flow peer review.

### Build your own Functional Operating System

*Anil Madhavapeddy, University of Cambridge (UK)*

*Richard Mortier, University of Nottingham (UK)*

**Abstract** Mirage proposes a radically different way to build virtual machines for the cloud. Rather than the traditional OS model where functionality is provided in layers, building up from a feature-rich kernel through userspace and language runtimes, Mirage progressively specialises application code, written in OCaml, replacing traditional OS components such as the filesystem, network stack and scheduler, with type-safe libraries. This allows you to code using your usual tools, only making the final push to the cloud once you are happy your code works. Your application becomes a “unikernel”: a sealed, fixed-purpose bootable image that runs directly on the Xen hypervisor without need for a guest OS. As unikernels only link in the libraries explicitly required by the application code, they are very compact: a self-hosting Mirage web server is less than a megabyte in size! We describe the architecture of Mirage, run through some samples, and get you started porting your own homepages onto Mirage. Audience participation is encouraged!

### Towards Language Composition

*Laurence Tratt, King's College London (UK)*

**Abstract** We want better programming languages, but “better” invariably ends up becoming “bigger”. Since we can't keep making our languages bigger, what alternatives do we have? I propose language composition as a possible solution to this long standing problem. Language composition means merging two languages and allowing them to be used together. At its most fine-grained, this could allow multiple programming languages to be used together within a single source file. However, language composition is not a new idea. It has failed in the past because editing composed programs was intolerably difficult and the resulting programs ran too slow to be usable. Without good solutions to these problems, language composition will remain an unrealised ideal. I will show how the work we are doing is beginning to address both aspects. We have built a prototype editor utilising a novel concept ‘language boxes’, which allows one to edit composed programs in a natural way. We are tackling the performance problem by composing together interpreters using meta-tracing, allowing us to build composed VMs with custom JITs that naturally optimise across different language's run-times. While we are much nearer the beginning of

the journey than the end, our initial research has allowed us to build a simple composition of two very different languages: Python and Prolog.

### **Benchmarking Computer Systems**

*Tomas Kalibera, Purdue University*

**Abstract** Comparing computer systems is an important part of computer science, and particularly in programming language and systems research, where performance is so important. Most scientific papers in this field measure performance, and for many the amount of measured performance improvement determines their acceptance and impact. Also, most papers published even at top conferences have significant flaws in their performance comparison methodology. Today's computer systems are so complicated that their behavior is similar to that of objects of the real world (there are also effects that can be modelled as random, there is also a lot of many different effects impacting performance, we are not aware of what many of them are, etc). Experimenting in the real world is covered by experiment design and statistical inference, now mature fields of statistics. In computer systems performance evaluation we are now at the very beginning. We have specific problems, but for now we can improve our performance comparisons even using elementary knowledge of applied statistics. The talk will offer several thoughts and examples on how to do things better with a bit of good will, exploratory data analysis, and statistical bootstrap.

### **Programming for Social Scientists**

*Ben Livshits, Microsoft Research and University of Washington (US)*

**Abstract** Experimental psychology is often jokingly referred to as the study of the college sophomore. In this talk, we show how crowd-sourced polls or surveys can be used to quickly and cost-efficiently collect data on a mass scale for a range of social science queries. We will present InterPoll, a LINQ-based programming abstraction and runtime system which allows novice pollsters and survey writers to easily express and correctly execute crowd-sourced polls and surveys. Through a series of examples, we show how developers can easily express common queries in social science, politics, and other areas of human knowledge via LINQ. Then, we demonstrate how InterPoll's runtime correctly executes said queries, automatically re-weighting responses to queries so as to handle potentially biased crowds.

### **Exercises in Programming Style**

*Cristina Videira Lopes, University of California, Irvine (US)*

**Abstract** Back in the 1940s, a French writer called Raymond Queneau wrote an interesting book with the title *Exercises in Style* featuring 99 renditions of the exact same short story, each written in a different style. In my book "Exercises in Programming Style" (available in June 2014) I shamelessly do the same for a simple program. From monolithic to object-oriented to continuations to relational to publish/subscribe to monadic to aspect-oriented to map-reduce, and much more, you will get a tour through the richness of human computational thought by means of implementing one simple program in many different ways. This is more than an academic exercise; large-scale systems design feeds on these ways of thinking. I will talk about the dangers of getting trapped in just one or two prescribed styles during your career, and the need to truly understand this wide variety of concepts when architecting software.

### Programming the Network with Frenetic

*Nate Foster, Cornell University (US)*

*Arjun Guha, University of Massachusetts Amherst (US)*

**Abstract** Modern networks provide a variety of services including routing, load-balancing, traffic monitoring, authentication, and access control. These services are logically distinct, but they must be implemented on top of low-level networking hardware, which offers no support for modular programming. Network programs are thus written in a monolithic style, which complicates programs, makes reasoning difficult, and frequently leads to failures. In recent years, several research groups have applied ideas from programming languages and formal methods to help make network programs safer and easier to write. The catalyst has been the emergence of software defined networking (SDN) and OpenFlow as a simple and open platform for developing network applications. This tutorial will provide an introduction to languages, abstractions, and reasoning tools for networks, focusing both on the low-level OpenFlow protocol and the high-level Frenetic language. Participants will spend most of the tutorial engaged in programming exercises.

### Linearizing the Heap: Thoughts on Communication, Sharing, and the Pervasive Use of Accelerators

*Nick Mitchell, IBM T J Watson Research Centre (US)*

**Abstract** Many high-level languages have, as their main use case, the encoding of software switches. These programs collate data, and the core essence of their program flow has changed little from the days of green screens and COBOL. Requests arrive, are parsed and dispatched to handlers that, in turn, fetch data from other services and then assemble a report by slotting that data into a static template. Wire protocols must be honored, both in the upstream direction from which the requests emanate, and in the downstream direction in which the data resides. Some conversions are thus an inherent part of these software collators, to cope with protocol and data format mismatches. At least from the perspective of the computational resources, we are left with code that streams through data, performing selections and sorting, gathering and scattering, and other flavors of bit manipulations on this data — in short: streaming bit manipulation.

It is a minor bit of computer science tragedy, then, that the kinds of hardware architectures perfectly suited to host this style of computation, such as specialized circuitry in the form of FPGAs, can only dream of executing the important parts of a typical Ruby, Node.js, or Java application. In large part, this incompatibility stems from the large separation between operational and serialized forms. Programs written in these high-level languages store and operate on data as a graph (where each memory allocation can be separately introspected upon), while wire protocols represent data as a linear stream. Even if one desired to accelerate a portion of a preexisting Ruby or Java computation (let alone the entirety), this mismatch of data storage would result in death by Amdahl's law: the sequential shuttling of data from the operational to a streamable form would dominate the gains of partial acceleration. In this lecture, we will motivate research towards this goal of a more pervasive use of accelerators. We will cover potential changes to languages, memory management, and evaluation semantics.

## How to Grow a Language for “Big” Applications

*Philipp Haller, Typesafe (Switzerland)*

**Abstract** In a few years, the requirements of data-intensive distributed applications, such as large-scale web applications, have changed dramatically, requiring response times orders of magnitude shorter for data in the petabytes. At the same time, the software industry is undergoing a fundamental shift toward Software-as-a-Service, where such “reactive applications” are deployed on cloud-based computing platforms with thousands of multicore processors. Reactive programming aims to explore programming languages, tools, and systems that enable the productive construction of reactive applications that are efficient and reliable. In this tutorial I’ll present recent efforts in the design and implementation of reactive programming abstractions for Scala. More specifically, I’ll show how to leverage implicits and macros in Scala for implementing constructs that are efficient, flexible, and reliable.

## Principles of Reactive Programming

*Erik Meijer, Applied Duality Inc./TU Delft (The Netherlands)*

**Abstract** As computers are moving further apart in space, we can no longer ignore the fundamental effects of programming: exceptions, asynchrony, and dimensionality. In these lectures we will explore the consequences of making these effects explicit in our code and present new programming language constructs and APIs to help developers handle them in their code. In particular we will focus on the language-independent Reactive Extensions (Rx) library that re-implements familiar operators over synchronous collections such as map, filter, fold, ... over asynchronous data streams.

## Hack: Type-Checking at Scale

*Julien Verlaquet, Facebook (US)*

**Abstract** Facebook released a new programming language called Hack. Hack is a gradually typed language that interoperates seamlessly with PHP. More importantly, Hack has been designed to scale on a very large codebases. In this talk, I will highlight the particularities of the type-checker’s architecture. The Hack checker was designed as a daemon watching the filesystem for changes. Making such an architecture scale to millions of lines of code while preserving instantaneous checks comes with many challenges. We believe that it could bring an interesting perspective for other programming languages to review those challenges and see how we overcame them.

## Implementing Ruby Using Truffle and Graal

*Chris Seaton, Oracle Labs and University of Manchester (UK)*

**Abstract** Oracle’s Truffle framework allows languages to be implemented as specialising AST interpreters written in Java. When run on a JVM with the Graal JIT compiler, these interpreters are compiled with partial evaluation, producing optimized machine code. Truffle and Graal also provide high-level access to JVM mechanisms such as dynamic deoptimization that allow us to implement highly speculative optimisations while still supporting the full range of functionality of dynamic languages. The JRuby project uses Truffle and Graal to implement Ruby. This talk will focus on how to use Truffle to implement a real language as part of a runtime already used in production and demonstrate the techniques that we

have found work well. We will address Ruby’s awkward parts and show how to implement them with both simplicity and high performance.

### 3.3 PhD Symposium

The 2014 PhD Symposium provides a forum for both early and late-stage PhD students to present their research and get detailed feedback and advice. The main objectives of this event are:

- to allow PhD students to practice writing clearly and to present their research proposal effectively;
- to get constructive feedback from other researchers;
- to build bridges for potential research collaboration; and
- to contribute to the conference goals through interaction with other researchers at the main conference.

#### 3.3.1 Main Organiser

Beatrice Åkerblom, Stockholm University (Sweden)

#### 3.3.2 Student Panel

Cyrus Omar, Carnegie Mellon University (US)

Sylvan Clebsch, Imperial College (UK)

Michael Homer, Victoria University of Wellington (New Zealand)

Paley Li, Victoria University of Wellington (New Zealand)

#### 3.3.3 Academic Panel

Tomas Kalibera, Purdue University (US)

Doug Lea, SUNY Oswego (US)

Nick Mitchell, IBM Research (US)

### 3.4 Workshops

#### 3.4.1 FTfJP: Formal Techniques for Java-Like Programs

The FTfJP workshop focuses on the use of formal techniques for analyzing programs, precisely describing their behavior, and verifying properties. The workshop focuses on languages such as Java, C#, and Scala which provide good platforms for bridging the gap between formal techniques and practical program development.

*Organiser*

David J. Pearce, Victoria University of Wellington (New Zealand)

#### Programme Committee

David J. Pearce, Victoria University of Wellington (New Zealand) (Chair)

Bart Jacobs, KU Leuven (Belgium)

Alex Summers, ETH Zurich (Switzerland)

Werner Dietl, University of Waterloo (Canada)

Dave Clarke, KU Leuven (Belgium)



Elena Zucca, University of Genoa (Italy)  
 Max Schaefer, Semmler (UK)  
 Nick Cameron, Mozilla Research (New Zealand)  
 Tijs van der Storm, CWI (The Netherlands)

### Steering Committee

Sophia Drossopoulou, Imperial College, London (UK)  
 Werner Dietl, University of Waterloo (Canada)  
 Gary T. Leavens, University of Central Florida, Orlando (USA)  
 K. Rustan M. Leino, Microsoft Research, Redmond (USA)  
 Peter Müller, ETH Zurich (Switzerland)  
 Arnd Poetzsch-Heffter, Universität Kaiserslautern (Germany)  
 Erik Poll, Radboud University Nijmegen (The Netherlands)

### FTfJP Programme

08:45–09:00	Opening Remarks
09:00–10:00	How to prove type soundness of Java-like languages without forgoing big-step semantics <i>Davide Ancona</i> Constraint Semantics for Abstract Read Permissions <i>John Tang Boyland, Peter Müller, Malte Schwerhoff and Alexander J. Summers</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	<b>Keynote:</b> TBA, <i>Erik Ernst</i>
12:00–13:30	Lunch at Norrland's nation (see Page 8)
13:30–15:00	Verifying Functional Behavior of Concurrent Programs <i>Marina Zaharieva-Stojanovski, Marieke Huisman and Stefan Blom</i> Tinygrace: A Simple Structurally Typed Language <i>Timothy Jones and James Noble</i> A Rational Reconstruction of the Escrow Example <i>James Noble and Sophia Drossopoulou</i>
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–16:00	Closing Remarks

### 3.4.2 JSTools: Third Annual Workshop on Tools for JavaScript Analysis

JSTools brings together participants from academia and industry working on analysis of JavaScript and its dialects to share ideas and problems, with a focus on presentations of shareable infrastructure created by the participants. We also aim to involve developers working on JavaScript dialects such as TypeScript to share their perspective.

### Programme Committee

Julian Dolby, IBM (US)

Shu-Yu Guo, Mozilla

Christian Hammer, CISPA, Saarland University, (Germany)

Simon Jensen, Samsung

### JSTools Programme

08:45–09:00	Opening Remarks
09:00–10:00	JIPDA: Reusable and Precise Static Analysis of Real-world JavaScript Programs <i>Jens Nicolay</i> Automatically Verifying JS Libraries without Client Code <i>Arlen Cox</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	Investigating the performance of upcoming web technologies <i>Erick Lavoie</i> Blended Taint Analysis for JavaScript <i>Shiyi Wei</i> Information Flow Control for WebKit’s Bytecode <i>Abhishek Bichhawat</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13:30–15:00	(TBD) Tizen <i>Ming Jin</i> Memory Tooling in Firefox (Work in Progress) <i>Jim Blandy</i> Tooling at Samsung <i>Simon Jensen</i>
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–17:00	A Security Architecture for Server-side JavaScript <i>Frank Piessens</i> Web browser security <i>Rezwana Karim</i> Proving Security Properties about Secure ECMAScript (SES) Programs <i>Philippa Gardner</i>
17:00	Closing Remarks

### 3.4.3 IC000LPS: 9th workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems)

The IC000LPS workshop series brings together researchers and practitioners working in the field of OO languages implementation and optimization. IC000LPS key goal is to identify current and emerging issues relating to the efficient implementation, compilation and optimization of such languages, and outlining future challenges and research directions.

*Organiser*

Laurence Tratt, King’s College London (UK)

**Programme Committee**

Laurence Tratt, King's College London (UK) (Chair)  
 Olivier Zendra, INRIA (France) (Chair)  
 Carl Friedrich Bolz, King's College London (UK)  
 Eric Jul, University of Copenhagen (Denmark)  
 José Manuel Redondo López, Universidad de Oviedo (Spain)  
 Stefan Marr, INRIA Lille (France)  
 Floréal Morandat, Labri (France)  
 Todd Mytkowicz, Microsoft (US)  
 Tobias Pape, Hasso-Plattner-Institut Potsdam (DE)  
 Ian Rogers, Google (US)  
 Jeremy Singer, University of Glasgow (UK)  
 Jan Vitek, Purdue University (US)  
 Mario Wolczko, Oracle Labs (US)

**Steering Committee**

Roland Ducournau, LIRMM (France)  
 Richard Jones, University of Kent (UK)  
 Eric Jul, Universitetet i Oslo (Norway) / Bell Labs (Ireland)  
 Jan Vitek, Purdue University (US)  
 Olivier Zendra, INRIA (France)

**ICOOOLPS Programme**

09:00–10:00	<b>Keynote:</b> Handcrafting VMs for Dynamic Languages: Reality and Dreams, <i>Vyacheslav Egorov, Google</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	An Efficient Approach for Accessing C Data Structures from JavaScript <i>Matthias Grimmer, Thomas Würthinger, Andreas Wöß, Hanspeter Mössenböck</i> Data Interface + Algorithms = Efficient Programs <i>Mattias De Wael, Stefan Marr, Wolfgang De Meuter</i> We are all Economists Now: Economic Utility for Multiple Heap Sizing <i>Callum Cameron, Jeremy Singer</i>
12:00–13:30	Lunch at Norrland's nation (see Page 8)
13:30–15:00	A Way Forward in Parallelising Dynamic Languages <i>Remigius Meier, Armin Rigo</i> Clash of the Lambdas <i>Aggelos Biboudis, Nick Palladinos, Yannis Smaragdakis</i>
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–17:00	The GOOL system: A lightweight Object-Oriented, Programming language trans- lator <i>Pablo Arrighi, Johan Girard, Miguel Lezama, Kévin Mazet</i> Why Inheritance Anomaly Is Not Worth Solving <i>Vincent Gramoli, Andrew E. Santosa</i> Discussion period

### **Keynote Abstract: Handcrafting VMs for Dynamic Languages: Reality and Dreams**

This is a story of what happens when the art of VM construction meets development constraints. This is a first-person narrative about my experience with the V8 and Dart virtual machines, fundamental engineering decisions, and their implications. I will try to show how production VMs are treading the thin line between "practical reality" and "dreams of academia". Finally I intend to share my own dreams on how ideas borrowed from academia could help to solve practical issues in VM-construction.

### **3.4.4 PLE: Workshop on Programming Language Evolution**

Programming languages tend to evolve in response to user needs, hardware advances, and research developments. Language evolution artefacts may include new compilers and interpreters or new language standards. Evolving programming languages is however challenging at various levels. Firstly, the impact on developers can be negative. For example, if two language versions are incompatible (e.g., Python 2 and 3) developers must choose to either co-evolve their codebase (which may be costly) or reject the new language version (which may have support implications). Secondly, evaluating a proposed language change is difficult; language designers often lack the infrastructure to assess the change. This may lead to older features remaining in future language versions to maintain backward compatibility, increasing the language's complexity (e.g., FORTRAN 77 to Fortran 90). Thirdly, new language features may interact badly with existing features, leading to unforeseen bugs and ambiguities (e.g., the addition of Java generics). This workshop brings together researchers and developers interested in programming language evolution, to share new ideas and insights, to discuss challenges and solutions, and to advance programming language design.

#### *Organisers*

Raoul-Gabriel Urma, University of Cambridge (UK)  
 Dominic Orchard, University of Cambridge (UK)  
 Alan Mycroft, University of Cambridge (UK)

#### **Programme Committee**

Raoul-Gabriel Urma, University of Cambridge (UK)  
 Dominic Orchard, University of Cambridge (UK)  
 Robert Bowdidge, Google  
 Sophia Drossopoulou, Imperial College London (UK)  
 Kim Mens, Université catholique de Louvain (Belgium)  
 Alan Mycroft, University of Cambridge (UK)  
 Dominic Orchard, University of Cambridge (UK)  
 Jeff Overbey, Auburn University, AL (US)  
 Chris Parnin, Georgia Institute of Technology (US)  
 Max Schaefer, Semmle Ltd., Oxford (UK)  
 Peter Sommerlad, IFS Institute for Software at FHO/HSR Rapperswil  
 Alexander J. Summers, ETH Zurich (Switzerland)  
 Raoul-Gabriel Urma, University of Cambridge (UK)  
 Louis Wasserman, Google

**Steering Committee**

Raoul-Gabriel Urma, University of Cambridge (UK)

Dominic Orchard, University of Cambridge (UK)

Alan Mycroft, University of Cambridge (UK)

**PLE Programme**

09:00–09:05	Opening ( <b>NOTE:</b> joint with Scala in Lecture Hall X)
09:05–10:05	<b>Keynote:</b> The Evolution of Scala (joint with PLE in Lecture Hall X), <i>Martin Odersky</i>
10:05–10:30	Coffee break – coffee is served outside Lecture Hall X ( <b>NOTE:</b> PLE now continues in Lecture Hall VIII)
10:30–11:20	Full papers session (Chair: Raoul-Gabriel Urma) Programming Language Feature Agglomeration (25min) <i>Jeremy Singer, Callum Cameron and Marc Alexander</i> HERCULES/PL: The Pattern Language of HERCULES (25min) <i>Christos Kartsaklis and Oscar Hernandez</i>
11:20–12:00	Talks session 1 (Chair: Dominic Orchard) Experiences from adding reverse inheritance to Eiffel (20min) <i>Markku Sakkinen</i> Hack: lessons learnt (20min) <i>Julien Verlaquet</i>
12:00–13:30	Lunch at Norrland's nation (see Page 8)
13:30–14:10	Talks session 2 (Chair: Alan Mycroft) Python 2 and 3 compatibility testing via optional run-time type checking (20min) <i>Raoul-Gabriel Urma</i> Evolving Fortran types with inferred units-of-measure (20min) <i>Dominic Orchard</i>
14:10–15:00	Discussion

**Keynote Abstract: The Evolution of Scala**

Scala is one of the relatively few languages that escaped from an academic research lab into widespread industrial usage. This was not something that was planned 10 years ago, when Scala was first announced. In this talk I'll give an overview of the development of Scala from its beginning to today, addressing some questions one might ask when looking back: How did motivations and expectations change? In hindsight, what were the important achievements? What was learned along the way?

**3.4.5 Scala: Fifth Workshop on Scala**

Scala is a general-purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages. The Scala Workshop is a forum for researchers and practitioners to share new ideas and results of interest to the Scala community.



### *Organisers*

Heather Miller, EPFL (Switzerland)  
Philipp Haller, TypeSafe (Switzerland)  
Doug Lea, SUNY Oswego (US)  
Martin Odersky, EPFL/TypeSafe (Switzerland)

### **Programme Committee**

- Heather Miller, EPFL (Switzerland)
- Philipp Haller, TypeSafe (Switzerland)
- Jonathan Aldrich, Carnegie Mellon University (US)
- Michael Armbrust, Databricks (US)
- Werner Dietl, U Waterloo (Canada)
- Marius Eriksen, Twitter (US)
- Shriram Krishnamurthi, Brown (US)
- Doug Lea, SUNY Oswego (US)
- Erik Meijer, Applied Duality Inc./TU Delft (The Netherlands)
- Bruno Oliveira, The University of Hong Kong (China)
- Klaus Ostermann, TU Darmstadt (Germany)
- Aleksandar Prokopec, EPFL (Switzerland)
- Ilya Sergey, IMDEA (Spain)
- Philippe Suter, IBM Research (US)

- Sam Tobin-Hochstadt, Indiana University (US)
- Tom Van Cutsem, Alcatel-Lucent Bell Labs, (Belgium)
- Peter Van Roy, Catholic University of Louvain (Belgium)
- Damien Zufferey, MIT (US)

### Scala Programme

#### Schedule Monday 28/7

09:05–10:05	<b>Keynote:</b> The Evolution of Scala (joint with PLE in Lecture Hall X), <i>Martin Odersky</i>
10:05–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	Typecasting Actors: from Akka to T Akka <i>Jiansen He, Philip Wadler, Philip Trinder</i>  Containers and Aggregates, Mutators and Isolates for Reactive Programming <i>Aleksandar Prokopec, Philipp Haller, Martin Odersky</i>  Type-Directed Language Extension for Effectful Computations <i>Evgenii Kotelnikov</i>
12:00–13:30	Lunch at Norrland's nation (see Page 8)
13:30–13:35	Opening remarks
13:30–14:30	<b>Keynote:</b> Experiences with Scala for Introductory CS, <i>Mark Lewis</i>
14:30–15:00	Short student talks
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–17:30	Resilient Applications with Akka Persistence <i>Jonas Bonér (Industrial talk)</i>  42.type: Literal-based singleton types in Scala <i>George Leontiev (Industrial Talk)</i>  Battle-hardened Scala: how I learnt to stop worrying and push code safely <i>Tim Dinsdale (Industrial Talk)</i>  TBA <i>Martin Schmidt (Industrial Talk)</i>

#### Schedule Tuesday 28/7

09:00–10:00	<b>Keynote:</b> Programming with Dependent Types in Idris, <i>Edwin Brady</i>
10:30–11:00	Coffee break – coffee is served outside Lecture Hall X
10:30–12:10	Towards Virtual Traits in Scala <i>Manuel Weiel, Ingo Maier, Sebastian Erdweg, Michael Eichberg, Mira Mezini (Long)</i>  Accelerating Parser Combinators with Macros <i>Eric Béguet, Manohar Jonnalagedda (Long)</i>  MorphScala: Safe Class Morphing with Macros <i>Aggelos Biboudis, Eugene Burmako (Short)</i>  ScalaDyno: Making Name Resolution and Type Checking Fault-Tolerant <i>Cédric Bastin, Vlad Ureche, Martin Odersky (Short)</i>

12:10–13:30	Lunch at Norrland’s nation (see Page 8)
13:30–14:30	SciFe: Scala Framework for Efficient Enumeration of Data Structures w/ Invariants <i>Ivan Kuraĵ, Viktor Kuncak (Short)</i> Real-Time Collaborative Scala Development with Clide <i>Martin Ring, Christoph Lüth (Short)</i> Improving the Performance of Scala Collections with Miniboxing <i>Aymeric Genêt, Vlad Ureche, Martin Odersky (Short)</i>
14:30–15:00	Short student talks
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–16:00	Abstracting over method return types using Return-Type Strategies, <i>Jon Pretty</i> (Open Source Talk)
16:15–17:30	Panel: Scala and Next-Generation Languages: Language Design for Mainstream Software Engineering <i>Panelists: Jonathan Aldrich, Edwin Brady, Martin Odersky, Peter Van Roy</i>

### Keynote abstract: The Evolution of Scala

Scala is one of the relatively few languages that escaped from an academic research lab into widespread industrial usage. This was not something that was planned 10 years ago, when Scala was first announced. In this talk I’ll give an overview of the development of Scala from its beginning to today, addressing some questions one might ask when looking back: How did motivations and expectations change? In hindsight, what were the important achievements? What was learned along the way?

### Keynote Abstract: Experiences with Scala for Introductory CS

At Trinity University, we have been using Scala for some of our CS1 and CS2 courses since fall of 2010, and for all such sections since fall of 2012. Given that a common critique of Scala is that it is simply too complex, this might seem like an odd choice for introductory CS courses, where many of the students have no previous programming experience. There have been various rebuttals to the idea that Scala is too complex, and I believe that usefulness as a starting language can be yet another. In this talk I will describe how we came to consider Scala for our curriculum, and our experiences with it. I will look at what makes Scala a good choice as an introductory language for our program and mention some of the things that can trip students up.

### Keynote Abstract: Programming with Dependent Types in Idris

Idris is a general purpose programming language with dependent types, building on state-of-the-art techniques in programming language research. Dependent types allow types to be predicated on any value - in this way, required properties of a program can be captured in the type system, and verified by a type checker. This includes functional properties (i.e. does the program give the correct answer) and extra-functional properties (i.e. does the program run within specified resource constraints).

In this talk, I will use a series of examples to show how dependent types in Idris may be used for verifying realistic and important properties of software, from simple properties such as array bounds verification, to more complex properties of communicating and distributed systems. I will also briefly discuss the relationship between the Idris and Scala type systems, and consider how type-based verification techniques may in the future become usable in more mainstream languages.



**Panel Abstract**

Scala and other next-generation languages promise to provide substantial benefits for mainstream software engineering by providing better abstraction capabilities, better modularity, and increased performance, among others. The development of new type systems enables checking richer program properties statically, thereby increasing the reliability of software systems. This panel discusses which aspects of mainstream software engineering will in the future benefit from new programming languages research and why.

**Sponsors**

TypeSafe, SoundCloud, Goldman Sachs, innoQ

**3.4.6 COP: 6th International Workshop on Context-Oriented Programming**

Context information plays an increasingly important role in our information-centric world. Software systems must adapt to changing contexts over time, and must change even while they are running. Context-Oriented Programming (COP) has emerged as a solution to directly support variability depending on a wide range of dynamic attributes, making it possible to dispatch run-time behaviour on any property of the execution context. The goal of the 6th International Workshop on Context-Oriented Programming (COP'14) is to further establish context orientation as a common thread to language design, application development, and system support.

*Organisers*

Tomoyuki Aotani, Tokyo Institute of Technology (Japan)  
Malte Appeltauer, SAP Innovation Center Potsdam (Germany)  
Robert Hirschfeld, Hasso-Plattner-Institut Potsdam (Germany)  
Atsushi Igarashi, Kyoto University (Japan)  
Hidehiko Masuhara, Tokyo Institute of Technology (Japan)

**Programme Committee**

Tomoyuki Aotani, Tokyo Institute of Technology (Japan)  
Shigeru Chiba, The University of Tokyo (Japan)  
Dave Clarke, Uppsala University, Sweden and KU Leuven (Belgium)  
Marcus Denker, INRIA (France)  
Richard P. Gabriel, IBM (US)  
Sebastián González, UC Louvain (Belgium)  
Robert Hirschfeld, Hasso-Plattner-Institut Potsdam (Germany)  
Tetsuo Kamina, Ritsumeikan University (Japan)  
Jens Lincke, Hasso-Plattner-Institut Potsdam (Germany)  
Friedrich Steimann, Fernuniversität in Hagen (Germany)  
Eddy Truyen, K.U. Leuven (Belgium)

**COP Program**

09:15–09:30	Opening
09:30–10:00	Session 1 Applying Data-driven Tool Development to Context-oriented Languages <i>Marcel Taeumel, Tim Felgentreff and Robert Hirschfeld</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	Session 2 Run-time Validation of Behavioral Adaptations <i>Nicolás Cardozo, Laurent Christophe, Coen De Roover and Wolfgang De Meuter</i> Towards Type-Safe JCop <i>Hiroaki Inoue, Atsushi Igarashi, Malte Appeltauer and Robert Hirscheferd</i> A Two-Component Language for COP <i>Pierpaolo Degano, Gianluigi Ferrari and Letterio Galletta</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13:30–15:00	Session 3 Program Execution Environments as Contextual Values <i>Markus Raab and Franz Puntigam</i> A Reflective Approach to Actor-Based Concurrent Context-Oriented System <i>Takuo Watanabe and Souhei Takeno</i> Unifying multiple layer activation mechanisms using one event sequence <i>Tomoyuki Aotani, Tetsuo Kamina and Hidehiko Masuhara</i>
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–16:00	Session 4 On-Demand Layer Activation for Type-Safe Deactivation <i>Tetsuo Kamina, Tomoyuki Aotani and Atsushi Igarashi</i>
16:00–17:00	Open discussion

**3.4.7 IWACO: International Workshop on Aliasing, Capabilities, and Ownership**

The IWACO workshop addresses the question how to reason about stateful (sequential or concurrent) programs. In particular, we will consider the following issues (among others): models, type and other formal systems, programming language mechanisms, analysis and design techniques, patterns and notations for expressing ownership, aliasing, capabilities, uniqueness, and related topics; optimization techniques, analysis algorithms, libraries, applications, and novel approaches exploiting ownership, aliasing, capabilities, uniqueness, and related topics; empirical studies of programs or experience reports from programming systems designed with these issues in mind; programming logics that deal with aliasing and/or shared state, or use ownership, capabilities or resourcing; applications of any of these techniques to a concurrent setting.

*Organisers*

Stephanie Balzer, Carnegie Mellon University (US)

Johan Östlund, Uppsala University (Sweden)

**Programme Committee**

Werner Dietl, University of Waterloo (Canada)  
 Colin Gordon, University of Washington (US)  
 Ana Milanova, Rensselaer Polytechnic Institute (US)  
 Greg Morrisett, Harvard University (US)  
 Frank Pfenning, Carnegie Mellon University (US)  
 Francois Pottier, INRIA (France)  
 Alex Summers, ETH Zurich (Switzerland)  
 Aaron Turon, Max Planck Institute for Software Systems (Germany)  
 Jan Vitek, Purdue University (US)  
 Janina Voigt, Cambridge University (UK)

**Steering Committee**

Dave Clarke, Uppsala University (Sweden)  
 Sophia Drossopoulou, Imperial College (UK)  
 Peter Müller, ETH Zurich (Switzerland)  
 James Noble, Victoria University of Wellington (New Zealand)  
 Matthew Parkinson, Microsoft Research (US)  
 Tobias Wrigstad, Uppsala University (Sweden)

**IWACO Programme**

09:00–09:10	Welcome
09:10–10:00	Language-Based Architectural Control <i>Jonathan Aldrich, Cyrus Omar, Alex Potanin and Du Li</i> Owners as Trusters: Trusted Ownership Declassification With Neighbourhood <i>Pradeepkumar Duraisamy Soundrapandian and Janardan Misra (short paper)</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–12:00	Capable: Capabilities for Scalability <i>Elias Castegren and Tobias Wrigstad</i> On Owners as Accessors <i>James Noble, Sophia Drossopoulou and Alex Potanin</i> Dynamic Semantics of Sheep cloning: Proving Cloning <i>Paley Li, Nicholas Cameron and James Noble</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13:30–14:30	<b>Keynote:</b> Memory Safety without Garbage Collection—aliasing, regions, uniqueness, and immutability in Rust, <i>Nicholas Cameron</i>
14:30–15:00	Structured Discussion (moderator: James Noble)
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–16:15	Structured Discussion (moderator: James Noble)

**Keynote Abstract: Memory Safety without Garbage Collection...**

Systems programming requires unfettered access to memory and predictable, low-cost abstractions. As such, garbage collection is impractical. However, years of experience have shown that programming in a memory unsafe language (such as C or C++) results in subtle and hard to find bugs, often with severe implications for security. Rust solves these constraints using static type checking. Rust's type system tracks aliasing and prevents memory leaks, dangling pointers, use-after-free errors, data races, and other sources of memory unsafety, all without run-time overhead.

In this talk, I will give a brief overview of Rust's syntax and semantics; describe Rust's system of alias control—borrowed references, explicit lifetimes, unique pointers—and its implementation; show how these features conspire to ensure memory safety, including in a multi-threaded environment; show how uniqueness and mutability are coordinated to prevent data races and make programs easier to reason about; and show how the type system can be safely circumvented using dynamic checking, where necessary.

**3.4.8 PLAS: ACM Ninth Workshop on Programming Languages and Analysis for Security**

PLAS aims to provide a forum for exploring and evaluating ideas on the use of programming language and program analysis techniques to improve the security of software systems. Strongly encouraged are proposals of new, speculative ideas, evaluations of new or known techniques in practical settings, and discussions of emerging threats and important problems.

*Organisers*

Omer Tripp, Omer Tripp, IBM TJ Watson Research Center (US)  
Alejandro Russo, Chalmers University (Sweden)

**Programme Committee**

Omer Tripp, Omer Tripp, IBM TJ Watson Research Center (US)  
Alejandro Russo, Chalmers University (Sweden)  
Benjamin C. Pierce, University of Pennsylvania (US)  
Boris Köpf, IMDEA (Spain)  
Christos Dimoulas, Harvard University (US)  
David Naumann, Stevens Institute of Technology (US)  
Frank Piessens, KU Leuven (Belgium)  
Marco Pistoia, IBM TJ Watson Research Center (US)  
Paolina Centonze, Iona College (US)  
Stephen McCamant, University of Minnesota (US)

**Steering Committee**

Aslan Askarov, Cornell (US)  
Anindya Banerjee, IMDEA (Spain)  
Deepak Garg, Carnegie Mellon University (US)  
Joshua D. Guttman, Worcester Polytechnic Institute (US)  
Sergio Maffei, Imperial College London (UK)  
Prasad Naldurg, Microsoft Research (US)  
Tamara Rezk, INRIA (France)  
Nikhil Swami, Microsoft (US)  
Steve Zdancewic, University of Pennsylvania (US) (Chair)

**PLAS Program**

09:00–10:00	<b>Keynote:</b> To Dream the Impossible Dream: Toward Static Taint Analysis for JavaScript Security, <i>Julian Dolby</i>
10:00–10:30	Coffee break – coffee is served outside Lecture Hall X
10:30–11:00	You Sank My Battleship! A Case Study in Secure Programming <i>Alley Stoughton, Andrew Johnson, Samuel Beller, Karishma Chadha, Dennis Chen, Kenneth Foner and Michael Zhivich</i>  Generalizing Permissive-Upgrade in Dynamic Information Flow Analysis <i>Abhishek Bichhawat, Vineet Rajani, Deepak Garg and Christian Hammer</i>  Building Secure Systems with LIO (Demo) <i>Deian Stefan</i>
12:00–13:30	Lunch at Norrland’s nation (see Page 8)
13:30–15:00	A Language-Based Approach to Secure Quorum Replication <i>Lantian Zheng and Andrew Myers</i>  Operational Semantics for Secure Interoperation <i>Adriaan Larmuseau, Marco Patrignani and Dave Clarke</i>  Domain-Polymorphic Programming of Privacy-Preserving Applications <i>Dan Bogdanov, Peeter Laud and Jaak Randmets</i>
15:00–15:30	Coffee break – coffee is served outside Lecture Hall X
15:30–16:30	Monitoring Reactive Systems with Dynamic Channels <i>Dante Zamarin and Mauro Jaskielof</i>  Paragon: Programming with Information Flow Control (Demo) <i>Niklas Broberg</i>

**Keynote Abstract: To Dream the Impossible Dream...**

As the Web becomes the dominant interface to ever more aspects of life, we become ever more dependent upon Web technology to protect our security and privacy. However, the impossible dream refers to the difficulty in ensuring that current Web technology, which is heavily based on JavaScript, actually does this. In this talk, I shall discuss our experience at IBM Research in building static program analysis to check security properties.

Static program analysis for JavaScript has to navigate between a modern version of the Scylla and Charybdis of Greek legend: attempts at soundness that become intractable on the one hand, and shortcuts that result in missing important parts of the program on the other. While much progress has been made on traditional program analysis for JavaScript (e.g. [3, 4]), doing conservative analysis remains challenging; our recent work has therefore focused on various forms of unsoundness to make analysis tractable (e.g. [1]).

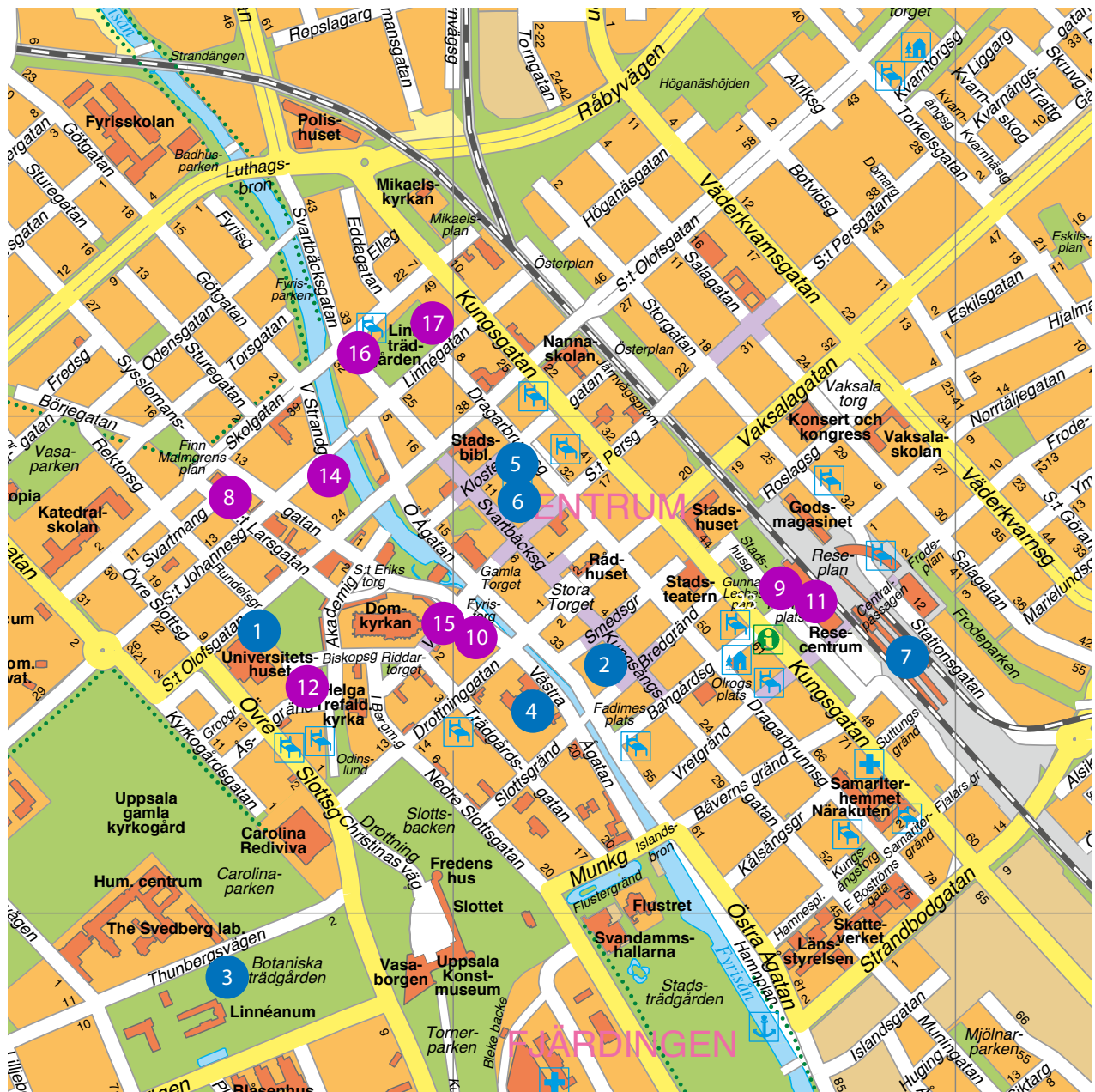
The difficulty of sound analysis impacts clients built on such analysis, both in terms of cost and precision. Our focus is on security analysis, in particular taint analysis. This has often been defined based on a call graph and pointer analysis (e.g. [2]). However, as framework-based JavaScript code has become ubiquitous, our ability to create precise call graphs and pointer analyses has become ever more limited. Hence, we have been exploring how more approximate techniques can support our security analyses.

To support our security analysis clients, such analysis must address two key challenges arising from the imprecise and incomplete data available from our approximate analysis: 1) Our imprecise call graph conflates methods with the same property name; however, e.g. `toString` is a source of taint in some cases and it is not acceptable to decide that all calls to such a ubiquitous method are sources of taint. 2) We do not have a complete pointer analysis approximation available, and hence we need to name sources of taint without relying on a heap abstraction. However, sources of taint in JavaScript are often complex paths in the heap.

To address these challenges, we have formulated a taint analysis *Messina2* that exploits JavaScript semantics and flow-sensitive analysis machinery to address these challenges: 1) We attempt to recover precision in the face of imprecise call graphs and a lack of pointer analysis by using a flow-sensitive propagation of access paths. (Note that the single-threaded semantics of JavaScript makes this sound). 2) We generalize traditional access paths to encompass method calls: an access path could be e.g. `document.getElementById(...).value`. This is done to adapt to a lack of pointer analysis that is typically needed to understand the values returned by functions. 3) We express finding sources and sinks for taint analysis as a separate analysis expressed using access paths.

I will present our analysis primarily through a series of examples, and discuss our experience so far that suggests this form of analysis is a practical approach that can, to a great extent, overcome the limitations of current static analysis technology.

## 4 | Map of Downtown Uppsala



**ECOOP Venues and Other “Important” Places**

Blue dots.

1. Conference venue. “Universitetshuset” (Main University Building). For address see
2. Ski Total (Bike Rental).
3. Workshop reception—The Botanical Gardens.
4. All lunches—Norrländ’s nation.
5. Hotell Clarion Gillet. (Many ECOOP participants stay here.)
6. Pharmacy—S:t Pers Apotek (inside the shopping centre).
7. Uppsala Central Stations. Trains, Taxis and Buses to Arlanda, Stockholm and beyond.

**Restaurants**

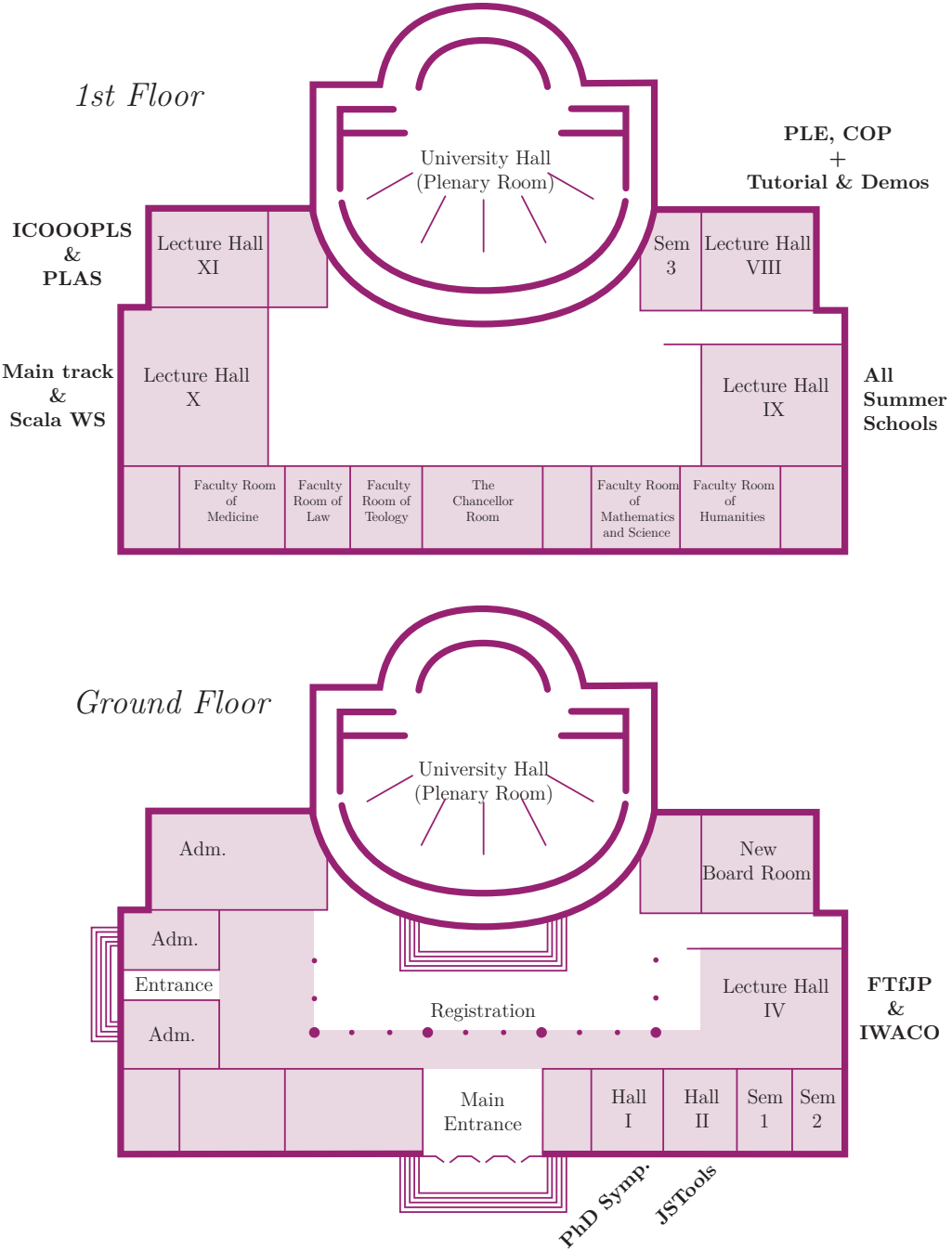
Purple dots.

8. Aaltos—Italian grill inside a piece of lovely architecture built by renowned Finnish architect Alvar Aalto. (Address: Sysslomansgatan 14.)
9. Dryck—At this little pub with just 17 seats focusing on drinks. Each week, a new drink menu is prepared accompanied by three complementing dishes. (Address: Olof Palmes Plats 4.)
10. Hambergs Fisk—Seafood restaurant with genuine interest in marine life. (Address: Fyrstorg 8.)
11. Stationen—continental dining inspired by Paris. (Address: Olof Palmes Plats 6.)
12. Villa Anna—rustic Swedish cuisine mostly from local produce and high-quality organic produce. Quite pricy, but excellent. (Address: Odinslund 3.)
13. Amandas Mat & Vinstudio. Run by one of Uppsala’s best accomplished restaurant keepers. (Address: Fyrislundsgatan 81, not on the map.)
14. Borgen. Good, classic Swedish food. (Address: Orphei Drängars plats 1.)
15. Domtrappkällaren. Normally great food. Huge outdoor eating area when the weather is good. (Address: S:t Eriks gränd 15.)
16. Lingon. Chefs serve you directly. (Address: Svartbäcksgatan 30.)
17. Smultron. Outdoor seating in beautiful gardens. (Address: Svartbäcksgatan 27.)



5

Schedule Overview & Venue Map



## Wednesday

08.15–08.30	Opening: Tobias Wrigstad
08.30–09.30	<b>Keynote:</b> Luca Cardelli (see abstract on Page 17)
09.30–10.00	Coffee
10.00–12.00	<b>Technical Papers I: Analysis</b> State-Sensitive Points-to Analysis for the Dynamic Behavior of JavaScript Objects <i>Shiyi Wei and Barbara G. Ryder</i> Self-Inferencing Reflection Resolution for Java <i>Yue Li, Tian Tan, Yulei Sui and Jingling Xue</i> Constructing Call Graphs of Scala Programs <i>Karim Ali, Marianna Rapoport, Ondřej Lhoták, Julian Dolby and Frank Tip</i> Finding Reference-Counting Errors in Python/C Programs with Affine Analysis <i>Siliang Li and Gang Tan</i>
12.00–13.30	Lunch at Norrland's nation
13.30–15.00	<b>Technical Papers II: Design</b> Safely Composable Type-Specific Languages <i>Cyrus Omar, Darya Kurilova, Ligia Nistor, Benjamin Chung, Alex Potanin and Jonathan Aldrich</i> Graceful Dialects <i>Michael Homer, Timothy Jones, James Noble, Kim B. Bruce and Andrew P. Black</i> Structuring Documentation to Support State Search: A Laboratory Experiment about Protocol Programming <i>Joshua Sunshine, James D. Berbsleb and Jonathan Aldrich</i>
15.00–15.30	Coffee
15.30–17.00	<b>Technical Papers III: Concurrency</b> Reusable Concurrent Data Types <i>Vincent Gramoli and Rachid Guerraoui</i> TaDA: A Logic for Time and Data Abstraction <i>Pedro da Rocha Pinto, Thomas Dinsdale-Young and Philippa Gardner</i> Infrastructure-Free Logging and Replay of Concurrent Execution on Multiple Cores <i>Kyu Hyung Lee, Dohyeong Kim and Xiangyu Zhang</i>
17.15–18.45	Demonstrations (see page 32 for details)

## Thursday

08.30–09.30	<b>Dahl-Nygaard Award Keynote:</b> Tudor Gîrba (see abstract on Page 18)
09.30–10.00	Coffee
10.00–12.00	<b>Technical Papers IV: Types</b> Understanding TypeScript <i>Gavin Bierman, Martín Abadi and Mads Torgersen</i> Sound and Complete Subtyping between Coinductive Types for Object-Oriented Languages <i>Davide Ancona and Andrea Corradi</i> Spores: A Type-Based Foundation for Closures in the Age of Concurrency and Distribution <i>Heather Miller, Philipp Haller and Martin Odersky</i> Rely-Guarantee Protocols <i>Filipe Militão, Jonathan Aldrich and Luís Caires</i>
12.00–13.30	Lunch at Norrland's nation
13.30–15.00	<b>Technical Papers V: Implementation</b> Stream Processing with a Spreadsheet <i>Mandana Vaziri, Olivier Tardieu, Rodric Rabbah, Philippe Suter and Martin Hirzel</i> Implicit Staging of EDSL Expressions: A Bridge Between Shallow and Deep Embedding <i>Maximilian Scherr and Shigeru Chiba</i> Babelsberg/JS - A Browser-based Implementation of an Object Constraint Language <i>Tim Felgentreff, Alan Borning, Jens Lincke, Robert Hirschfeld, Yoshiki Ohshima, Bert Freudenberg and Robert Krahn</i>
15.00–15.30	Coffee
15.30–16.15	Evaluated Artifacts (see page 27 for details)
16.15–17.15	<b>Dahl-Nygaard Award Keynote:</b> Robert France (see abstract on Page 19)
17.20–18.45	Demonstrations and Tutorial (see page 32 for details)

## Friday

08.30–09.30 **Dahl-Nygaard Award Keynote:** William Cook (see abstract on Page 19)

09.30–10.00 Coffee

10.00–12.00 **Technical Papers VI: Refactoring**

Automated Multi-Language Artifact Binding and Refactoring between Java and DSLs  
used by Java Frameworks

*Philip Mayer and Andreas Schroeder*

Retargetting Legacy Browser Extensions to Modern Extension Frameworks

*Rezwana Karim, Mohan Dhawan and Vinod Ganapathy*

Capture-Avoiding and Hygienic Program Transformations

*Sebastian Erdweg, Tijs van der Storm and Yi Dai*

Converting Parallel Code from Low-Level Abstractions to Higher-Level Abstractions

*Semih Okur, Cansu Erdogan and Danny Dig*

12.00–13.30 Lunch at Norrland's nation

13.30–15.00 **Technical Papers VII: Javascript, PHP and Frameworks**

Portable and Efficient Run-time Monitoring of JavaScript Applications using Virtual  
Machine Layering

*Erick Lavoie, Bruno Dufour and Marc Feeley*

An Executable Formal Semantics of PHP

*Daniele Filaretti and Sergio Maffeis*

Identifying Mandatory Code for Framework Use via a Single Application Trace

*Naoya Nitta, Izuru Kume and Yasuhiro Takemura*

15.00–15.30 Coffee

15.30–17.00 **Technical Papers VIII: Parallelism**

Cooperative Scheduling of Parallel Tasks with General Synchronization Patterns

*Shams Imam and Vivek Sarkar*

MiCA: A Compositional Architecture for Gossip Protocols

*Lonnie Princehouse, Rakesh Chenchu, Zhefu Jiang, Kenneth Birman, Nate Foster  
and Robert Soulé*

Semantics of (Resilient) X10

*Silvia Crafa, David Cunningham, Vijay Saraswat, Avraham Shinnar and Olivier  
Tardieu*