Programming language semantics in Dedukti

1st EuroProofNet Dedukti School, Nantes, 25 June 2022

Most systems that are translated into Dedukti are logical systems.

Here we will deviate a little from this type of systems and address the encoding in Dedukti of programming languages features.

Part 1. Objects, objects types and structural subtyping in Dedukti (Catherine Dubois)

Part 2. Semantics definitions as they are provided by the $\mathbb{K}$ framework in Dedukti (Amélie Ledein)

Part 1. Objects and Subtyping in Dedukti

Catherine Dubois

ENSIIE, Samovar, Évry, France

Joint work with Raphaël Cauderlier

The simply-typed ς-calculus (Abadi, Cardelli 1996)

Types	$A, B, \ldots$	$::=$	$[l_i : A_i]_{i \in 1 \ldots n}$	
Terms	$a, b, \ldots$	$::=$	x	*variable*
		$\mid$	$[l_i = \varsigma(x_i : A) a_i]_{i \in 1 \ldots n}$	*object*
		$\mid$	$a.l$	*method selection*
		$\mid$	$a.l \Leftarrow \varsigma(x : A) b$	*method update*

Operational semantics

$t.l_j$	$\hookrightarrow$	$t_j\{x := t\}$	*method selection*
$t.l_j \Leftarrow \varsigma(x : A) u$	$\hookrightarrow$	$t\ \{l_j := \varsigma(x : A) u\}$	*method update*

The simply-typed ς-calculus (Abadi, Cardelli 1996)

Typing rules

Here A abbr. for $[l_i : A_i]_{i=1\ldots n}$

$$\frac{}{\Delta \vdash x : \Delta(x)} \qquad \frac{\Delta, x : A \vdash t_i : A_i \quad \forall i = 1 \ldots n}{\Delta \vdash [l_i = \varsigma(x : A)t_i]_{i=1\ldots n} : [l_i : A_i]_{i=1\ldots n}}$$

$$\frac{\Delta \vdash a : A \qquad (l_j : A_j) \in A}{\Delta \vdash a.l_j : A_j} \qquad \frac{\Delta \vdash a : A \qquad \Delta, x : A \vdash b : A_i}{\Delta \vdash a.l_i \Leftarrow \varsigma(x : A)b : A}$$

Subtyping is defined by

$$[l_i : A_i]_{i=1\ldots n+m} <: [l_i : A_i]_{i=1\ldots n}$$

A is a subtype of B iff A and B coincide on the labels of B

$+$ reflexivity and transitivity of $<:$ and subsumption

A shallow embedding in Dedukti

Shallow here means preservation of variable binding, typing, and operational semantics.

Roadmap:

1. study of the calculus without subtyping
2. subtyping

Translation of ς-types

```
label : Type.
type : Type.
def Obj : type -> Type.
```

No records in Dedukti $\longrightarrow$ lists (of pairs of labels and types) in Dedukti

In lists, order of labels matters

Our solution: rely on the translator to always print the labels in the same order (e.g. alphabetic order);

```
tnil: type.
tcons: label -> type -> type -> type.
```

Translation of ς-types

Other solutions:

1. use dependent types with logical arguments enforcing that the lists are sorted and duplicate-free by construction;
   $\longrightarrow$ doable but complex!

```
lt : label -> label -> Type.
def minors : label -> type -> Type.
nil : type .
cons : l : label -> type -> B : type -> minors l B -> type.
minors_nil : l : label -> minors l nil.
minors_cons : l1 : label -> l2  : label -> A : type -> B : type
  -> H : minors l2 B -> lt l1 l2 -> minors l1 (cons l2 A B H ).
```

2. use rewriting to make lists given in different order convertible.
   $\longrightarrow$ declare the list concatenation as an associative commutative operation or use rewrite rules to sort lists

Notion of position

Position of a pair (l, A) in a type: at head or deeper

```
pos : label -> type -> type -> Type.

at_head: l : label -> A : type -> B : type -> pos l A (tcons l A B).
in_tail : l : label -> A : type -> m : label -> C : type -> B : type
    ->  pos l A B -> pos l A (tcons m C B).
```

(Terminating) Translation of ς-terms

Objective: define a context Σ_ς and a translation function $[\![\_]\!]$ mapping ς-terms to Dedukti terms preserving typing.

`Meth A B`: type of methods of objects of type A returning type B
`def Meth : type -> type -> Type .`

An object = a record of methods
$\longrightarrow$ in Dedukti, object = association list of pairs of labels and methods.

(Terminating) Translation of ς-terms

Objective: define a context Σ_ς and a translation function $[\![\_]\!]$ mapping ς-terms to Dedukti terms preserving typing.

`Meth A B`: type of methods of objects of type A returning type B

```
def Meth : type -> type -> Type .
```

An object = a record of methods
$\longrightarrow$ in Dedukti, object = association list of pairs of labels and methods.

Create an object step by step, adding a method one at a time
$\longrightarrow$ preobjects (because a sublist of an object is not an object)
A preobject of an object of type A = a list of methods implementing part of A (type B) : `Preobj A B`

```
Preobj : type -> type -> Type.
pnil : A : type -> Preobj A tnil .
pcons : A : type -> l : label -> B : type -> C : type ->
   Meth A B -> Preobj A C -> Preobj A ( tcons l B C ).
```

(Terminating) Translation of ς-terms

Objective: define a context Σ_ς and a translation function $[\![\_]\!]$ mapping ς-terms to Dedukti terms preserving typing.

`Meth A B`: type of methods of objects of type A returning type B

```
def Meth : type -> type -> Type .
```

An object = a record of methods
$\longrightarrow$ in Dedukti, object = association list of pairs of labels and methods.

Create an object step by step, adding a method one at a time
$\longrightarrow$ preobjects (because a sublist of an object is not an object)
A preobject of an object of type A = a list of methods implementing part of A (type B) : `Preobj A B`

```
Preobj : type -> type -> Type.
pnil : A : type -> Preobj A tnil .
pcons : A : type -> l : label -> B : type -> C : type ->
   Meth A B -> Preobj A C -> Preobj A ( tcons l B C ).
```

An object of type A = a preobject of type A defined on A

```
[A] Obj A --> Preobj A A .
```

(Terminating) Translation of ς-terms

```
[A, B] Meth A B --> (Obj A -> Obj B).
```

$\longrightarrow$ simplest way but not possible

$\longrightarrow$ equivalence between `Meth A B` and `Obj A -> Obj B`

```
def Eval_meth : A : type -> B : type -> Meth A B
                -> Obj A -> Obj B.
def Make_meth : A : type -> B : type -> (Obj A -> Obj B)
                -> Meth A B.
```

(Terminating) Translation of ς-terms

First selection and update on preobjects

```
def preselect : A : type -> l : label -> B : type ->
  C : type -> pos l B C -> Preobj A C -> Meth A B.
def preupdate : A : type -> l : label -> B : type ->
  C : type -> pos l B C -> Preobj A C -> Meth A B -> Preobj A C.

[m] preselect _ _ _ _ (at_head _ _ _ ) (pcons _ _ _ _ m _) --> m.
...
[A, B, C, l, o, m]
preupdate _ _ _ _ (at_head _ _ _  ) (pcons A l B C _ o ) m -->
   pcons A l B C m o.
...
```

(Terminating) Translation of ς-terms

First selection and update on preobjects

```
def preselect : A : type -> l : label -> B : type ->
  C : type -> pos l B C -> Preobj A C -> Meth A B.
def preupdate : A : type -> l : label -> B : type ->
  C : type -> pos l B C -> Preobj A C -> Meth A B -> Preobj A C.

[m] preselect _ _ _ _ (at_head _ _ _ ) (pcons _ _ _ _ m _) --> m.
...
[A, B, C, l, o, m]
preupdate _ _ _ _ (at_head _ _ _  ) (pcons A l B C _ o ) m -->
   pcons A l B C m o.
...
```

Now selection and update on objects

```
def objselect : A : type -> l : label -> B : type -> pos l B A ->
   Obj A -> Obj B .
[A,l,B,p,a]  objselect A l B p a -->
   Eval_meth A B ( preselect A l B A p a ) a .
def objupdate : A : type -> l : label -> B : type -> pos l B A ->
   Obj A -> Meth A B -> Obj A .
[A,l,B,p,a,m] objupdate A l B p a m --> preupdate A l B A p a m .
```

(Terminating) Translation of ς-terms

Σ_ς= all the declarations and rewrite rules introduced so far

Theorem

The underlying rewrite system is strongly normalizing and congruent.

(Terminating) Translation of ς-terms

Translation function from ς-terms to Dedukti terms:

$$
\begin{array}{lcl}
[\![x]\!] & := & x \\
[\![[l_i = \varsigma(x_i : A)a_i]_{l_1<\ldots<l_n}]\!] & := & \texttt{pcons}\ [\![A]\!]\ l_1\ [\![A_1]\!]\ [\![[l_i : A_i]_{1<i\leq n}]\!]\ [\![\varsigma(x_1 : A)a_1]\!](\ldots \\
 & & \quad (\texttt{pcons}\ [\![A]\!]\ l_n\ [\![A_n]\!]\ \texttt{tnil}\ [\![\varsigma(x_n : A)a_n]\!]\ (\texttt{pnil}\ [\![A]\!]))\ldots) \\
 & & \textit{when } A \textit{ is } [l_i : A_i]_{l_1<\ldots<l_n} \\
[\![a.l]\!] & := & \texttt{objselect}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![a]\!] \\
 & & \textit{when } a : A,\ a.l : B \textit{ and } p \textit{ is the position of } (l : B) \textit{ in } A \\
[\![a.l \Leftarrow \varsigma(x : A)b]\!] & := & \texttt{objupdate}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![\varsigma(x : A)b]\!]\ [\![a]\!] \\
 & & \textit{when } b : B \textit{ and } p \textit{ is the position of } (l, B) \textit{ in } A \\
[\![\varsigma(x : A)b]\!] & := & \texttt{Make\_meth}\ [\![A]\!]\ [\![B]\!]\ (x : \texttt{Obj}\ [\![A]\!]\ \texttt{=>}\ [\![b]\!]) \\
 & & \textit{when } b : B
\end{array}
$$

(Terminating) Translation of ς-terms

Translation function from ς-terms to Dedukti terms:

$$
\begin{array}{lcl}
[\![x]\!] & := & x \\
[\![[l_i = \varsigma(x_i : A)a_i]_{l_1<\ldots<l_n}]\!] & := & \texttt{pcons}\ [\![A]\!]\ l_1\ [\![A_1]\!]\ [\![[l_i : A_i]_{1<i\leq n}]\!]\ [\![\varsigma(x_1 : A)a_1]\!](\ldots \\
 & & \quad (\texttt{pcons}\ [\![A]\!]\ l_n\ [\![A_n]\!]\ \texttt{tnil}\ [\![\varsigma(x_n : A)a_n]\!]\ (\texttt{pnil}\ [\![A]\!]))\ldots) \\
 & & \textit{when } A \textit{ is } [l_i : A_i]_{l_1<\ldots<l_n} \\
[\![a.l]\!] & := & \texttt{objselect}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![a]\!] \\
 & & \textit{when } a : A,\ a.l : B \textit{ and } p \textit{ is the position of } (l : B) \textit{ in } A \\
[\![a.l \Leftarrow \varsigma(x : A)b]\!] & := & \texttt{objupdate}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![\varsigma(x : A)b]\!]\ [\![a]\!] \\
 & & \textit{when } b : B \textit{ and } p \textit{ is the position of } (l, B) \textit{ in } A \\
[\![\varsigma(x : A)b]\!] & := & \texttt{Make\_meth}\ [\![A]\!]\ [\![B]\!]\ (x : \texttt{Obj}\ [\![A]\!]\ \texttt{=>}\ [\![b]\!]) \\
 & & \textit{when } b : B
\end{array}
$$

Theorem *Typing preservation for simply-typed ς-calculus.*
The translation of a typing derivation $\Delta \vdash a : A$ is a well-typed Dedukti term $[\![a]\!]$ of type $\texttt{Obj}\ [\![A]\!]$ in any context extending $[\![\Delta]\!]$ by the declarations of the labels occurring in $[\![A]\!]$.

(Terminating) Translation of ς-terms

Translation function from ς-terms to Dedukti terms:

$$
\begin{array}{lcl}
[\![x]\!] & := & x \\
[\![[l_i = \varsigma(x_i : A)a_i]_{l_1<\ldots<l_n}]\!] & := & \texttt{pcons}\ [\![A]\!]\ l_1\ [\![A_1]\!]\ [\![[l_i : A_i]_{1<i\leq n}]\!]\ [\![\varsigma(x_1 : A)a_1]\!](\ldots \\
 & & \quad (\texttt{pcons}\ [\![A]\!]\ l_n\ [\![A_n]\!]\ \texttt{tnil}\ [\![\varsigma(x_n : A)a_n]\!]\ (\texttt{pnil}\ [\![A]\!]))\ldots) \\
 & & \textit{when } A \textit{ is } [l_i : A_i]_{l_1<\ldots<l_n} \\
[\![a.l]\!] & := & \texttt{objselect}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![a]\!] \\
 & & \textit{when } a : A,\ a.l : B \textit{ and } p \textit{ is the position of } (l : B) \textit{ in } A \\
[\![a.l \Leftarrow \varsigma(x : A)b]\!] & := & \texttt{objupdate}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![\varsigma(x : A)b]\!]\ [\![a]\!] \\
 & & \textit{when } b : B \textit{ and } p \textit{ is the position of } (l, B) \textit{ in } A \\
[\![\varsigma(x : A)b]\!] & := & \texttt{Make\_meth}\ [\![A]\!]\ [\![B]\!]\ (x : \texttt{Obj}\ [\![A]\!]\ \texttt{=>}\ [\![b]\!]) \\
 & & \textit{when } b : B
\end{array}
$$

Theorem *Typing preservation for simply-typed ς-calculus.*
The translation of a typing derivation $\Delta \vdash a : A$ is a well-typed Dedukti term $[\![a]\!]$ of type $\texttt{Obj}\ [\![A]\!]$ in any context extending $[\![\Delta]\!]$ by the declarations of the labels occurring in $[\![A]\!]$.

Yes .. but this encoding does not preserve reduction ...

Shallow Embedding

Objective: define a new context Σ'_ς and a translation function $[\![\_]\!]$ mapping ς-terms to Dedukti terms preserving typing and reduction

.. may not terminate as it is the case in the ς-calculus.

$\longrightarrow$ Identify `Meth A B` and `Obj A -> Obj B`

```
[A,B] Meth A B --> ( Obj A -> Obj B ).
[f] Eval_meth _ _ f --> f .
[f] Make_meth _ _ f --> f .
```

Shallow Embedding

Objective: define a new context Σ'_ς and a translation function $[\![\_]\!]$ mapping ς-terms to Dedukti terms preserving typing and reduction

.. may not terminate as it is the case in the ς-calculus.

$\longrightarrow$ Identify `Meth A B` and `Obj A -> Obj B`

```
[A,B] Meth A B --> ( Obj A -> Obj B ).
[f] Eval_meth _ _ f --> f .
[f] Make_meth _ _ f --> f .
```

Theorem *Reduction preservation*.
Let a and a' be two ς-terms of type A such that a $\rightsquigarrow$ a', we have $[\![a]\!] \longrightarrow^+ [\![a']\!]$.

Subtyping

- Translation of $<:$:

```
def subtype ( A : type ) ( B : type ) :=   (;  A <: B ;)
l : label -> C : type -> pos l C B -> pos l C A .
```

Subtyping

- Translation of $<:$:

```
def subtype ( A : type ) ( B : type ) :=   (;  A <: B ;)
l : label -> C : type -> pos l C B -> pos l C A .
```

- In Dedukti, explicit coercions to remedy the lack of subtyping

```
def coerce: A: type -> B: type -> subtype A B -> Obj A -> Obj B .
```

Subtyping

- Translation of $<:$:

```
def subtype ( A : type ) ( B : type ) :=   (;  A <: B ;)
l : label -> C : type -> pos l C B -> pos l C A .
```

- In Dedukti, explicit coercions to remedy the lack of subtyping

```
def coerce: A: type -> B: type -> subtype A B -> Obj A -> Obj B .
```

- Adaptation of selection and update to handle `coerce`

```
def select :
   A: type -> l: label -> B: type -> pos l B A -> Obj A -> Obj B.

[l,B,p,l1,B1,C1,m,o]
select ( tcons _ _ _  ) l B p ( pcons _ l1 B1 C1 m o )
   --> objselect ...

[A,B,C,l,p,st,a] select _ l C p ( coerce A B st a )
  --> select A l C ( st l C p ) a .
```

Subtyping

- Adaptation of $[\![\_]\!]$, translation of typing derivation

$$
\begin{array}{lcl}
[\![a.l]\!] & := & \texttt{select}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![a]\!] \\
 & & \textit{when } a : A,\ a.l : B \textit{ and } p \textit{ is the position of } (l : B) \textit{ in } A \\
[\![a.l \Leftarrow \varsigma(x : A)b]\!] & := & \texttt{update}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![\varsigma(x : A)b]\!]\ [\![a]\!] \\
 & & \textit{when } b : B \textit{ and } p \textit{ is the position of } (l, B) \textit{ in } A \\
[\![a]\!] & := & \texttt{coerce}\ [\![A]\!]\ [\![B]\!]\ st\ [\![a]\!] \\
 & & \textit{when the derivation ends with a subsume rule} \\
 & & \textit{between types } A \textit{ and } B
\end{array}
$$

Subtyping

- Adaptation of $[\![\_]\!]$, translation of typing derivation

$$
\begin{array}{lcl}
[\![a.l]\!] & := & \texttt{select}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![a]\!] \\
 & & \textit{when } a : A,\ a.l : B \textit{ and } p \textit{ is the position of } (l : B) \textit{ in } A \\
[\![a.l \Leftarrow \varsigma(x : A)b]\!] & := & \texttt{update}\ [\![A]\!]\ l\ [\![B]\!]\ p\ [\![\varsigma(x : A)b]\!]\ [\![a]\!] \\
 & & \textit{when } b : B \textit{ and } p \textit{ is the position of } (l, B) \textit{ in } A \\
[\![a]\!] & := & \texttt{coerce}\ [\![A]\!]\ [\![B]\!]\ st\ [\![a]\!] \\
 & & \textit{when the derivation ends with a subsume rule} \\
 & & \textit{between types } A \textit{ and } B
\end{array}
$$

- Theorem *Properties of* (D[ς-calculus], $[\![\_]\!]$)
  (D[ς-calculus), $[\![\_]\!]$) preserves typing and reduction.

  Conjecture (D[ς-calculus), $[\![\_]\!]$) is strongly conservative.
- Implementation: improvements to get more readable Dedukti terms (remove positions and subtyping proofs, concrete objects created from loop methods), type inference for ς-terms

FoCaLiZe and Focalide

- A language/environment for specification, implementation and proof : 1^{st} order logics, pure OCaml like code, declarative proof language and use of ATP Zenon
- Basic component: species
- Parametrized species (aka functors)
- (Multiple) Inheritance mechanism providing some refinement through redefinition
- Type or definition dependency helps to determine if a proof has to be erased
- Datatypes, pattern-matching, recursive functions
- Backends: Coq and Dedukti (Focalide)
- Standard FoCaLiZe library translated in Dedukti

References

Raphaël Cauderlier, Catherine Dubois. ML Pattern-Matching, Recursion, and Rewriting: From FoCaLiZe to Dedukti. ICTAC 2016.

Raphaël Cauderlier. Object-Oriented Mechanisms for Interoperability between Proof Systems. PhD thesis, 2016.

Raphaël Cauderlier, Catherine Dubois. FoCaLiZe and Dedukti to the Rescue for Proof Interoperability. ITP 2017.

Thank you!