ASP_{fun}: a Deadlock-free Calculus for Distributed Active Objects

Florian Kammüller

Institut für Softwaretechnik und Theoretische Informatik

Synchrony and Asynchrony in Distributed Systems

Braunschweig, tubs.CITY, Haus der Wissenschaft

1. Juli 2009

Motivation and goals

- New language ASP_{fun}
  - functional
  - active objects
  - distributed
  - plus typing
- Formal, mechanically supported language development
  - "Killer-Application" of theorem proving in Higher Order Logic (HOL)
  - Java (with JVM) completely formalized in Isabelle/HOL (Tobias Nipkow, TU München)
  - Complete re-engineering of a C-Compiler in Coq (Xavier Leroy, INRIA Roquencourt)

$\Longrightarrow$ ASP_{fun} in Isabelle/HOL

Overview

1. ASP_{fun}
2. ASP_{fun} in Isabelle/HOL
3. Example for ASP_{fun}
4. Results, Discussion, Outlook

ASP$_{fun}$– Asynchronous Sequential Processes – functional

- ProActive (Inria/ActiveEON): Java API for active objects

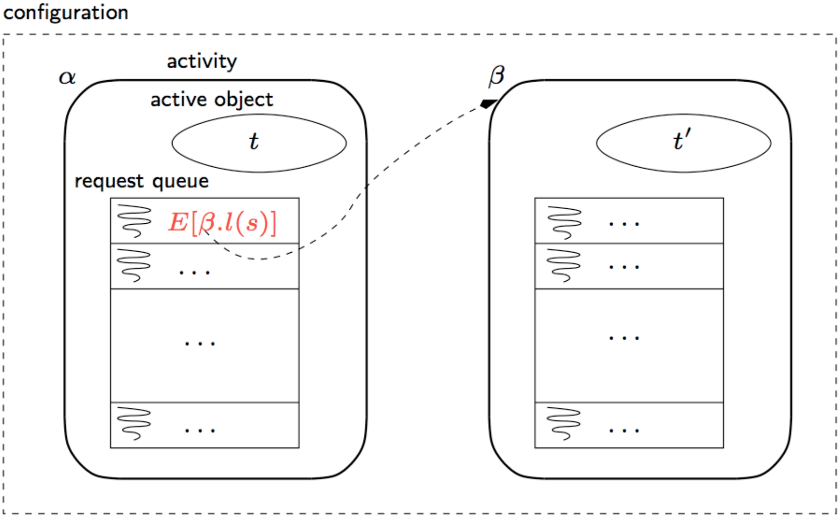
ProActive
Programming, Composing, Deploying on the Grid

- New calculus ASP$_{fun}$ for ProActive
- Functional better properties: many applications can be seen as *functions*, for example web-services
- Asynchronous communication with *futures*
  - Futures : asynchronous method calls
  - Objects: ς-calculus of Abadi/Cardelli
  - Future access may cause deadlock: *wait-by-necessity*
  - Functional: *reply* with partially evaluated *requests*

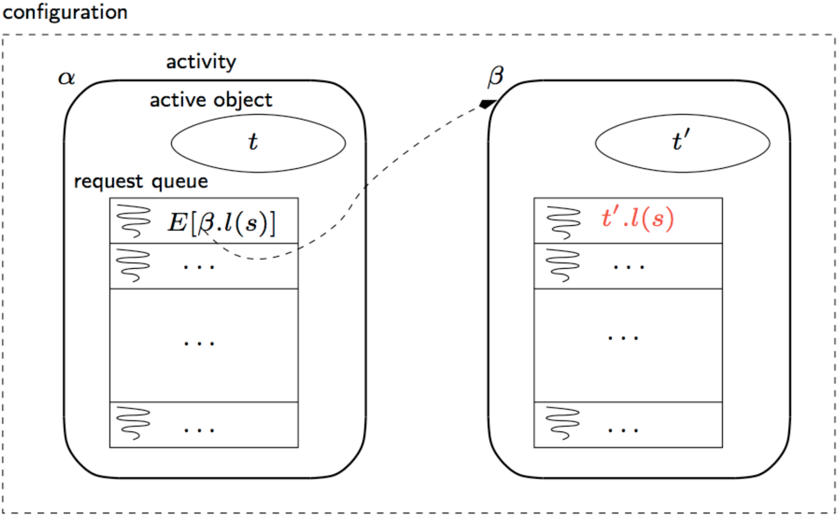
$\Rightarrow$ ASP$_{fun}$ avoids deadlocks when accessing futures

ASP$_{fun}$

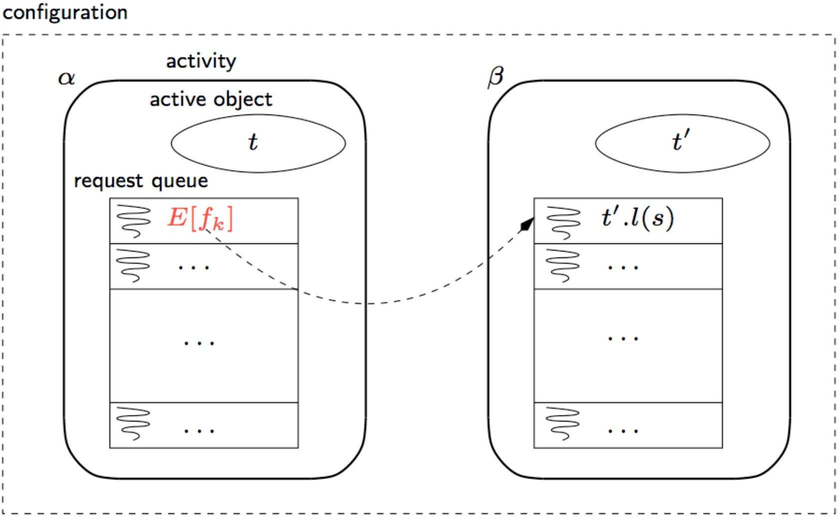
ASP$_{fun}$: at a glance



ASP$_{fun}$: at a glance



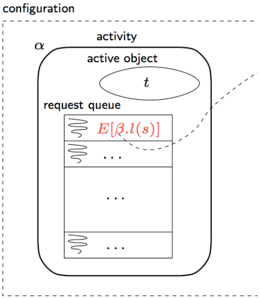
ASP$_{fun}$: at a glance



From ς-calculus to ASP$_{\text{fun}}$

Syntactic extension of ς-calculus by:

- *Active*: creation of a new active object
- *FutRef* and *ActRef*: references for activities and futures (transparent)



- Semantics: *local* $\to_\varsigma$ and *parallel* evaluation
- Parallel semantics $\to_\parallel$: inductive relation on configurations

$$\textit{configuration} = \textit{ActRef} \rightharpoonup (\textit{FutRef} \rightharpoonup \textit{term}) \times \textit{term}$$

Parallel semantics $\rightarrow_{\parallel}$ informally

idea: evaluate terms (only) in future-lists

- LOCAL: reduction $\rightarrow_{\varsigma}$ of ς-calculus
- REQUEST: *method call* $\beta.l$ creates new future f_k in future-list of activity β
- REPLY: *return result*, i.e. replace future f_k by referenced result term

$$\text{REPLY}\ \frac{\beta[f_k \mapsto s :: R, t'] \in \alpha[f_i \mapsto E[f_k] :: Q, t] :: C}{\alpha[f_i \mapsto E[f_k] :: Q, t] :: C \rightarrow_{\parallel} \alpha[f_i \mapsto E[s] :: Q, t] :: C}$$

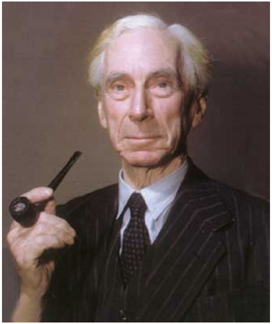
- UPDATE-AO: *update activity*, creates a copy on which update (change) - original remains the same (*immutable*)

Overview

1. ASP_{fun}
2. ASP_{fun} in Isabelle/HOL
3. Example for ASP_{fun}
4. Results, Discussion, Outlook

Language development in Isabelle/HOL

- Isabelle/HOL: interactive theorem prover for HOL
- Enables formalization of syntax, semantics, and type systems of languages
- Proofs of language properties



- Example property: typing is unique
  $\vdash x : T \wedge \vdash x : T' \Rightarrow T = T'$

$\Longrightarrow$ interactive proof tool enables control (and code generation)

ASP_{fun} is type safe and deadlock free

- Proof of properties in Isabelle/HOL, for example
  - Wellformedness: no dangling activity references or futures
  - Typing implies wellformedness
- Type safety: preservation and progress

ASP$_{fun}$ is type safe and deadlock free

- Proof of properties in Isabelle/HOL, for example
  - Wellformedness: no dangling activity references or futures
  - Typing implies wellformedness
- Type safety: preservation and progress

Theorem (Preservation)

$$\vdash C : \langle \Gamma_{act}, \Gamma_{fut} \rangle \wedge C \to_{\parallel} C' \Longrightarrow \exists\, \Gamma'_{act}, \Gamma'_{fut} \,.\; \vdash C' : \langle \Gamma'_{act}, \Gamma'_{fut} \rangle$$

where $\Gamma_{act} \subseteq \Gamma'_{act} \wedge \Gamma_{fut} \subseteq \Gamma'_{fut}$

ASP_{fun} is type safe and deadlock free

- Proof of properties in Isabelle/HOL, for example
  - Wellformedness: no dangling activity references or futures
  - Typing implies wellformedness
- Type safety: preservation and progress

Theorem (Preservation)

$$\vdash C : \langle \Gamma_{act}, \Gamma_{fut} \rangle \wedge C \rightarrow_{\parallel} C' \Longrightarrow \exists\, \Gamma'_{act}, \Gamma'_{fut} \,.\; \vdash C' : \langle \Gamma'_{act}, \Gamma'_{fut} \rangle$$

where $\Gamma_{act} \subseteq \Gamma'_{act} \wedge \Gamma_{fut} \subseteq \Gamma'_{fut}$

Theorem (Progress)

$$\vdash C : \langle \Gamma_{act}, \Gamma_{fut} \rangle \;\wedge\; \alpha[f_i \mapsto a :: Q, t] \in C$$
$$\Longrightarrow isvalue(a) \vee \exists\, C' \,.\; C \rightarrow_{\parallel} C'$$

ASP_{fun} is type safe and deadlock free

- Proof of properties in Isabelle/HOL, for example
  - Wellformedness: no dangling activity references or futures
  - Typing implies wellformedness
- Type safety: preservation and progress

Theorem (Preservation)

$$\vdash C : \langle \Gamma_{act}, \Gamma_{fut} \rangle \wedge C \rightarrow_{\parallel} C' \Longrightarrow \exists\, \Gamma'_{act}, \Gamma'_{fut} \,.\, \vdash C' : \langle \Gamma'_{act}, \Gamma'_{fut} \rangle$$

$$\textit{where } \Gamma_{act} \subseteq \Gamma'_{act} \wedge \Gamma_{fut} \subseteq \Gamma'_{fut}$$

Theorem (Progress)

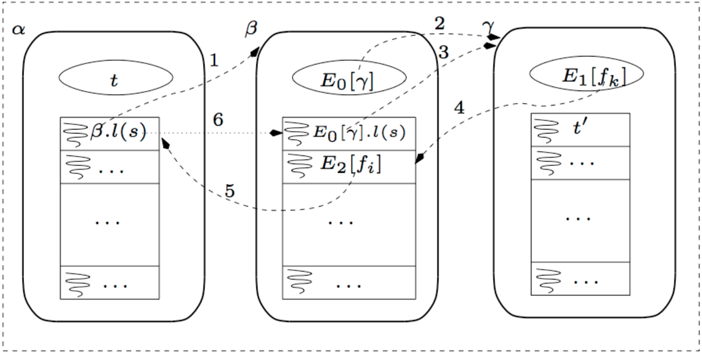
$$\vdash C : \langle \Gamma_{act}, \Gamma_{fut} \rangle \;\wedge\; \alpha[f_i \mapsto a :: Q, t] \in C$$

$$\Longrightarrow \textit{isvalue}(a) \vee \exists\, C' \,.\; C \rightarrow_{\parallel} C'$$

$\Rightarrow$ Always progress, hence no deadlock!

Further results

- *Cycles of futures*: reduction introduces no cycles



- General results for Isabelle/HOL
  - `FMaps`: axiomatic type classes for *finite maps*
  - *Theory of Objects* ς
  - Contexts: “contextual semantics”, à la $E(\bullet)$
  - *Locally nameless* for ς

Overview

1. ASP_{fun}
2. ASP_{fun} in Isabelle/HOL
3. Example for ASP_{fun}
4. Results, Discussion, Outlook

Example: service broker

Client reserves a hotel using a *broker*

Example: service broker

Client reserves a hotel using a *broker*

$$\text{customer}[f_0 \mapsto \text{broker.find(date,limit)}, \varnothing]$$
$$\parallel \text{broker}[\varnothing, [\text{find} = \varsigma(x, (\mathit{date}, \mathit{limit})).\text{hotel.room}(\mathit{date}), \ldots]]$$
$$\parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots]$$

Example: service broker

Client reserves a hotel using a *broker*

$$
\begin{aligned}
& \text{customer}[f_0 \mapsto \text{broker.find(date,limit)}, \varnothing] \\
& \parallel \text{broker}[\varnothing, [\text{find} = \varsigma(x, (\mathit{date}, \mathit{limit})).\text{hotel.room}(\mathit{date}), \ldots]] \\
& \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow_{\parallel}^{*} \quad & (\text{REQUEST, LOCAL}) \\
& \text{customer}[f_0 \mapsto f_1, \varnothing] \\
& \parallel \text{broker}[f_1 \mapsto \text{hotel.room}(\mathit{date}), \ldots] \\
& \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots]
\end{aligned}
$$

Example: service broker

Client reserves a hotel using a *broker*

$$
\begin{array}{ll}
& \text{customer}[f_0 \mapsto \text{broker.find(date,limit)}, \varnothing] \\
& \parallel \text{broker}[\varnothing, [\text{find} = \varsigma(x, (\mathit{date}, \mathit{limit})).\text{hotel.room}(\mathit{date}), \ldots]] \\
& \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow_{\parallel}^{*} & (\text{REQUEST, LOCAL}) \\
& \text{customer}[f_0 \mapsto f_1, \varnothing] \\
& \parallel \text{broker}[f_1 \mapsto \text{hotel.room}(\mathit{date}), \ldots] \\
& \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow_{\parallel}^{*} & (\text{REQUEST, LOCAL}) \\
& \text{customer}[f_0 \mapsto f_1, \varnothing] \\
& \parallel \text{broker}[f_1 \mapsto f_2, \ldots] \\
& \parallel \text{hotel}[f_2 \mapsto \text{bookingref}, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots]
\end{array}
$$

Example: service broker

Client reserves a hotel using a *broker*

$$
\begin{array}{ll}
 & \text{customer}[f_0 \mapsto \text{broker.find(date,limit)}, \varnothing] \\
 & \parallel \text{broker}[\varnothing, [\text{find} = \varsigma(x, (\mathit{date}, \mathit{limit})).\text{hotel.room}(\mathit{date}), \ldots]] \\
 & \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow^{*}_{\parallel} & (\text{REQUEST, LOCAL}) \\
 & \text{customer}[f_0 \mapsto f_1, \varnothing] \\
 & \parallel \text{broker}[f_1 \mapsto \text{hotel.room}(\mathit{date}), \ldots] \\
 & \parallel \text{hotel}[\varnothing, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow^{*}_{\parallel} & (\text{REQUEST, LOCAL}) \\
 & \text{customer}[f_0 \mapsto f_1, \varnothing] \\
 & \parallel \text{broker}[f_1 \mapsto f_2, \ldots] \\
 & \parallel \text{hotel}[f_2 \mapsto \text{bookingref}, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots] \\
\rightarrow^{*}_{\parallel} & (\text{REPLY}) \\
 & \text{customer}[f_0 \mapsto \text{bookingref}, \varnothing] \\
 & \parallel \text{broker}[f_1 \mapsto f_2, \ldots] \\
 & \parallel \text{hotel}[f_2 \mapsto \text{bookingref}, [\text{room} = \varsigma(x, \mathit{date})\text{bookingref}, \ldots]
\end{array}
$$

Observations

- Service broker has a private domain of hotel addresses.
- He searches, negotiates with hotel, and gives only the future f_2 to client.
- Client receives bookingref using f_2 without viewing details of the hotel nor others from broker's domain.

Overview

1. ASP_{fun}
2. ASP_{fun} in Isabelle/HOL
3. Example for ASP_{fun}
4. Results, Discussion, Outlook

Next goal: security, noninterference

- Noninterference: formal definition of security (confidentality)
- Security types for static security analysis, [3]
    - *Type safety* ⇒ *security*

[3] F. Kammüller. Formalizing Non-Interference for Bytecode-Languages in Coq. Formal Aspects of Computing: **20**(3):259–275. Springer, 2008.

Next goal: security, noninterference

- Noninterference: formal definition of security (confidentality)
- Security types for static security analysis, [3]
  - *Type safety $\Rightarrow$ security*
- Challenge: *Literature [2] shows that for parallel programs noninterference more difficult because a process can observe termination of others.*

[2] G. Boudol, I. Castellani. Noninterference for Concurrent Programs. ICALP'01. LNCS:**2076**, Springer, 2001.

[3] F. Kammüller. Formalizing Non-Interference for Bytecode-Languages in Coq. Formal Aspects of Computing: **20**(3):259–275. Springer, 2008.

Next goal: security, noninterference

- Noninterference: formal definition of security (confidentality)
- Security types for static security analysis, [3]
  - *Type safety ⇒ security*
- Challenge: *Literature [2] shows that for parallel programs noninterference more difficult because a process can observe termination of others.*

⇒ ASP_{fun} separate date spaces in active objects; stronger noninterference property expected

[2] G. Boudol, I. Castellani. Noninterference for Concurrent Programs. ICALP'01. LNCS:**2076**, Springer, 2001.

[3] F. Kammüller. Formalizing Non-Interference for Bytecode-Languages in Coq. Formal Aspects of Computing: **20**(3):259–275. Springer, 2008.

Discussion and outlook

- $\text{ASPEN}_{\text{DFG}}$: Security analysis of distributed active objects
- Development of a new language ASP_{fun}
  - Modelling language concepts
  - Proof of meta-theorems: well-formedness, no cycles
  - Type system and proof of type safety
    - ⇒ deadlock freedom
    - ⇒ security
- Development in Isabelle/HOL
  - 100 % consistency (correctness)
  - Generation of prototypical tools (interpreter and type checker)
- Outlook: components with futures in Isabelle/HOL
- Synergies with formalisation of aspect-oriented programming ($\text{ASCOT}_{\text{DFG}}$)

Current papers (selection)

[1] L. Henrio, F. Kammüller. A Mechanized Model of the Theory of Objects. 9th IFIP Int. Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS'07. LNCS **4468**, Springer 2007.

[2] L. Henrio and F. Kammüller. Functional Active Objects: Typing and Formalisation. Foundations of Coordination Languages and System Architectures, FOCLASA'09. Satellite to ICALP'09. To appear in ENTCS, 2009.

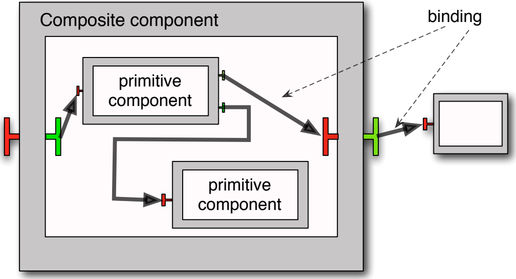
[3] F. Kammüller. Formalizing Non-Interference for A Small Bytecode-Language in Coq. Formal Aspects of Computing: **20**(3):259–275. Springer, 2008.

[4] F. Kammüller, H. Sudhof. Composing safely – A Type System for Aspects. Software Composition, Satellite to ETAPS'08. LNCS **4954**:231–247, Springer 2008.

[5] F. Kammüller, H. Sudhof. Compositionality of Aspect Weaving. Autonomous Systems – Self-Organisation, Management, and Control. B. Mahr, Z. Sheng (Eds.), Springer, 2008.

[6] F. Kammüller, R. Kammüller. Enhancing Privacy Implementations of Database Enquiries. The Fourth International Conference on Internet Monitoring and Protection. IEEE Computer Press, to appear 2009.

Further goals: components



- Too much detail at object level
- Formalising components
  - Abstraction of data and algorithms
  - Model only structure of communication, i.e futures
  - Primitive und composite components
  - Specification of behaviour for primitive components
  - Composition of behaviour for composites

ASP_{fun} : Premier Exemple

- Création d'une Activité paràllele, qui incrément 5 par 1

ASP_{fun} : Premier Exemple

- Création d'une Activité paràllele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul

ASP_{fun} : Premier Exemple

- Création d'une Activité paràllele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul
- Configuration initial contient objet vide $\varnothing$ et le program comme suivant
  $\alpha([f_0 \mapsto Active([m = \varsigma(x)x.z + 1, z = 5]).m], \varnothing)$

ASP_{fun} : Premier Exemple

- Création d'une Activité parállele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul
- Configuration initial contient objet vide $\emptyset$ et le program comme suivant
  $\alpha([f_0 \mapsto Active([m = \varsigma(x)x.z + 1, z = 5]).m], \emptyset)$
- *Active* crée activité β ...
  $\beta([], [m = \varsigma(x)x.z + 1, z = 5])$ ... remplace à la location appellante la reference β de l'Activité
  $\alpha([f_0 \mapsto \beta.m], \emptyset)$

ASP$_{\text{fun}}$: Premier Exemple

- Création d'une Activité parállele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul
- Configuration initial contient objet vide $\emptyset$ et le program comme suivant
  $\alpha([f_0 \mapsto Active([m = \varsigma(x)x.z + 1, z = 5]).m], \emptyset)$
- *Active* crée activité β ...
  $\beta([], [m = \varsigma(x)x.z + 1, z = 5])$ ... remplace à la location appellante la reference β de l'Activité
  $\alpha([f_0 \mapsto \beta.m], \emptyset)$
- Request restant $\beta.\mathtt{m}$ crée Future, en résumé :
  $\alpha([f_0 \mapsto f_1], \emptyset) \| \beta([f_1 \mapsto [m = \varsigma(x)x.z + 1, z = 5].m], \ldots)$

ASP_{fun} : Premier Exemple

- Création d'une Activité parállele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul
- Configuration initial contient objet vide $\emptyset$ et le program comme suivant
  $\alpha([f_0 \mapsto Active([m = \varsigma(x)x.z + 1, z = 5]).m], \emptyset)$
- *Active* crée activité β ...
  $\beta([], [m = \varsigma(x)x.z + 1, z = 5])$ ... remplace à la location appellante la reference β de l'Activité
  $\alpha([f_0 \mapsto \beta.m], \emptyset)$
- Request restant $\beta.\mathtt{m}$ crée Future, en résumé :
  $\alpha([f_0 \mapsto f_1], \emptyset) \| \beta([f_1 \mapsto [m = \varsigma(x)x.z + 1, z = 5].m], \ldots)$
- Evaluation suivant règle LOCAL donne
  $\alpha([f_0 \mapsto f_1], \emptyset) \| \beta([f_1 \mapsto 6, [m = \varsigma(x)x.z + 1, z = 5])$

ASP_{fun} : Premier Exemple

- Création d'une Activité parallele, qui incrément 5 par 1
- Supposition : Nombres naturels en ς-calcul
- Configuration initial contient objet vide $\emptyset$ et le program comme suivant
  $\alpha([f_0 \mapsto Active([m = \varsigma(x)x.z + 1, z = 5]).m], \emptyset)$
- *Active* crée activité β ...
  $\beta([], [m = \varsigma(x)x.z + 1, z = 5])$ ... remplace à la location appellante la reference β de l'Activité
  $\alpha([f_0 \mapsto \beta.m], \emptyset)$
- Request restant $\beta.\mathtt{m}$ crée Future, en résumé :
  $\alpha([f_0 \mapsto f_1], \emptyset) \| \beta([f_1 \mapsto [m = \varsigma(x)x.z + 1, z = 5].m], \ldots)$
- Evaluation suivant règle LOCAL donne
  $\alpha([f_0 \mapsto f_1], \emptyset) \| \beta([f_1 \mapsto 6, [m = \varsigma(x)x.z + 1, z = 5])$
- Par REPLY le résultat est rendu.
  $\alpha([f_0 \mapsto 6], \emptyset) \| \beta([f_1 \mapsto 6, [m = \varsigma(x)x.z + 1, z = 5])$

ASP$_{\text{fun}}$-Semantik

LOCAL

$$\frac{s \to_\varsigma s'}{\alpha[f_i \mapsto s :: Q, t] :: C \to_\parallel \alpha[f_i \mapsto s' :: Q, t] :: C}$$

ACTIVE

$$\frac{\gamma \notin (\mathrm{dom}(C) \cup \{\alpha\}) \qquad noFV(s)}{\alpha[f_i \mapsto E[Active(s)] :: Q, t] :: C \to_\parallel \alpha[f_i \mapsto E[\gamma] :: Q, t] :: \gamma[\varnothing, s] :: C}$$

REQUEST

$$\frac{f_k \text{ fresh} \qquad noFV(s)}{\alpha\,[f_i \mapsto E[\beta.l(s)] :: Q, t] :: \beta[R, t'] :: C \to_\parallel \alpha\,[f_i \mapsto E[f_k] :: Q, t] :: \beta\,[f_k \mapsto t'.l(s) :: R, t'] :: C}$$

SELF-REQUEST

$$\frac{f_k \text{ fresh} \qquad noFV(s)}{\alpha\,[f_i \mapsto E[\alpha.l(s)] :: Q, t] :: C \to_\parallel \alpha\,[f_k \mapsto t.l(s) :: f_i \mapsto E[f_k] :: Q, t] :: C}$$

REPLY

$$\frac{\beta[f_k \mapsto s :: R, t'] \in \alpha[f_i \mapsto E[f_k] :: Q, t] :: C}{\alpha[f_i \mapsto E[f_k] :: Q, t] :: C \to_\parallel \alpha[f_i \mapsto E[s] :: Q, t] :: C}$$

UPDATE-AO

$$\frac{\begin{array}{c}\gamma \notin (\mathrm{dom}(C) \cup \{\alpha\}) \\ noFV(\varsigma(x,y)s) \qquad \beta[Q, t'] \in (\alpha[f_i \mapsto E[\beta.l := \varsigma(x,y)s] :: Q, t] :: C)\end{array}}{\alpha[f_i \mapsto E[\beta.l := \varsigma(x,y)s] :: Q, t] :: C \to_\parallel \alpha[f_i \mapsto E[\gamma] :: Q, t] :: \gamma[\varnothing, t'.l := \varsigma(x,y)s] :: C}$$

Table 1. ASP$_{\text{fun}}$ semantics

ASP_{fun}-Typsystem Lokal

Val x

$$x{:}A :: T \vdash x : A$$

Val not x

$$\frac{y \neq x \qquad T \vdash x : B}{y{:}A :: T \vdash x : B}$$

Object Formation

$$\frac{\forall i \in 1..n, \vdash B_i \;\wedge\; \vdash D_i \qquad \forall i,j \in 1..n, i \neq j \Rightarrow l_i \neq l_j}{\vdash [l_i : B_i {\rightarrow} D_i]^{i \in 1..n}}$$

Type Object

$$\frac{A = [l_i : B_i {\rightarrow} D_i]^{i \in 1..n} \qquad \forall i \in 1..n,\ x_i{:}A :: y_i{:}B_i :: T \vdash b_i : D_i}{T \vdash [l_i = \varsigma(x_i : A, y_i : B_i)b_i]^{i \in 1..n} : A}$$

Type Call

$$\frac{T \vdash a : [l_i : B_i {\rightarrow} D_i]^{i \in 1..n} \qquad j \in 1..n \qquad T \vdash b : B_j}{T \vdash a.l_j(b) : D_j}$$

Type Update

$$\frac{A = [l_i : B_i {\rightarrow} D_i]^{i \in 1..n} \qquad T \vdash a : A \qquad j \in 1..n \qquad x{:}A :: y{:}B :: T \vdash b : D_j}{T \vdash a.l_j := \varsigma(x : A, y : B)b : A}$$

Table 2. Typing the local calculus

ASP$_{fun}$-Typsystem Global

Type Active

$$\frac{\langle \Gamma_{act}, \Gamma_{fut}\rangle, T \vdash a : A}{\langle \Gamma_{act}, \Gamma_{fut}\rangle, T \vdash Active(a) : A}$$

Type Activity Reference

$$\frac{\beta \in \mathrm{dom}(\Gamma_{act})}{\langle \Gamma_{act}, \Gamma_{fut}\rangle, T \vdash \beta : \Gamma_{act}(\beta)}$$

Type Future Reference

$$\frac{f_k \in \mathrm{dom}(\Gamma_{fut})}{\langle \Gamma_{act}, \Gamma_{fut}\rangle, T \vdash f_k : \Gamma_{fut}(f_k)}$$

Type Configuration

$$\frac{\begin{array}{l} \mathrm{dom}(\Gamma_{act}) = \mathrm{dom}(C) \qquad \mathrm{dom}(\Gamma_{fut}) = \bigcup\{\mathrm{dom}(Q) \mid \exists\, \alpha, a.\ \alpha[Q,a] \in C\} \\ \forall\, \alpha, Q, a, C'.\ C = \alpha[Q,a] :: C' \Rightarrow \begin{cases} \langle \Gamma_{act}, \Gamma_{fut}\rangle, \varnothing \vdash a : \Gamma_{act}(\alpha)\ \wedge \\ \forall\, f_i \in \mathrm{dom}(Q).\ \langle \Gamma_{act}, \Gamma_{fut}\rangle, \varnothing \vdash Q(f_i) : \Gamma_{fut}(f_i) \end{cases} \end{array}}{\vdash C : \langle \Gamma_{act}, \Gamma_{fut}\rangle}$$

Typsysteme und Typkorrektheit

- Typsystem gestattet statische Überprüfung gewisser semantischer Eigenschaften
- Typsicherheit informell: Ein wohlgetyptes Programm `t` verhält sich *vernünftig*
  1. Auswertung von `t` respektiert die Typen (*Preservation/Subject Reduction*)

     $\llbracket \mathtt{E} \vdash \mathtt{t} : \mathtt{T};\ \mathtt{t} \longrightarrow_{\varsigma}^{*} \mathtt{t'} \rrbracket \Longrightarrow \mathtt{E} \vdash \mathtt{t'} : \mathtt{T}$
  2. Programm `t` bleibt nicht stecken (*Progress*)

     $\llbracket \mathtt{E} \vdash \mathtt{t} : \mathtt{T};\ \neg\ \mathtt{value}\ (\mathtt{t}) \rrbracket \Longrightarrow \exists\ \mathtt{t'},\ \mathtt{t} \longrightarrow_{\varsigma} \mathtt{t'}$
  - $\mathtt{E} \vdash \mathtt{t} : \mathtt{T}$ heisst *term* `t` *hat Typ* `T` *in Typumgebung* `E`

Bindungs-Techniken in Isabelle

- Wir benutzen DeBruijn-Indizes
- Sehr gewöhnungsbedürftig aber praktisch und einfach
- Nominal Techniques leider nicht einsatzbar
- Locally Nameless:
  - noch experimentell
  - Vorteile noch unklar
  - Vermutung: insbesondere bei Konfigurationen weniger Problem mit “fresh”

Vom ς-Kalkül zu ASP$_{\text{fun}}$

- Isabelle datatype für ς-Terme

```
datatype term = Var nat
    | Obj label ⇀f term
    | Call term label
    | Upd term label term
```

Vom ς-Kalkül zu ASP$_{\text{fun}}$

- Isabelle datatype für ς-Terme
- Erweiterung um `Active-`, `ActRef-` und `FutRef-`Terme für Activation, Activity- und Futurereferenzen

```
datatype term = Var nat
    | Obj label ⇀f term
    | Call term label
    | Upd term label term
    | Active term
    | ActRef ActivityRef
    | FutRef FutureRef
```

O: Theory of Objects, ς-calculus

- Terms in the ς-calculus

$$
\begin{array}{rll}
a, b ::= & [l_j = \varsigma(x_j)a_j]^{j\in 1..n} & \text{object definition} \\
& \mid a.l_j & (j \in 1..n) \text{ method call} \\
& \mid a.l_j := \varsigma(x)b & (j \in 1..n) \text{ update}
\end{array}
$$

- Semantics/Reduction for $o \equiv [l_j = \varsigma(x_j)a_j]^{j\in 1..n}$ (l_i distinct).

$$
\begin{array}{ll}
o & \text{object with method names } l_i \\
& \text{methods } \varsigma(x_i)b_i \\
o.l_j(b) \rightarrow_\varsigma b_j\{x_j \leftarrow o\} & (j \in 1..n) \text{ selection / method call} \\
o.l_j := \varsigma(x)b \rightarrow_\varsigma [l_j = \varsigma(y)b, & l_i = \varsigma(x_i)b_i^{i\in(1..n)-\{j\}}] \\
& (j \in 1..n) \text{ update/override}
\end{array}
$$

Die Theorie der Objekte: ς-Kalkül

Definition (Sigma-Term)

Ein Sigma-Term ist ein Wort der folgenden Grammatik:

$a, b ::=$		Terme
	x	Variable
	$[l_i = \varsigma(x_i)b_i]^{i \in 1..n}$	Objekt
	$a.l$	Feld-Auswahl / Methodenaufruf
	$a.l \Leftarrow \varsigma(x)b$	Feld-Update / Methoden-Update

Die Theorie der Objekte: ς-Kalkül

Definition (Sigma-Term)

Ein Sigma-Term ist ein Wort der folgenden Grammatik:

$a, b ::=$		Terme
	x	Variable
	$[l_i = \varsigma(x_i)b_i]^{i \in 1..n}$	Objekt
	$a.l$	Feld-Auswahl / Methodenaufruf
	$a.l \Leftarrow \varsigma(x)b$	Feld-Update / Methoden-Update

- Semantik: Reduktionsrelation $\rightarrow_\varsigma$
- Substitution des formalen Parameters mit a it”self”

$$a \equiv [l_j = \varsigma(x_j)b_j]^{j \in 1..n}$$

$$a.l_j \rightarrow_\varsigma b_j[a/x_j] \qquad j \in 1..n$$

Beispiel ς-Kalkül

```
zero = [ iszero = true;
         pred   = ς(x)x,
         succ   = ς(x)(x.iszero := false).pred := x]
```

Beispiel ς-Kalkül

```
zero = [ iszero = true;
         pred   = ς(x)x,
         succ   = ς(x)(x.iszero := false).pred := x]

one =  zero.succ
```

Beispiel ς-Kalkül

```
zero = [ iszero = true;
         pred   = ς(x)x,
         succ   = ς(x)(x.iszero := false).pred := x]

one =  zero.succ

   →ς (zero.iszero := false).pred := zero
```

Beispiel ς-Kalkül

```
zero = [ iszero = true;
         pred   = ς(x)x,
         succ   = ς(x)(x.iszero := false).pred := x]

one =  zero.succ

   →ς (zero.iszero := false).pred := zero

   →ς[iszero = false,
        pred   = ς(x)x,
        succ   = ς(x)(x.iszero := false).pred := x].pred := zero
```

Beispiel ς-Kalkül

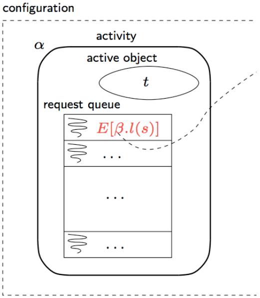
```
zero = [ iszero = true;
         pred   = ς(x)x,
         succ   = ς(x)(x.iszero := false).pred := x]

one =  zero.succ

   →ς (zero.iszero := false).pred := zero

   →ς[iszero = false,
       pred   = ς(x)x,
       succ   = ς(x)(x.iszero := false).pred := x].pred := zero

   →ς[iszero = false,
       pred   = zero,
       succ   = ς(x)(x.iszero := false).pred := x]
```

T: Typing rules for Sigma only

```
inductive typing
intros
  Var: ⟦x < length E; (E!x) = T ⟧⟹ E ⊢ Var x : T
  Obj: ⟦length b = len B;
        ∀ i < len B. E ⟨0:B ⟩⊢ (b!i): (B!i)⟧
        ⟹ E ⊢ (Obj b) : B
  Call: ⟦E ⊢ a: A; l < len A ⟧⟹ E ⊢ (Call a l): (A!l)
  Upd: ⟦E ⊢ a: A; l < len A; E ⟨0:A ⟩⊢ n: (A!l) ⟧
        ⟹ E ⊢ (Upd a l n) : A
```

From ς-calculus to ASP_{fun}

- ASP_{fun} builds on ς-Kalkül of Abadi and Cardelli
- Activities $\alpha[R, t]$:
  - Lists of futures R (request queue)
  - ς-Objekt t (immutable)



- Syntactic extension of ς-calculus by:
  - *Active*: creation of a new active object
  - *FutRef* and *ActRef*: references for activities and futures (transparent)
- Semantics: *local* and *parallel* evaluation

Illustration: rule in semantic notation and in Isabelle/HOL

REPLY

$$\frac{\beta[f_k \mapsto s :: R, t'] \in \alpha[f_i \mapsto E[f_k] :: Q, t] :: C}{\alpha[f_i \mapsto E[f_k] :: Q, t] :: C \rightarrow_{\parallel} \alpha[f_i \mapsto E[s] :: Q, t] :: C}$$

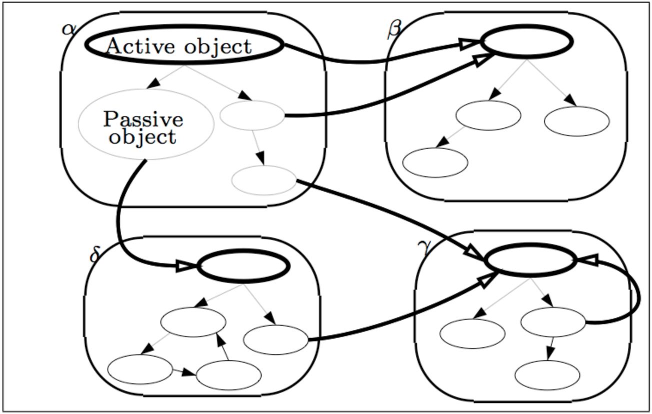
Illustration: rule in semantic notation and in Isabelle/HOL

REPLY

$$\frac{\beta[f_k \mapsto s::R, t'] \in \alpha[f_i \mapsto E[f_k]::Q, t] :: C}{\alpha[f_i \mapsto E[f_k]::Q, t] :: C \rightarrow_{\|} \alpha[f_i \mapsto E[s]::Q, t] :: C}$$

```
reply:
⟦ C α = Some (Ra, t); Ra(fi) = Some(E ↑(FutRef(fk)));
  C β = Some(Rb, t'); Rb(fk) = Some(s) ⟧
 ⟹ C →‖ C (α↦(Ra (fi ↦(E↑s)), t)
```

ASP: Topology of Activities



Characteristics of ASP

- Object-oriented language
- Asynchronous communication
- Parallel processes (active objects)
- Futures
  - Asynchronous method calls to active objects
  - Results of such calls are represented by futures until corresponding response is returned
- Synchronization through wait-by-necessity: wait occurs when a strict operation on a future is performed