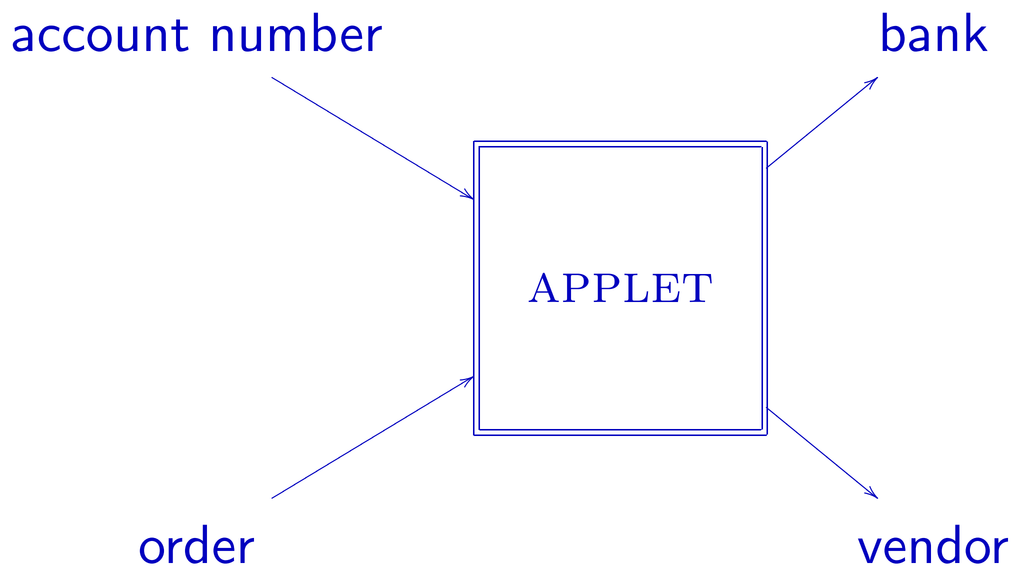
Information Flow Inference for ML

Vincent Simonet
INRIA Rocquencourt – Projet Cristal

MIMOSA

September 27, 2001

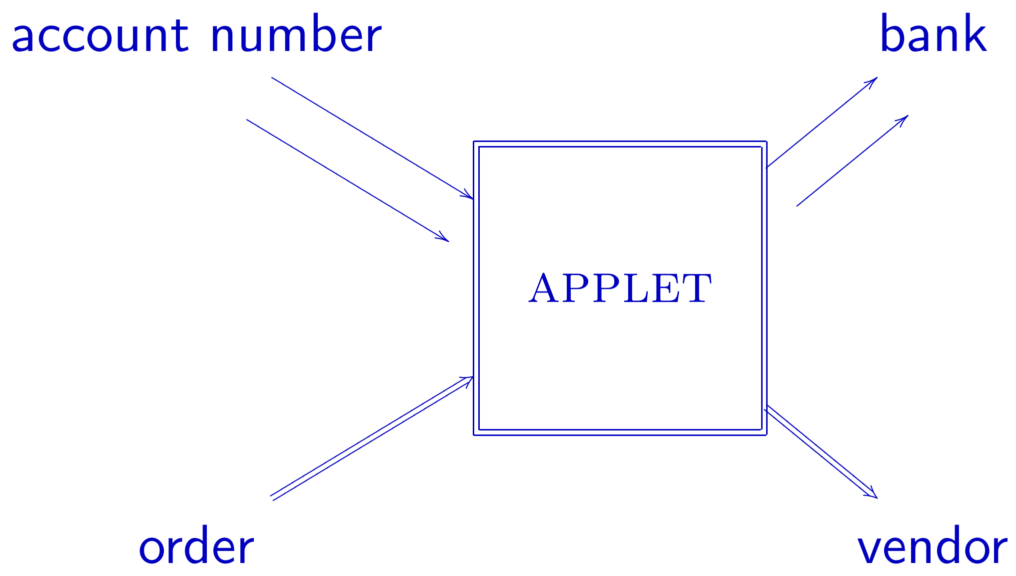
Information flow



$$\text{account}^H \times \text{order}^L \rightarrow \text{bank}^H \times \text{vendor}^L$$

$$(\forall \alpha\beta\gamma\delta)\ [\alpha \sqcup \beta \leq \gamma, \beta \leq \delta]\ \text{account}^\alpha \times \text{order}^\beta \rightarrow \text{bank}^\gamma \times \text{vendor}^\delta$$

Non-interference



Existing systems

Dennis Volpano et Geoffrey Smith (1997)
Type system on a simple imperative langage. Restricted to the first order and a finite number of global references.

Nevin Heintze et Jon G. Riecke SLam Calculus (1997)
λ-calculus with references and threads. The typing of mutable cells is not fine enough. No security property is stated.

Andrew C. Myers JFlow (1999)
Information flow analysis for Java. This sytem is complex and not proven.

Steve Zdancewic et Andrew C. Myers (2001)
Analysis on a low-level language with linear continuations.

The ML language

Call-by-value λ-calculus with let-polymorphism

$$x \qquad k \qquad \text{fun } x \rightarrow e$$

$$e_1\, e_2 \qquad \text{let } x = v \text{ in } e \qquad \text{bind } x = e_1 \text{ in } e_2$$

with references

$$\text{ref } e \qquad e_1 := e_2 \qquad !\, e$$

and exceptions

$$\varepsilon\, e \qquad \text{raise } e \qquad e_1 \text{ handle } \varepsilon\, x \succ e_2 \qquad e_1 \text{ handle } x \succ e_2$$

v-normal forms

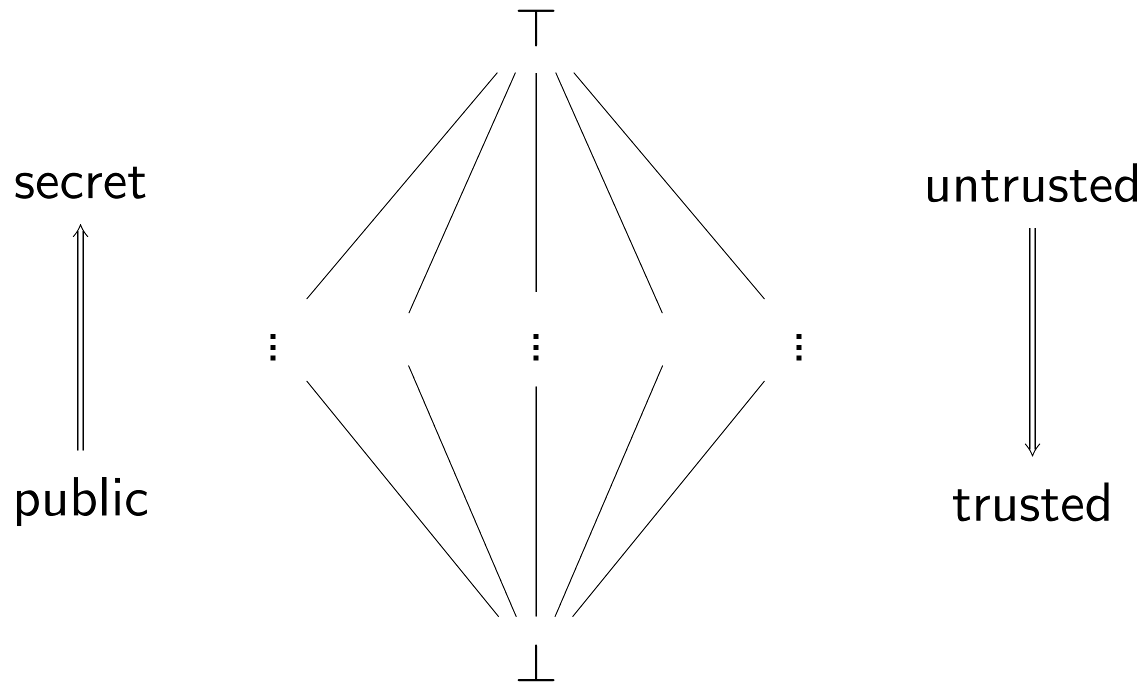
$$
\begin{array}{rcl}
v & ::= & x \mid k \mid \mathsf{fun}\ x \rightarrow e \mid \varepsilon\, v \\
e & ::= & v\, v \mid \mathsf{ref}\ v \mid v := v \mid !\, v \mid \mathsf{raise}\ v \mid \mathsf{let}\ x = v\ \mathsf{in}\ e \mid E[v] \\
E & ::= & \mathsf{bind}\ x = [\,]\ \mathsf{in}\ e \mid [\,]\ \mathsf{handle}\ \varepsilon\, x \succ e \mid [\,]\ \mathsf{handle}\ x \succ e
\end{array}
$$

Any source expression may be rewritten into a v-normal form provided an evaluation strategy is fixed :

$$
e_1\, e_2 \Rightarrow \begin{cases} \mathsf{bind}\ x_1 = e_1\ \mathsf{in}\ (\mathsf{bind}\ x_2 = e_2\ \mathsf{in}\ x_1\, x_2) & \textit{left to right eval.} \\ \mathsf{bind}\ x_2 = e_2\ \mathsf{in}\ (\mathsf{bind}\ x_1 = e_1\ \mathsf{in}\ x_1\, x_2) & \textit{right to left eval.} \end{cases}
$$

Information levels

An information level is associated to each piece of data. Information levels (which belong to a lattice $\mathcal{L}$) may represent different properties: security, integrity...



In the rest of the talk, we fix $\mathcal{L} = \{L \leq H\}$.

Direct and indirect flow

Direct flow

$$x := \text{not } y$$

$$x := (\text{if } y \text{ then } \mathit{false} \text{ else } \mathit{true})$$

Indirect flow

$$\text{if } y \text{ then } x := \mathit{false} \text{ else } x := \mathit{true}$$

$$x := \mathit{true};\ \text{if } y \text{ then } x := \mathit{false} \text{ else } ()$$

$$x := \mathit{true};\ (\text{if } y \text{ then raise } A \text{ else } ())\text{ handle } \_ \succ x := \mathit{false}$$

A level pc is associated to each point of the program. It tells how much information the expression may acquire by gaining control; it is a lower bound on the level of the expression's effects.

Semi-syntactic approach

(examples in the case of ML)

Logical system	**Syntactic system**
Ground types	Type expressions
e.g. int, int $\rightarrow$ int...	e.g. int, α, $\alpha \rightarrow \alpha$...
Polytypes	Schemes
e.g. $\{t \rightarrow t \mid t$ type brut$\}$	e.g. $\forall \alpha . \alpha \rightarrow \alpha$

We reason with the logical system. The syntactic system is interpreted into the logical one.

Type algebra

The information levels ℓ, pc belong to the lattice $\mathcal{L}$.

Exceptions are described by rows of alternatives r :

$$\begin{array}{rcl} a & ::= & \mathsf{Abs} \mid \mathsf{Pre}\ pc \\ r & ::= & \{\varepsilon \mapsto a\}_{\varepsilon \in \mathcal{E}} \end{array}$$

Types are annotated with levels and rows :

$$t \quad ::= \quad \mathsf{int}^{\ell} \mid \mathsf{unit} \mid (t \xrightarrow{pc\ [r]} t)^{\ell} \mid t\ \mathsf{ref}^{\ell} \mid r\ \mathsf{exn}^{\ell}$$

Judgements

The type system involves two kinds of judgements:

Judgements on values

$$\Gamma \vdash v : t$$

Judgements on expressions

$$pc, \Gamma \vdash e : t \ \ [r]$$

Constraints

Subtyping constraints $t_1 \leq t_2$

The subtyping relation extends the order on information levels. E.g.:

$$\mathsf{int}^{\ell_1} \leq \mathsf{int}^{\ell_2} \Leftrightarrow \ell_1 \leq \ell_2 \qquad\qquad \mathsf{Abs} \leq \mathsf{Pre}\; pc$$

Guards $\ell \lhd t$

Guards allow to mark a type with an information level:

$$pc \lhd \mathsf{int}^{\ell} \Leftrightarrow pc \leq \ell \qquad\qquad pc \lhd t\; \mathsf{ref}^{\ell} \Leftrightarrow pc \leq \ell$$

Conditional constraints $pc \leq_{\mathsf{Pre}} a$

$pc \leq_{\mathsf{Pre}} a$ is a shortcut for $a \neq \mathsf{Abs} \Rightarrow \mathsf{Pre}\; pc \leq a$.

Subtyping and polymorphism

Subtyping and polymorphism act in orthogonal ways:

Subtyping Allows increasing the level of any piece of data (e.g. considering a *public* piece of data as *secret*):

$$\frac{\Gamma \vdash v : t \qquad t \leq t'}{\Gamma \vdash v : t'}$$

Polymorphism Required for applying the same function to inputs with different levels:

$$\text{let } succ \; = \; \text{fun } x \rightarrow (x + 1)$$

References

$$\textsc{Ref}\quad \frac{\Gamma \vdash v : t \qquad pc \lhd t}{pc, \Gamma \vdash \mathsf{ref}\ v : t\ \mathsf{ref}^{\ell}\ \ [r]}$$

$$\textsc{Deref}\quad \frac{\Gamma \vdash v : t'\ \mathsf{ref}^{\ell} \qquad t' \leq t \qquad \ell \lhd t}{pc, \Gamma \vdash\ !e : t\ \ [r]}$$

$$\textsc{Assign}\quad \frac{\Gamma \vdash e_1 : t\ \mathsf{ref}^{\ell} \qquad \Gamma \vdash e_2 : t \qquad \ell \lhd t \qquad pc \lhd t}{pc, \Gamma \vdash e_1 := e_2 : \mathsf{unit}\ \ [r]}$$

The content of a reference must have a level greater than (or equal to)

- the pc of the point where the reference is created,
- the pc of each point where its content is likely to be modified.

Exceptions

RAISE

$$\frac{\Gamma \vdash v : \mathit{typexn}(\varepsilon)}{pc, \Gamma \vdash \textsf{raise}\ (\varepsilon\, v) : * \ \left[\varepsilon : \textsf{Pre}\ pc; \partial\textsf{Abs}\right]}$$

HANDLE

$$\frac{\begin{array}{c} pc, \Gamma \vdash e_1 : t \ \left[\varepsilon : \textsf{Pre}\ pc'; r_1\right] \\ pc \sqcup pc', \Gamma[x \mapsto \mathit{typexn}(\varepsilon)] \vdash e_2 : t \ \left[\varepsilon : a_2; r_2\right] \qquad pc' \lhd t \end{array}}{pc, \Gamma \vdash e_1\ \textsf{handle}\ \varepsilon\, x \succ e_2 : t \ \left[\varepsilon : a_2; r_1 \sqcup r_2\right]}$$

Non-interference

Let us consider an expression e of type int^L with a "hole" x marked H:

$$(x \mapsto t) \vdash e : \mathsf{int}^L \qquad\qquad H \lhd t$$

Non-interference

$$\text{If } \begin{cases} \vdash v_1 : t \\ \vdash v_2 : t \end{cases} \text{ and } \begin{cases} e[x \Leftarrow v_1] \longrightarrow^* v_1' \\ e[x \Leftarrow v_2] \longrightarrow^* v_2' \end{cases} \text{ then } v_1' = v_2'$$

The result of e's evaluation does not depend on the input value inserted in the hole.

Non-interference proof

1. Define a particular extension of the language allowing to reason about the common points and the differences of two programs.

2. Prove that the type system for the extended language satisfies *subject reduction*.

3. Show that non-interference for the initial language is a consequence of *subject reduction*.

Shared calculus

The shared calculus allows to reason about two expressions and proving that they share some sub-terms throughout reduction.

Syntax

$$v ::= \ldots \mid \langle v \mid v \rangle \qquad\qquad e ::= \ldots \mid \langle e \mid e \rangle$$

We restrict our attention to expressions where $\langle \cdot \mid \cdot \rangle$ are not nested.

Non-interference proof

Encoding

A shared expression encodes two expressions of the source calculus:

$$\begin{array}{ccc} \text{if } \mathit{true} \text{ then } 0 \text{ else } 1 & & \text{if } \mathit{false} \text{ then } 0 \text{ else } 1 \\ & \searrow \llbracket\cdot|\cdot\rrbracket \swarrow & \\ & \Downarrow & \\ & \text{if } \langle \mathit{true} \mid \mathit{false} \rangle \text{ then } 0 \text{ else } 1 & \end{array}$$

Two projections $\lfloor\cdot\rfloor_1$ and $\lfloor\cdot\rfloor_2$ allow to recover original expressions:

$$\begin{array}{ccc} & \text{if } \langle \mathit{true} \mid \mathit{false} \rangle \text{ then } 0 \text{ else } 1 & \\ \swarrow \lfloor\cdot\rfloor_1 & & \lfloor\cdot\rfloor_2 \searrow \\ \text{if } \mathit{true} \text{ then } 0 \text{ else } 1 & & \text{if } \mathit{false} \text{ then } 0 \text{ else } 1 \end{array}$$

Reducing the shared calculus

Reduction rules for the shared calculus are derived from the source calculus ones. When $\langle \cdot \mid \cdot \rangle$ constructs block reduction, they have to be lifted.

Example:

$$(\mathsf{fun}\ x \rightarrow e)\, v \rightarrow e[x \Leftarrow v] \qquad (\beta)$$

$$\langle v_1 \mid v_2 \rangle\, v \rightarrow \langle v_1\, \lfloor v \rfloor_1 \mid v_2\, \lfloor v \rfloor_2 \rangle \qquad (\text{lift-app})$$

Simulation

Soundness

If $e \rightarrow e'$ then $\begin{cases} \lfloor e \rfloor_1 \rightarrow^{=} \lfloor e' \rfloor_1 \\ \lfloor e \rfloor_2 \rightarrow^{=} \lfloor e' \rfloor_2 \end{cases}$

(shared calculus) (source calculus)

Completeness

If $\begin{cases} e_1 \rightarrow^{*} v_1 \\ e_2 \rightarrow^{*} v_2 \end{cases}$ then $\llbracket e_1 \mid e_2 \rrbracket \rightarrow^{*} \llbracket v_1 \mid v_2 \rrbracket$

(source calculus) (shared calculus)

Non-interference proof

Typing $\langle \ldots \mid \ldots \rangle$

$$\textsc{Bracket}\quad \frac{\Gamma \vdash v_1 : t \qquad \Gamma \vdash v_2 : t \qquad H \lhd t}{\Gamma \vdash \langle v_1 \mid v_2 \rangle : t}$$

A value whose type is int^H may be an integer k or a bracket $\langle k_1 \mid k_2 \rangle$.

A value whose type is int^L must be a simple integer k.

Subject reduction and non-interference

Let us consider $(x \mapsto t) \vdash e : \mathsf{int}^L$ with $H \lhd t$.

Subject-reduction

If $\vdash e' : \mathsf{int}^L$ and $e' \rightarrow^* v'$ then $\vdash v' : \mathsf{int}^L$

$$\uparrow$$

$$e' = e[x \Leftarrow v] \qquad\qquad\qquad v' = k$$

$$\downarrow$$

Non-interference (shared calculus)

If $\vdash v : t$ and $e[x \Leftarrow v] \rightarrow^* v'$ then $\lfloor v' \rfloor_1 = \lfloor v' \rfloor_2$

Non-interference

Let us consider $(x \mapsto t) \vdash e : \mathsf{int}^L$ with $H \lhd t$.

Non-interference (shared calculus)

If $\vdash v : t$ and $e[x \Leftarrow v] \rightarrow^* v'$ then $\lfloor v' \rfloor_1 = \lfloor v' \rfloor_2$

$$\uparrow \qquad\qquad\qquad\qquad\qquad\qquad |$$

$$v = \langle v_1 \mid v_2 \rangle \qquad\qquad\qquad\qquad v' = [\![v_1 \mid v_2]\!]$$

$$| \qquad\qquad\qquad\qquad\qquad\qquad \downarrow$$

Non-interference (source calculus)

If $\begin{cases} \vdash v_1 : t \\ \vdash v_2 : t \end{cases}$ and $\begin{cases} e[x \Leftarrow v_1] \rightarrow^* v'_1 \\ e[x \Leftarrow v_2] \rightarrow^* v'_2 \end{cases}$ then $v'_1 = v'_2$

Extending the language

One can extend the studied language in order to

Increase its expressiveness Adding sums, products. A general case for primitive operations of real languages (arithmetic operations, comparisons, hashing...)

Have a better typing of some idioms

$$e_1 \text{ finally } e_2 \hookrightarrow \text{bind } x = (e_1 \text{ handle } y \succ e_2;\ \text{raise } y) \text{ in } e_2;\ x$$

$$e_1 \text{ handle } x \succ e_2 \text{ reraise} \hookrightarrow e_1 \text{ handle } x \succ (e_2;\ \text{raise } x)$$

Our approach allows to deal with such extensions in a simple way: one just needs to extend the *subject reduction* proof with the new reduction rules.

Primitive operations

$$\frac{\Gamma \vdash v_1 : \mathsf{int}^{\ell} \qquad \Gamma \vdash v_2 : \mathsf{int}^{\ell}}{pc, \Gamma \vdash v_1 + v_2 : \mathsf{int}^{\ell} \ [\partial \mathsf{Abs}]} \qquad \frac{\Gamma \vdash v_1 : t \qquad \Gamma \vdash v_2 : t \qquad t \blacktriangleleft \ell}{pc, \Gamma \vdash v_1 = v_2 : \mathsf{bool}^{\ell} \ [\partial \mathsf{Abs}]}$$

$$\frac{\Gamma \vdash v : t \qquad t \blacktriangleleft \ell}{pc, \Gamma \vdash \mathsf{hash}\ v : \mathsf{int}^{\ell} \ [\partial \mathsf{Abs}]}$$

A new form of constraints $t \blacktriangleleft \ell$

$t \blacktriangleleft \ell$ constrains *all* information levels in t and its sub-terms to be less than (or equal to) ℓ.

Extending the language

Products

$$t \quad ::= \quad \ldots \mid t_1 \times t_2$$

Products carry no security annotations because, in the absence of a physical equality operator, all of the information carried by a tuple is in fact carried by its components:

$$\begin{array}{rcl} \ell \lhd t_1 \times t_2 & \Leftrightarrow & \ell \lhd t_1 \wedge \ell \lhd t_2 \\ t_1 \times t_2 \blacktriangleleft \ell & \Leftrightarrow & t_1 \blacktriangleleft \ell \wedge t_2 \blacktriangleleft \ell \end{array}$$

Towards an extension of the Caml compiler

The studied language allows us to consider the whole Caml language (excepted the `threads` library).

We are currently implementing a prototype. It will require to solve several problems due to the use of a type system with subtyping:

- Efficiency of the inference algorithm
- Readability of the inferred types
- Clarity of error messages
- ...

Type inference

An inference algorithm is divided into two distinct parts.

A set of inference rules It may be derivated from typing rules in a quasi-systematic way.

$$\textsc{Ref}\ \frac{\Gamma \vdash v : t \qquad pc \lhd t}{pc, \Gamma \vdash \mathsf{ref}\ v : t\ \mathsf{ref}^{\ell}\ \ [r]} \quad \leadsto \quad \textsc{Inf-Ref}\ \frac{\Gamma, C \vdash v : \alpha}{\pi, \Gamma, C \cup \{\beta = \alpha\ \mathsf{ref}^{\lambda}, \pi \lhd \alpha\} \vdash \mathsf{ref}\ v : \beta\ \ [\rho]}$$

A solver Type schemes involve constraint sets. It is necessary to test their satisfiability and to simplify them.

Example: lists

```
type ('a, 'b) list = <'b>
  | []
  | (::) of 'a * ('a, 'b) list
```

```
let rec length = function
  | []       -> 0
  | _ :: l -> 1 + length l
```

$$\forall[\alpha \leq \beta].\ * \ \mathsf{list}^{\alpha} \rightarrow \mathsf{int}^{\beta}$$

Example: lists (2)

```
let rec iter f = function
  | []     -> ()
  | x :: l -> f x; iter f l
```

$$\forall[\delta \leq \partial\gamma].(\alpha \xrightarrow{\gamma\ [\delta]} *)^{\gamma} \rightarrow \alpha\ \mathsf{list}^{\gamma} \xrightarrow{\gamma\ [\delta]} \mathsf{unit}$$

```
let rec iter2 f = fun
  | []          []          -> ()
  | (x1 :: l1) (x2 :: l2) -> f x1 x2; iter2 f l1 l2
  | _           _           -> raise X
```

$$\forall[\epsilon \leq \zeta; \mathsf{Pre}\ \gamma \leq \zeta; \delta \leq \partial\gamma].$$

$$(\alpha \xrightarrow{\gamma\ [X:\epsilon;\delta]} \beta \xrightarrow{\gamma\ [X:\epsilon;\delta]} *)^{\gamma} \rightarrow \alpha\ \mathsf{list}^{\gamma} \rightarrow \beta\ \mathsf{list}^{\gamma} \xrightarrow{\gamma\ [X:\zeta;\delta]} \mathsf{unit}$$