Specification of ONNX operator: concat

COMMERCIAL AIRCRAFT

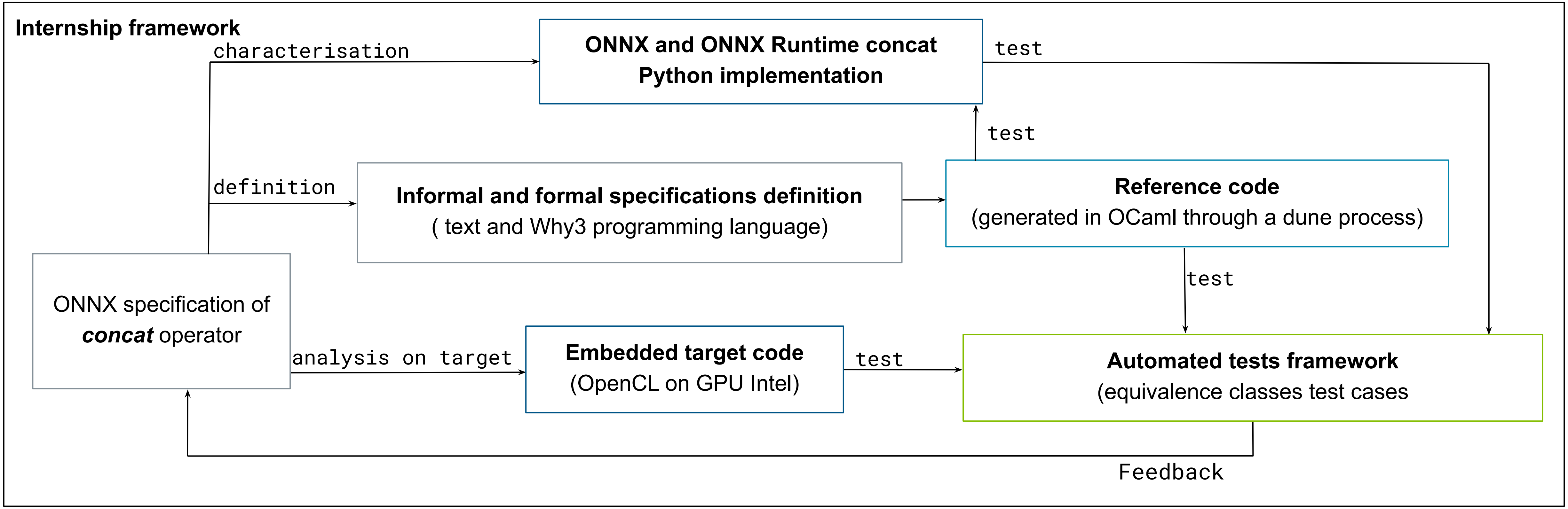
Salomé Marty Laurent - 1YYWA
14/05/2025

AIRBUS

Agenda

- Global workflow
- General ONNX specification
- SONNX specification
- Reference code and validation tests
- Conclusion

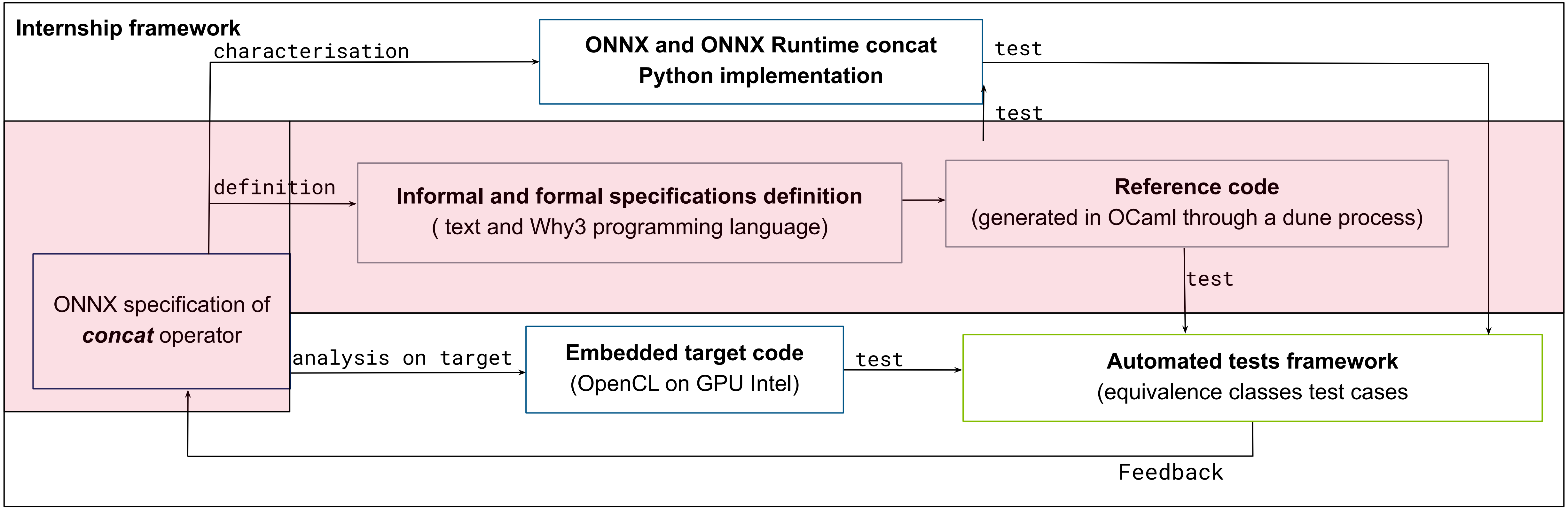
Global workflow



- Closed loop process outlined above for the ONNX ***concat*** operator serves as a template.
- Future work will involve repeating this methodology for additional key ONNX operators such as ***reLU, maxPool, and resize***, progressively building a library of formally verified and tested implementations.

AIRBUS

Today’s topic



- Today's topic, highlighted by the red outline, centers on studying the ONNX specification of ***concat***, then looking at the informal and formal specifications (our SONNX specifications) for the ONNX ***concat*** operator and finishing by the reference code generation.

AIRBUS

ONNX *concat* operator specification

Definition:

Concatenate a list of tensors into a single tensor. All input tensors must have the same shape, except for the dimension size of the axis to concatenate on.

- Focus on ONNX ***concat*** operator :
  - **Version 13**

- **Main parameter / inputs / output :**
  - **Parameter :** `axis(int)`
  - **Input(s) :** ***variadic***
  - **Output :** tensor

- **Main data types supported :**
  - **16 types :** `tensor(bfloat16)` … `tensor(uint8)` including `tensor(bool)` and `tensor(string)`

- To fully grasp the ONNX ***concat*** operator, its specification is best understood alongside the ONNX Runtime implementation.

- The official ONNX definition alone is incomplete, and its limitations will be highlighted when compared to our SONNX informal specification.

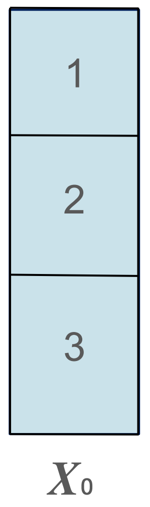
Example

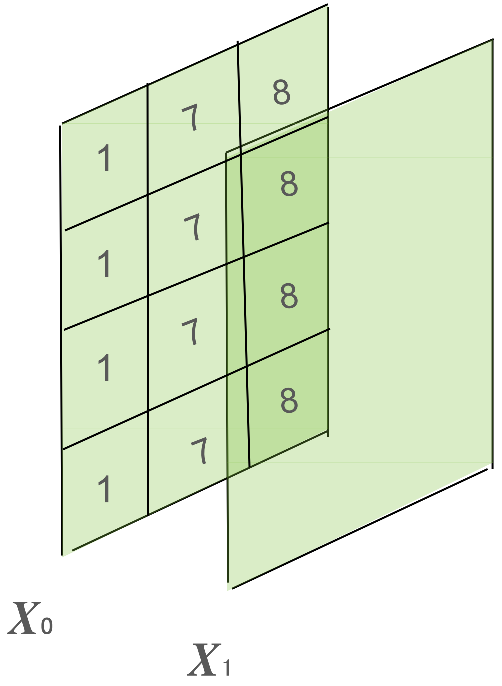
– Inputs (***variadic***)

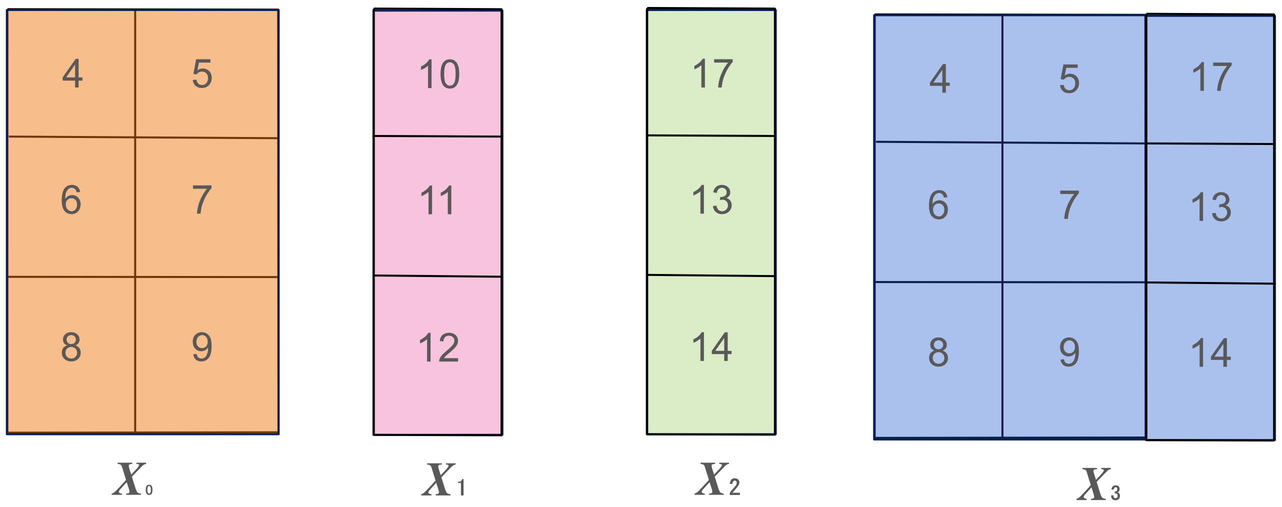
Case 1: one input tensor

Case 2: two input tensors

Case 3: four input tensors





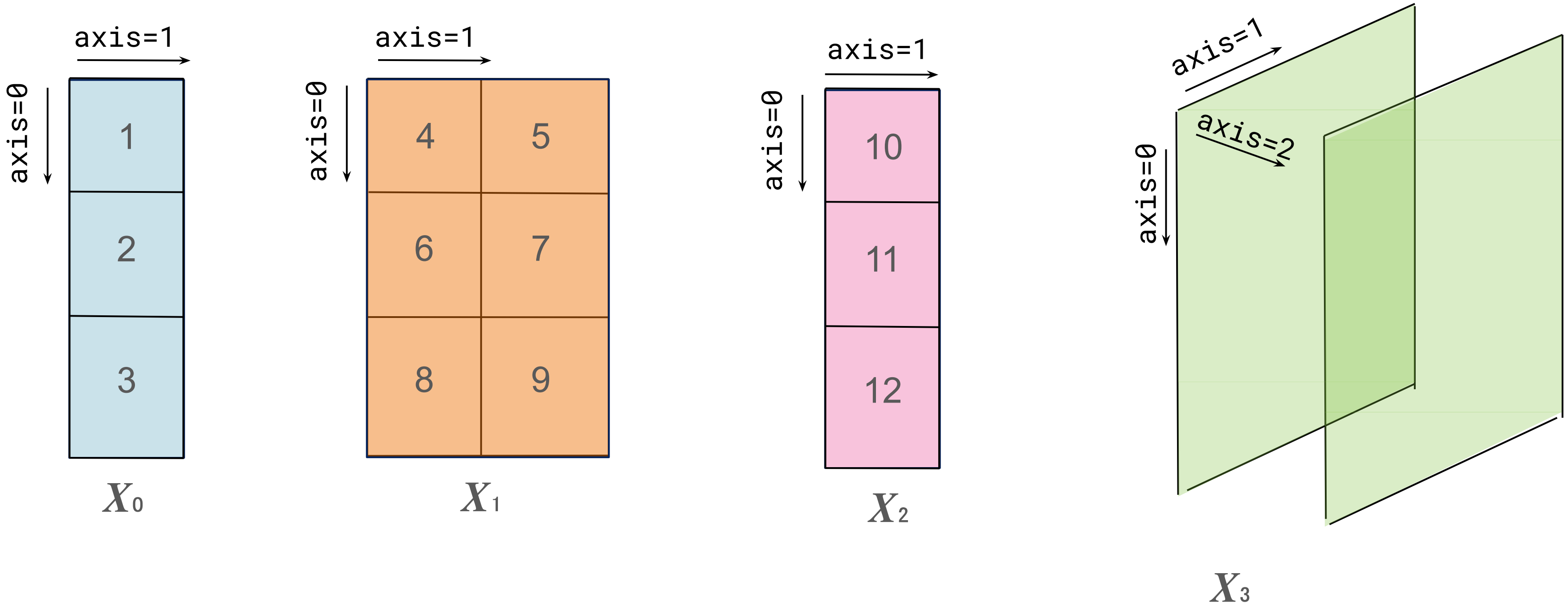


- These illustrations show input values ranging from 1 to **4**.

AIRBUS

Example

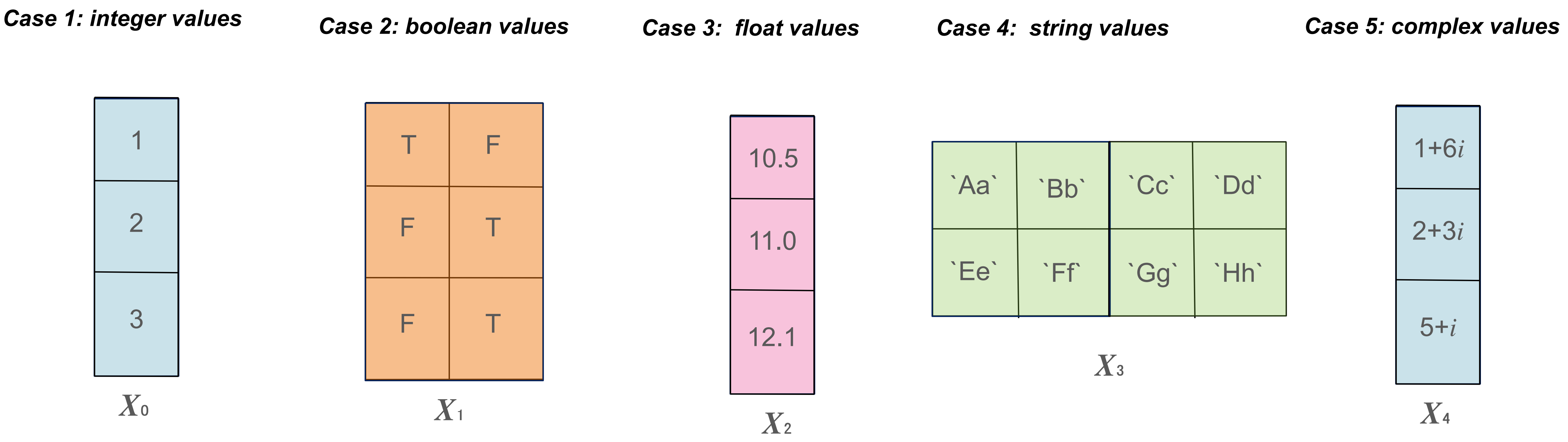
– `axis(int)`



- These illustrations show axis values ranging from **0** to **2**.

Example

– Data types



- Illustrations of different data type values as **integer**, **float**, **boolean**, **string** and **complex**.

Informal specification

- Document **<u>concat.md</u>** available in the folder :
  working-groups / safety-related-profile / documents / profile_opset / concat

Inputs constraints :

- Bounded number of inputs from **1** tensor to **2^{31}-1** tensors.
- All input tensors must have identical dimensions, except for the dimension specified by the **axis.**

Output constraints :

- Same rank as input tensors.
- Same dimension except for the dimension along the ***axis*** which should be the sum of the dimensions of the input tensors.

Attribute constraints

- ***axis*** value ranges from **0** to **rank-1** and the lower bound restricted to **0** in our case.
- Tensors with rank equal to 0 (scalars) are not considered.

concat.md

SONNX ***concat*** operator signature

$$Y = Concat(X_0, \ldots, X_n)$$

ONNX ***concat*** operator limitations

- Limited details in the ONNX specification regarding the output dimension along the **axis** being the sum of input dimensions.
- No details about the rank definition.
- No information about the scalar handling.
- ONNX ***concat*** operator specification includes `tensor(complex64)` and `tensor(complex128)` types, ONNX Runtime not support them (on the CPU), leading to errors even with a valid ONNX model.

Informal specification

Mathematical semantics of *concat*:

Let $\boldsymbol{a}$ the concatenation axis,

Let $d_{k,a}$ the dimension of the X_k input tensor k along the axis $\boldsymbol{a}$,

Let $\mathbf{Y}$ the output tensor with $\text{shape}(Y) = (d_0, d_1, \ldots, d_{r-1})$,

Let $\mathbf{r}$ the rank of the input tensors where $r = dim(inputs)$,

Let i_a the global index along a and be i'_a the local index associated with $i_a$for a local tensorX_k where $i'_a = i_a - s_k$

$$s_k = \sum_{j=0}^{k-1} d_{j,a} \text{ and } Y[i_0, \ldots, i_{r-1}] = X_k[i_0, \ldots, i_a - s_k, \ldots, i_{r-1}] \text{ if } s_k \leq i_a < s_k + d_{k,a}$$

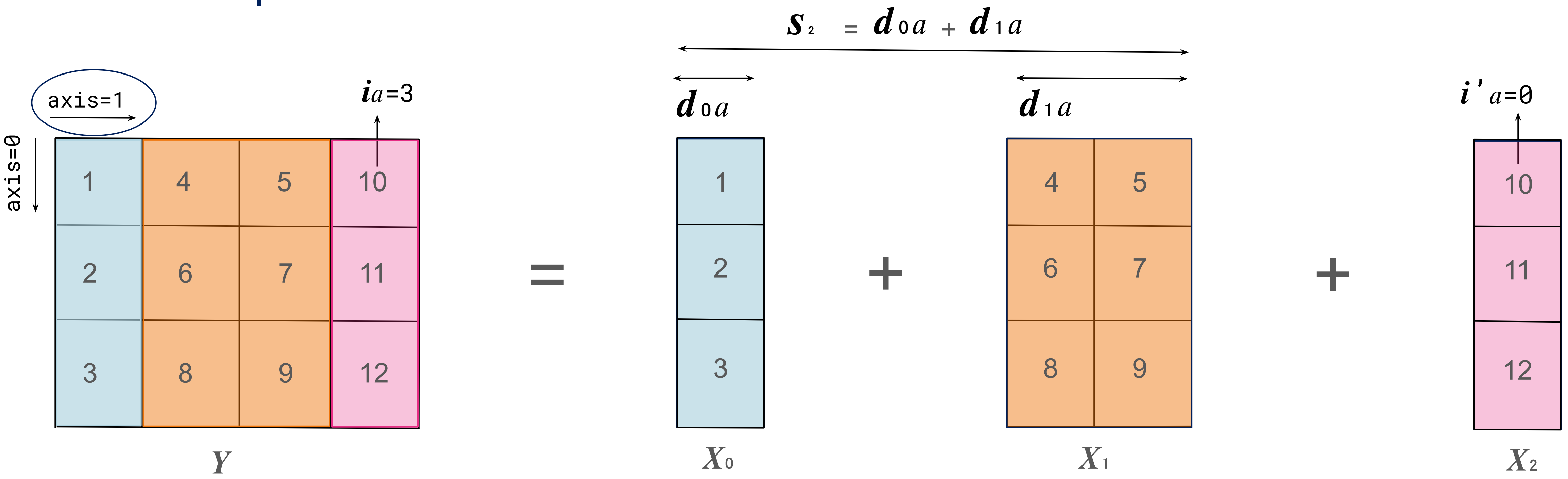
s_k be the cumulative offset along axis $\boldsymbol{a}$ before input $\boldsymbol{X}_\mathbf{k}$.

This equality illustrates the behavior of the ***concat*** operator.

This inequality determines which tensor $\boldsymbol{k}$ the element at global index $\boldsymbol{i_a}$ originates from.

AIRBUS

Informal specification



i_a Global index along the axis a.

i'_a Local index along the axis a.

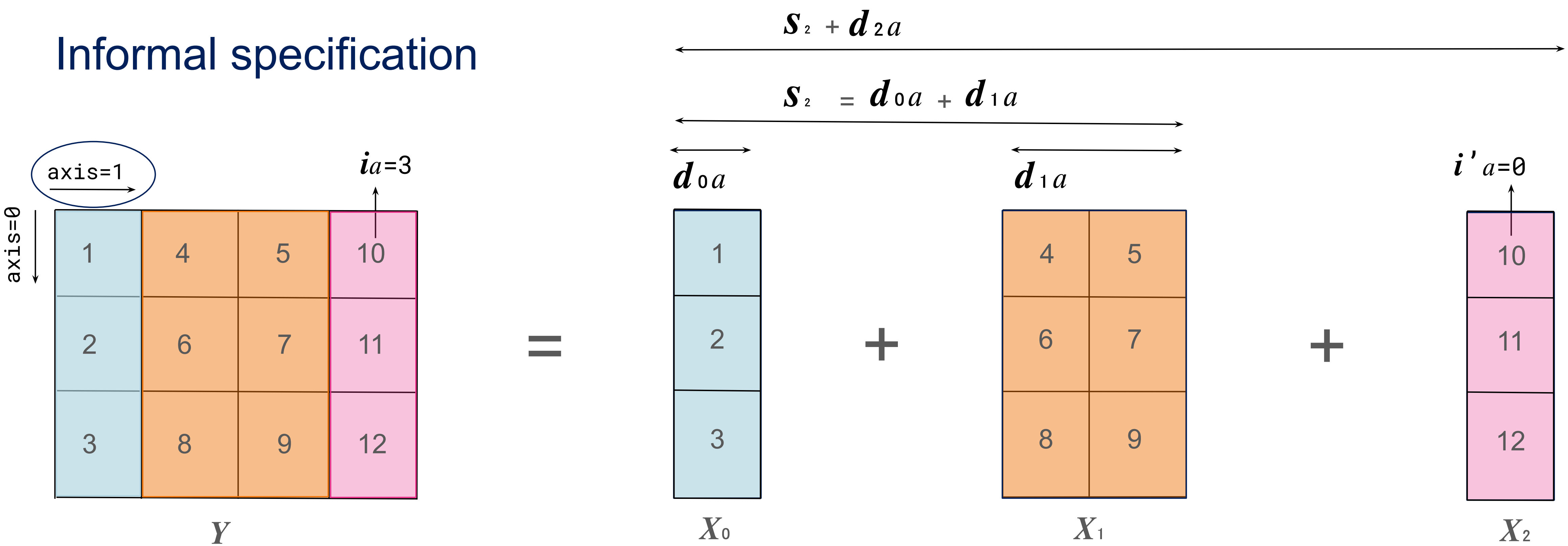
$$Y = Concat(X_0, \ldots, X_n)$$

$$Y = Concat(X_0, X_1, X_2) \text{ with } axis = 1$$

d_{0a} Dimension of the tensor **0**, X_0 along the axis a.

d_{1a} Dimension of the tensor **1**, X_1 along the axis a.

Informal specification



$$s_k = \sum_{j=0}^{k-1} d_{j,a} \text{ and } Y[i_0, \ldots, i_{r-1}] = X_k[i_0, \ldots, i_a - s_k, \ldots, i_{r-1}] \text{ if } s_k \leq i_a < s_k + d_{k,a}$$

$$Y[0, \ldots, 2, i_a] = X_2[0, \ldots, 2, i'_a] = X_2[0, \ldots, 2, i_a - s_2] = X_2[0, \ldots, 2, 3 - s_2] = X_2[0, \ldots, 2, 3-3] = X_2[0, \ldots, 2, 0]$$

$$= \begin{pmatrix} 10 \\ 11 \\ 12 \end{pmatrix}$$

AIRBUS

Informal specification

Complex operator :

- Define for n tensors from **1** tensor to **2^{31}-1** tensors.
- Define for m dimensions from **1** to **2^{31}-1** dimensions.

Source of complexity: defining the mathematical semantics of ***concat*** considering a generic behavior.

- Define for many different data types including *string* and *boolean*.

Constructing the definition :

- Multiple references used (NNEF, ONNX specification, ONNX Runtime etc.)
- Using diverse examples (*cf working-groups / safety-related-profile / documents / profile_opset / concat / tests*).

Previous slide.

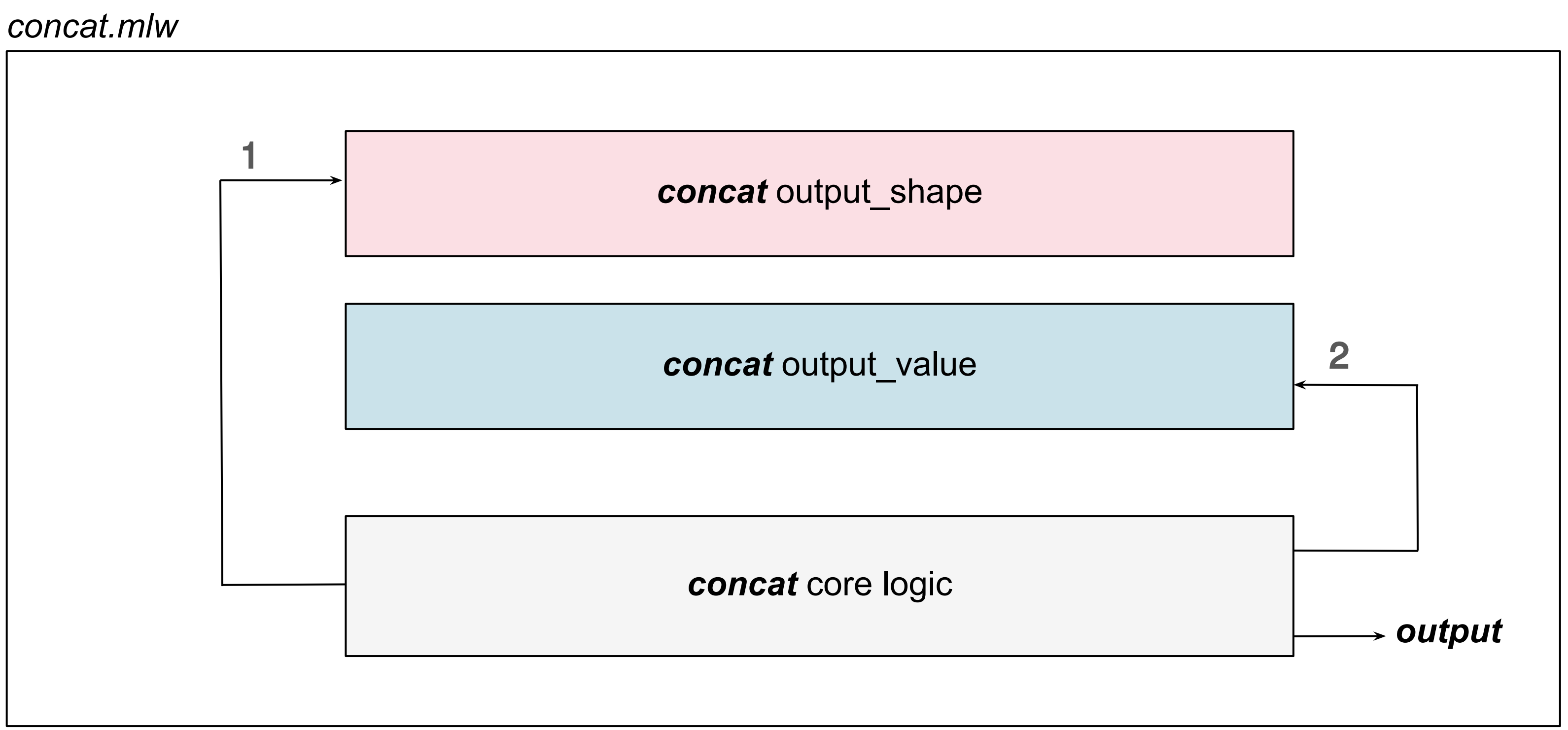
AIRBUS

Formal specification

- Document **<u>concat.mlw</u>** available in the folder :
  working-groups / safety-related-profile / documents / profile_opset / concat / why3

Concat.mlw architecture:

- Based on L. Correnson ONNX ***where*** operator work.



AIRBUS

Traceability between formal and informal specifications

Specifications coherence :

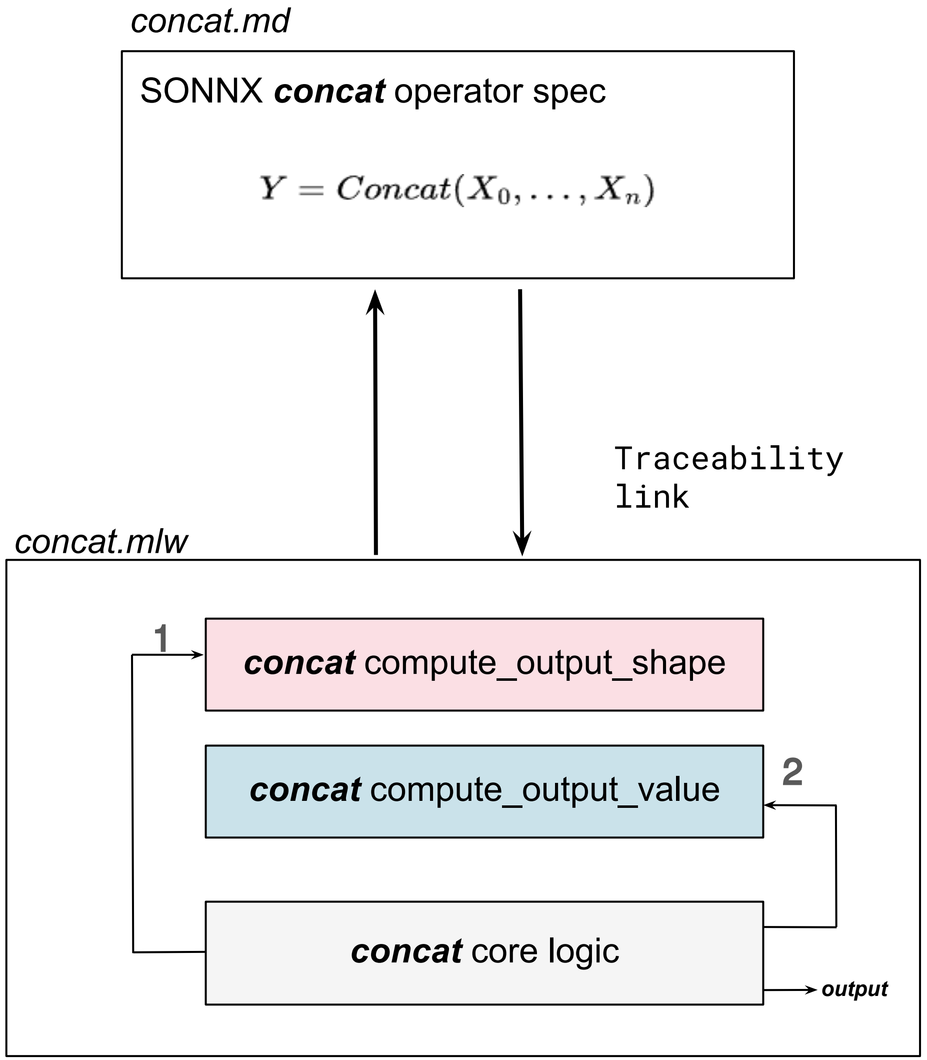
- Link between informal specification and the formal one.
- Ease for the user of reading both specifications and understanding their coherence.

Key points (E1… E9) :

- Major notions in the informal specification that need to be found in (or reflected in) the formal specification.
- In theory, these points guide the development of the formal specification.

Alignment of variables names

- Reuse of the same variable names to refer to definitions introduced in the informal specification.
- Function names also refer to these concepts.



AIRBUS

Reference code generation

Serves as a reference standard for testing :

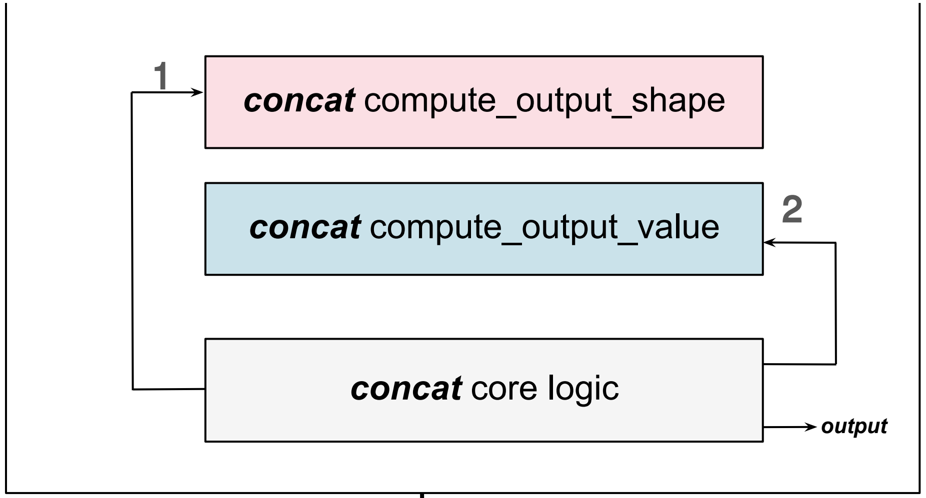
- Backed by formal proofs, the generated OCaml code provides a definitive reference for testing the correctness of others ***concat*** operator implementations.
- Crucial for comprehensive testing processes.

Complexity of the extraction process & configuration :

- The process relies heavily on dune, a build system that requires considerable time to master.

OCaml reference code

- Used for testing the Why3 implementation before mastering the proof process.
- The rigor and strong typing from Why3 are propagated to the OCaml code.
- Leveraged by the automated test framework.



concat.ml

concat operator in OCaml:
let concat (**x_k** (a **Tensor__Tensor.tensor**) list)
(axis: int) : (a **Tensor__Tensor.tensor**) =
fun x_k axis ->
{ **Tensor__Tensor.shape** = compute_output_shape
a_k axis;
Tensor__Tensor.value = compute_output_value
a_k axis }

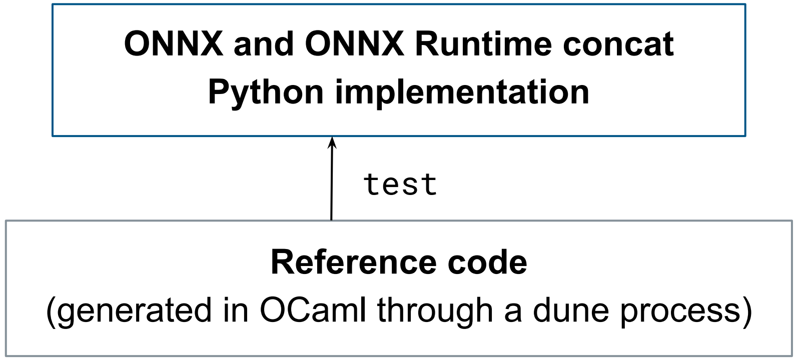
Validation tests

Test to check the correctness of our definition of *concat* :

- To validate our Why3 formal specification of the ***concat*** operator *prior* to undertaking the full proof process, we first evaluated its OCaml implementation.

- Our strategy involved comparing this implementation behavior against the standard ONNX and ONNX Runtime Python versions using a diverse set of test cases to ensure consistency.

- Our initial testing phase used L. Correnson's existing dune project.



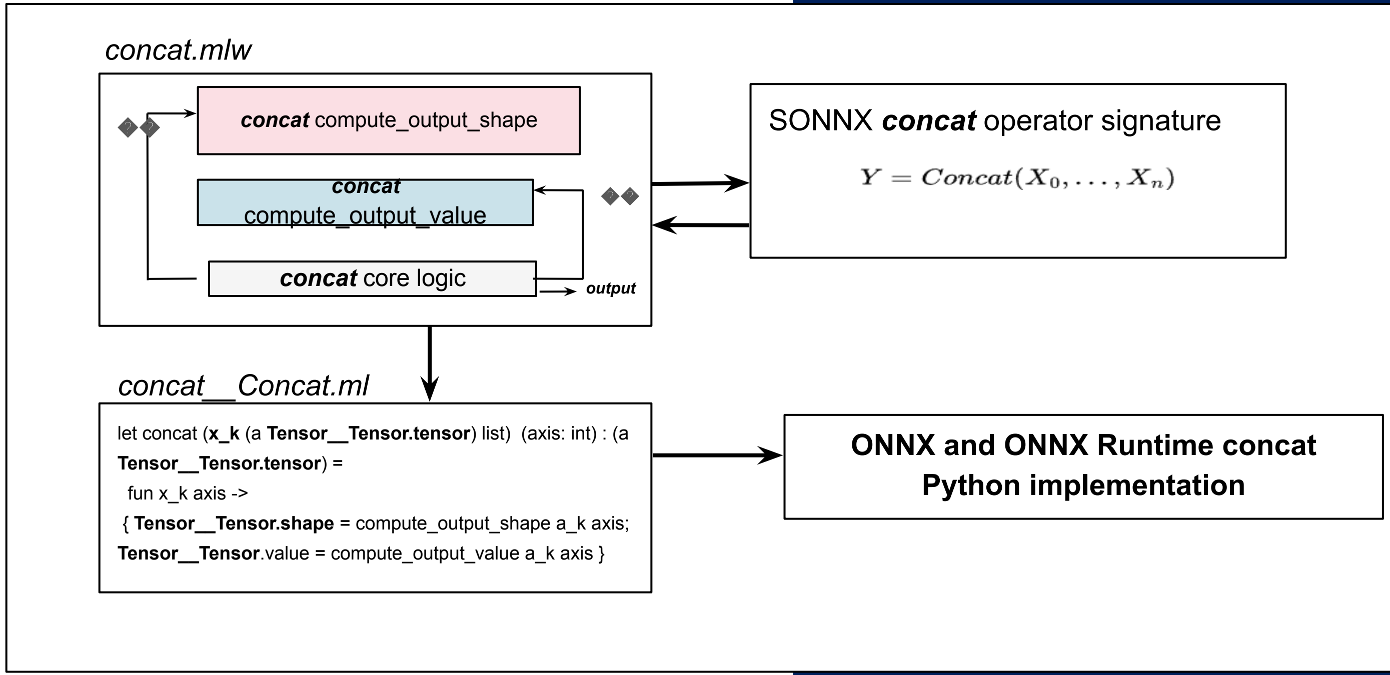
- We adapted its *make test* command and dune files, implementing specific *test.expected* file to verify the output for floating-point and integer values.

- The need for more comprehensive testing across a broader range of data types, including string and boolean tensors, directly motivated the development of the automated test framework.

Conclusion

Key takeaways :

- The ***concat*** operator's complexity stems from its variadic input (variable number of tensors) and the extensive range of dimensions it supports.
- Refining the informal specification for conciseness without sacrificing accuracy demonstrated a clear trade-off between verbosity and precision.
- The formal specification, though potentially seeming complex at first glance, faithfully translates the key points from its informal counterpart.



Process overview diagram

- Rather than being independent, the formal and informal specifications are mutually reinforcing and complementary.
- Consistent syntax across formal and informal specifications, which can be applied to other operators, helps ensure and demonstrate overall specification coherence.
- Generated OCaml code significantly benefits the testing process by providing a reliable standard against which other ***concat*** implementations can be verified.

Futur work

Planned next steps :

- **Declarative and imperative specifications:** further investigate the application and interplay of both declarative and imperative specification styles for ONNX operators.

- **Advancement of the proof process:** continue to enhance the existing proof process. Extending the scope of formal verification to cover a broader range of ***concat*** properties.

- **Dune project organization and shared libraries:** refine the dune build system configuration and project structure. A primary objective is to establish a common library (lib) folder to house shared functionalities in the SONNX repository.

- **Broader operator coverage with the established process:** apply the formal specification, verification, and reference code generation framework (as developed for ***concat***) to other ONNX operators as ***reLU, maxpool and resize***.

AIRBUS