SONNX WG

Towards an ONNX profile for critical systems

SONNX Working Group

Speaker: Eric JENN , IRT Saint-Exupery, WG co-lead (with Jean SOUYRIS, Airbus)

Main contributors (to this presentation)

- Henri BELFY (Thales AVS)
- Sebastian BOBLEST (BOSCH)
- Loïc CORRENSON (CEA LIST)
- Jean-Loup FARGES (ONERA)
- Eric JENN (IRT Saint-Exupery, co lead)
- Cong LIU (Collins)
- João Costa MACHADO (Critical Software)
- Edoardo MANINO (U of Manchester)
- Ricardo MOREIRA (Critical Software)
- Jean-Baptiste ROUFFET (Airbus Protect)
- Jean SOURYS (Airbus Commercial, co-lead)
- Mariem TURKI (IRT Saint-Exupery)
- Nicolas VALOT (Airbus Helicopter)
- Franck VEDRINE (CEA LIST)

ONNX

Agenda

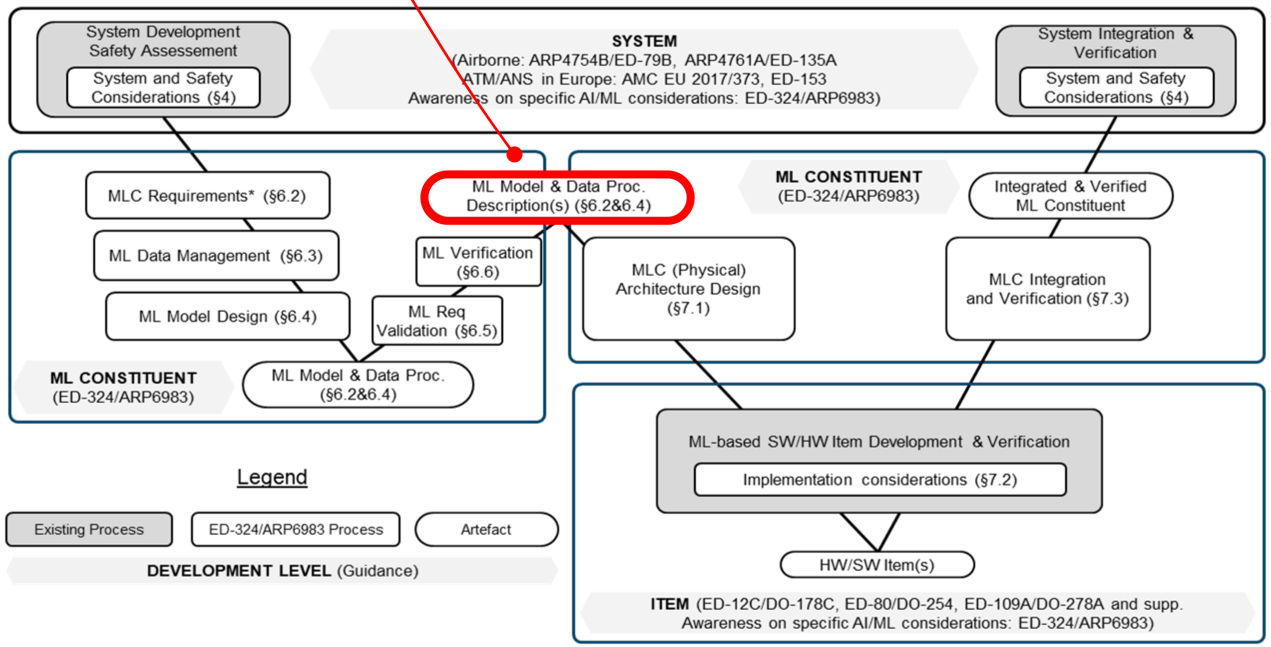
- Objectives, organization, workplan of the Working Group…
- Some issues addressed by the WG…
- Some issues not addressed by the WG… 

- Some first results…
- Next…

Objectives of the SONNX WG

Towards a “safe” profile…

- General objective
  - Provide a language to describe ML models



Objectives of the SONNX WG

Towards a “safe” profile…

- General objective
  - Provide a language to describe ML models

1. Complete the ONNX standard
   - Clarify the semantics of operators and graph…
   - Remove ambiguities…
2. Restrict the ONNX standard
   - To simplify demonstration of compliance with standards (e.g. aero standards)
   - To facilitate 1)
3. Provide a simple reference implementation compliant with the standard

What is ONNX?

ONNX

- A set of operators and a graph execution semantics
- An API
- An Intermediate Representation (IR) described using Protobuf
- A “reference implementation” coded in Python
- A runtime (ONNXruntime) [managed as a separate project in [ONNX Runtime | Home](#)]

ONNX

ONNX 1.19.0 documentation

Search

Introduction to ONNX
API Reference
ONNX Operators
Sample operator test code
Abs
Acos
Acosh
ArgMax
ArgMin

ONNX Operators

Lists out all the ONNX operators. For each operator, lists out the usage guide, parameters, examples, and line-by-line version history. This section also includes tables detailing each operator with its versions, as done in Operators.md.

All examples end by calling function *expect*. which checks a runtime produces the expected output for this example. One implementation based on onnxruntime can be found at Sample operator test code.

ai.onnx | ai.onnx.ml | ai.onnx.preview.training

operator	versions	differences
Abs	13, 6, 1	13/6, 13/1, 6/1
Acos	22, 7	22/7
		22/9
		14/13, 14/7, 13/7, 14/6, 13/6, 7/6, 14/1, 13/1, 7/1, 6/1
And	7, 1	7/1

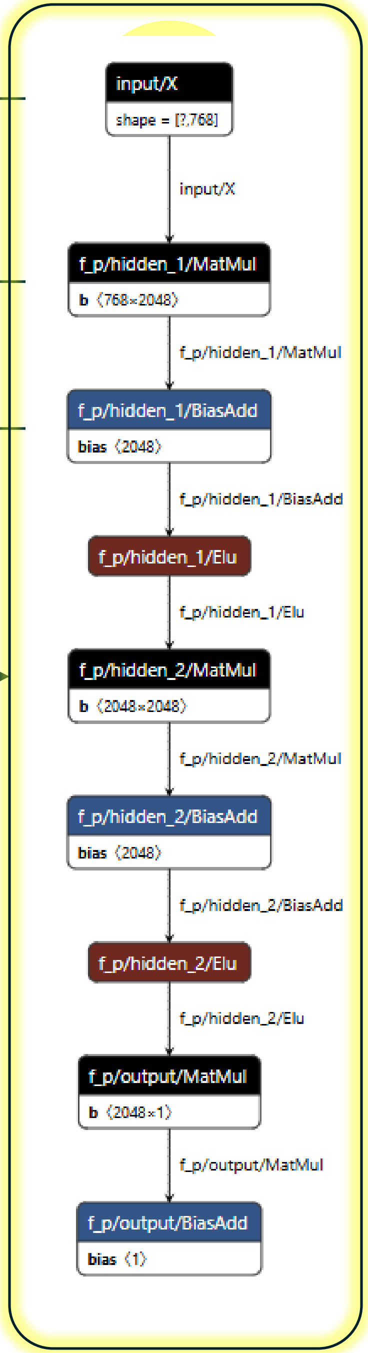
124 operators in ai.onnx
19 operators in ai.onnx.ml domain

```
    // Additional named attributes.
    repeated AttributeProto attribute = 5;

    // A human-readable documentation for this node. Markdown is allowed.
    optional string doc_string = 6;

    // Named metadata values; keys should be distinct.
    repeated StringStringEntryProto metadata_props = 9;

    // Configuration of multi-device annotations.
    repeated NodeDeviceConfigurationProto device_configurations = 10;
}
```

Challenges for SONNX

- Provide an accurate description of the ML model, leaving no room to interpretation and approximations
  - The operator semantics (for all datatypes)
  - The graph semantics
  - The ONNX abstract (metamodel) and concrete (format) syntax

```
ir_version: 8
opset_import {
  domain: ""
  version: 13
}
producer_name: "example"
graph {
  name: "SimpleReluGraph"

  input {
    name: "input_tensor"
    type {
      tensor_type {
        elem_type: 1  # FLOAT
        shape {
          dim { dim_value: 1 }
          dim { dim_value: 3 }
        }
      }
    }
  }
```



Typical flow

PyTorch

ONNX

Build and train model → Serialize model → Deploy and Execute model

XILINX VITIS

STM32 Cube.AI

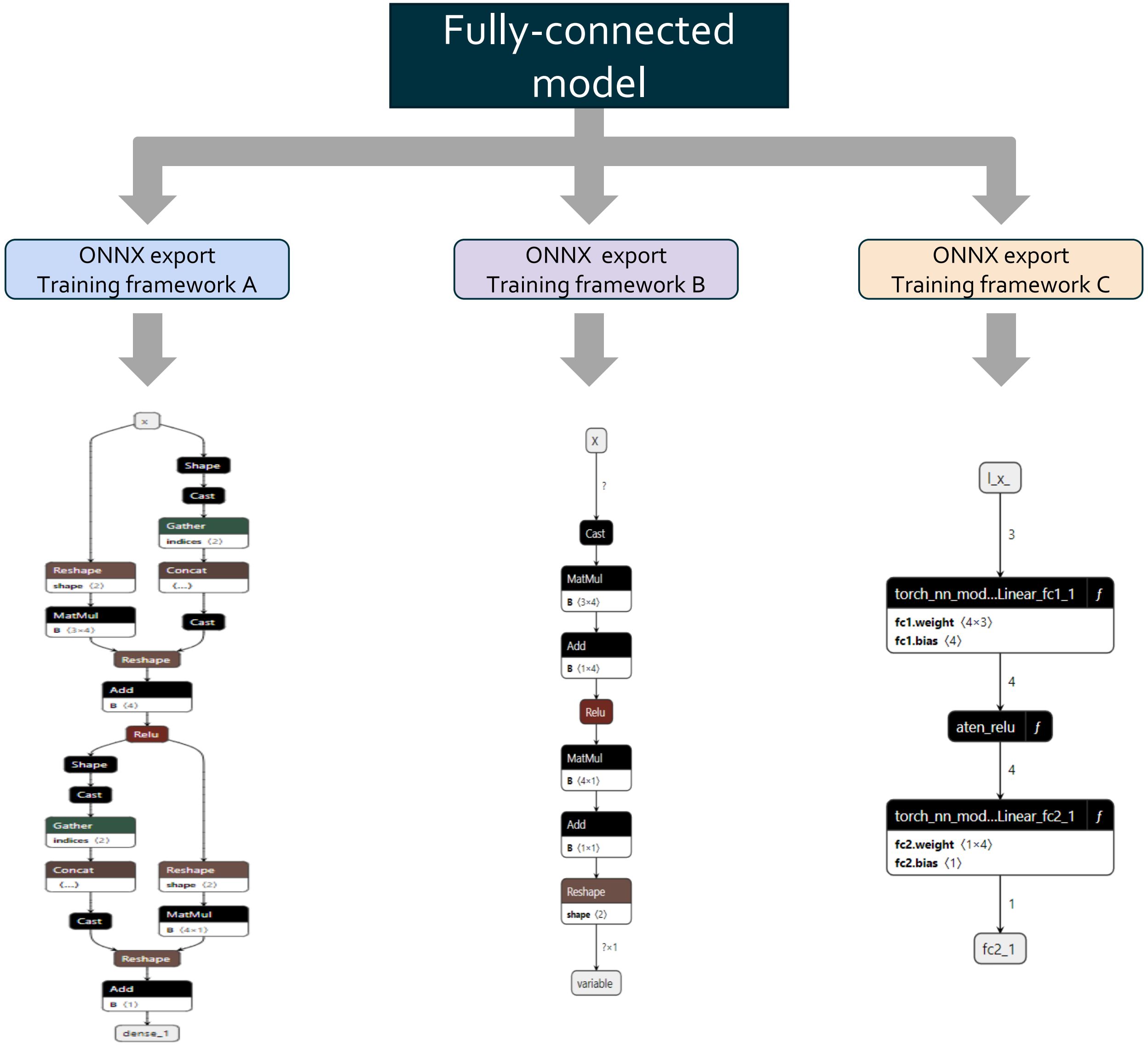
ONNX RUNTIME


```
import torch
import torch.nn as nn

# Define model directly (no class)
torch_model = nn.Sequential(
    nn.Linear(4, 3),
    nn.ReLU()
)
torch_model.eval()

# Random but fixed weights for reproducibility
with torch.no_grad():
    torch_model[0].weight.copy_(torch.randn_like(torch_model[0].weight))
    torch_model[0].bias.copy_(torch.randn_like(torch_model[0].bias))

dummy = torch.randn(2, 4)  # [batch=2, features=4]
```

```
# Export to ONNX
pt_onnx_path = "from_pytorch.onnx"
torch.onnx.export(
    torch_model, dummy, pt_onnx_path,
    input_names=["x"], output_names=["y"],
    opset_version=15, do_constant_folding=True
)
```

Export

The same graph generates different ONNX models



(Example provided by Nicolas VALOT, Airbus Helicopter and ONERA)

SONNX

Meetings and attendance

- Self-funded…
- 17 bi-weekly meetings (see minutes at https://github.com/ericjenn/working-groups/blob/ericjenn-srpwg-wg1/safety-related-profile/meetings/minutes.md)
- Actual participation
  - Around 6-15 people per meeting

CEA (FR)
INRIA (FR)
IRT Saint-Exupery (FR)
ISAE SupAero (FR)
ONERA (FR)
TUM (DE)
U of Manchester (UK)
U of Minho (PT)

- **Aeronautics** : Airbus Helicopter, Airbus Operations, Airbus Protect, Collins, Embraer, Safran Electronics an Defense, THALES AVS, THALES Research and Technologies, DGA-TA
- **Space** : Airbus Defence and Space
- **Automotive** : Bosch, Ampere
- **Naval**: Naval Group
- **Industry** : Trumpf, Crosscontrol
- **Energy**: ARCYS
- **Other**: Critical Software, SopraSteria, Mathworks, Infineon, ANSYS

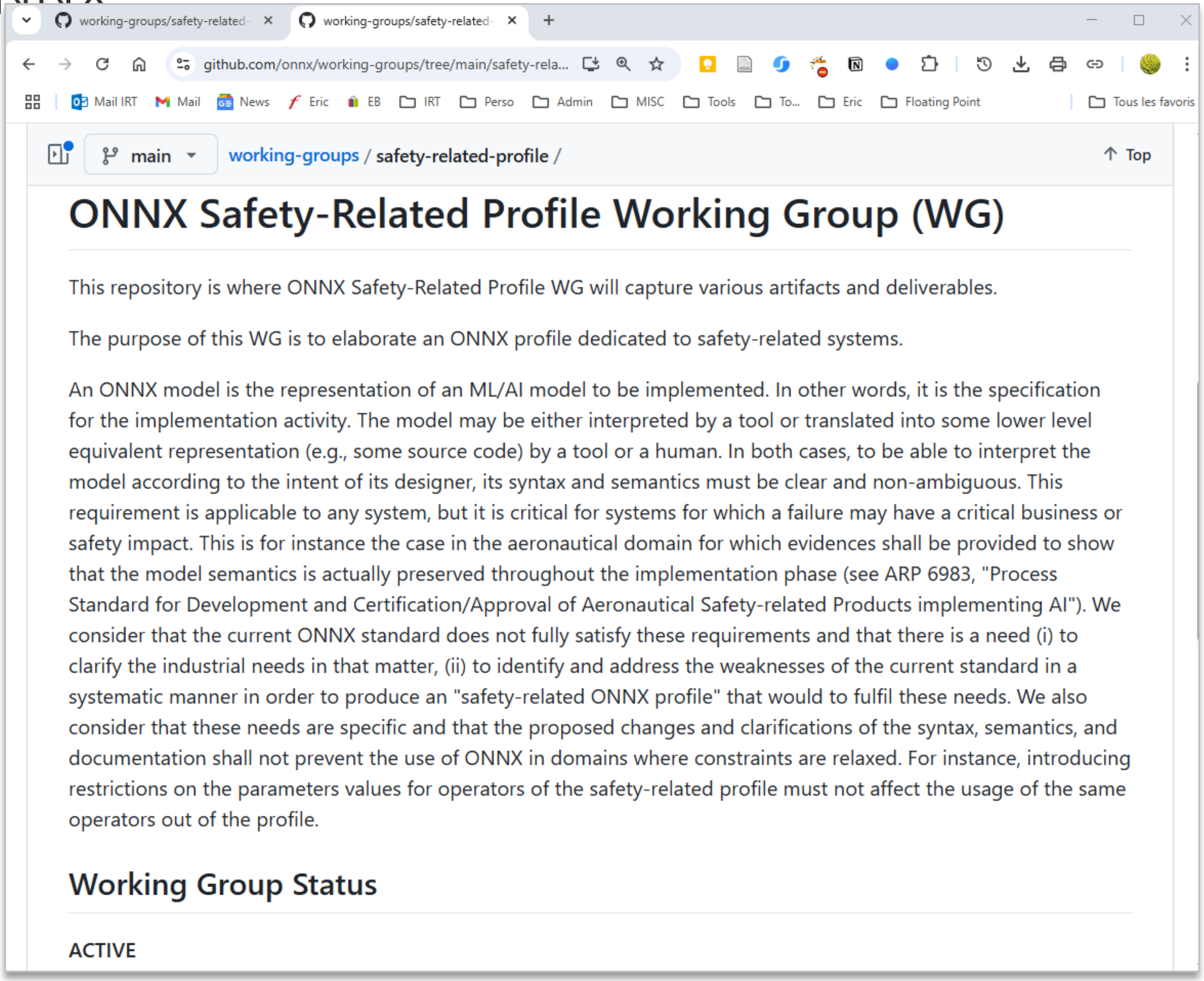
In red: regular participation
In black: at least 1 participation

SONNX

Work area

https://github.com/onnx/working-groups/tree/main/safety-related-profile

(current) https://github.com/ericjenn/working-groups/tree/ericjenn-srpwg-wg1/safety-related-profile


main working-groups / safety-related-profile /

ONNX Safety-Related Profile Working Group (WG)

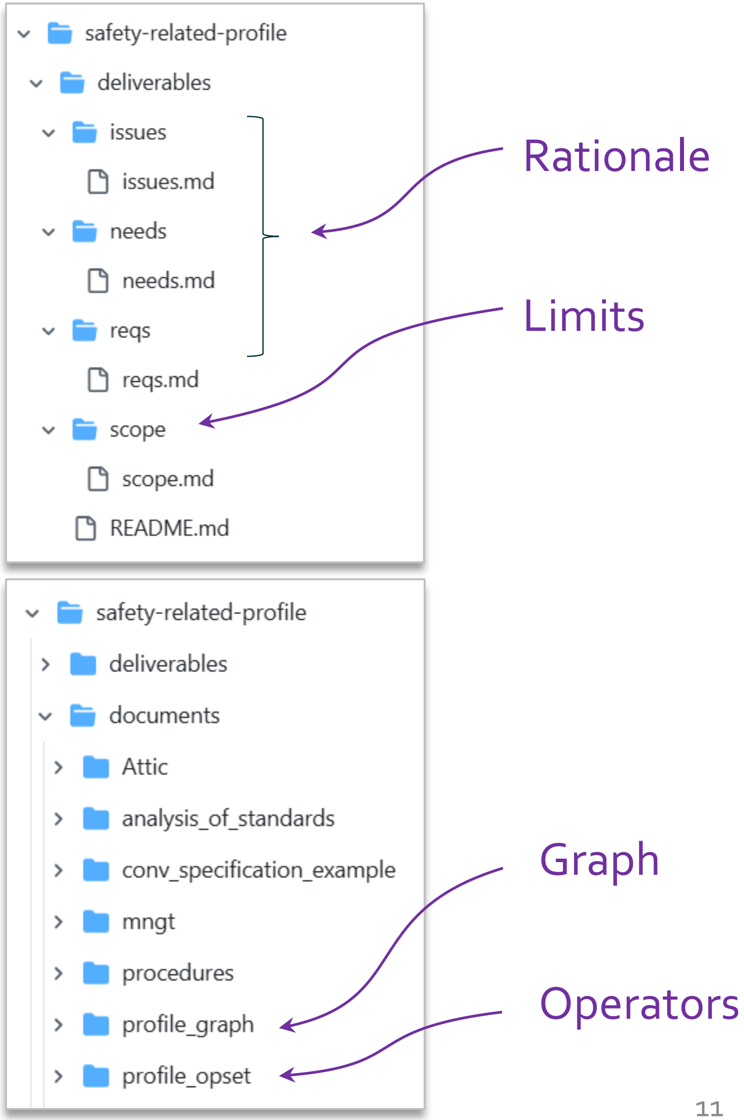
This repository is where ONNX Safety-Related Profile WG will capture various artifacts and deliverables.

The purpose of this WG is to elaborate an ONNX profile dedicated to safety-related systems.

An ONNX model is the representation of an ML/AI model to be implemented. In other words, it is the specification for the implementation activity. The model may be either interpreted by a tool or translated into some lower level equivalent representation (e.g., some source code) by a tool or a human. In both cases, to be able to interpret the model according to the intent of its designer, its syntax and semantics must be clear and non-ambiguous. This requirement is applicable to any system, but it is critical for systems for which a failure may have a critical business or safety impact. This is for instance the case in the aeronautical domain for which evidences shall be provided to show that the model semantics is actually preserved throughout the implementation phase (see ARP 6983, "Process Standard for Development and Certification/Approval of Aeronautical Safety-related Products implementing AI"). We consider that the current ONNX standard does not fully satisfy these requirements and that there is a need (i) to clarify the industrial needs in that matter, (ii) to identify and address the weaknesses of the current standard in a systematic manner in order to produce an "safety-related ONNX profile" that would to fulfil these needs. We also consider that these needs are specific and that the proposed changes and clarifications of the syntax, semantics, and documentation shall not prevent the use of ONNX in domains where constraints are relaxed. For instance, introducing restrictions on the parameters values for operators of the safety-related profile must not affect the usage of the same operators out of the profile.

Working Group Status

ACTIVE



Deliverables

(D1.a) Safety-related Profile **Scope** Definition (2024/11/01)

(D1.c) **Consolidated needs** for all industrial domains (2025/01/01)

(D2.a) ONNX safety-related Profile **requirements** (2025/02/01)

(D3.a) ONNX Safety-related profile – graph (2025/05/01)

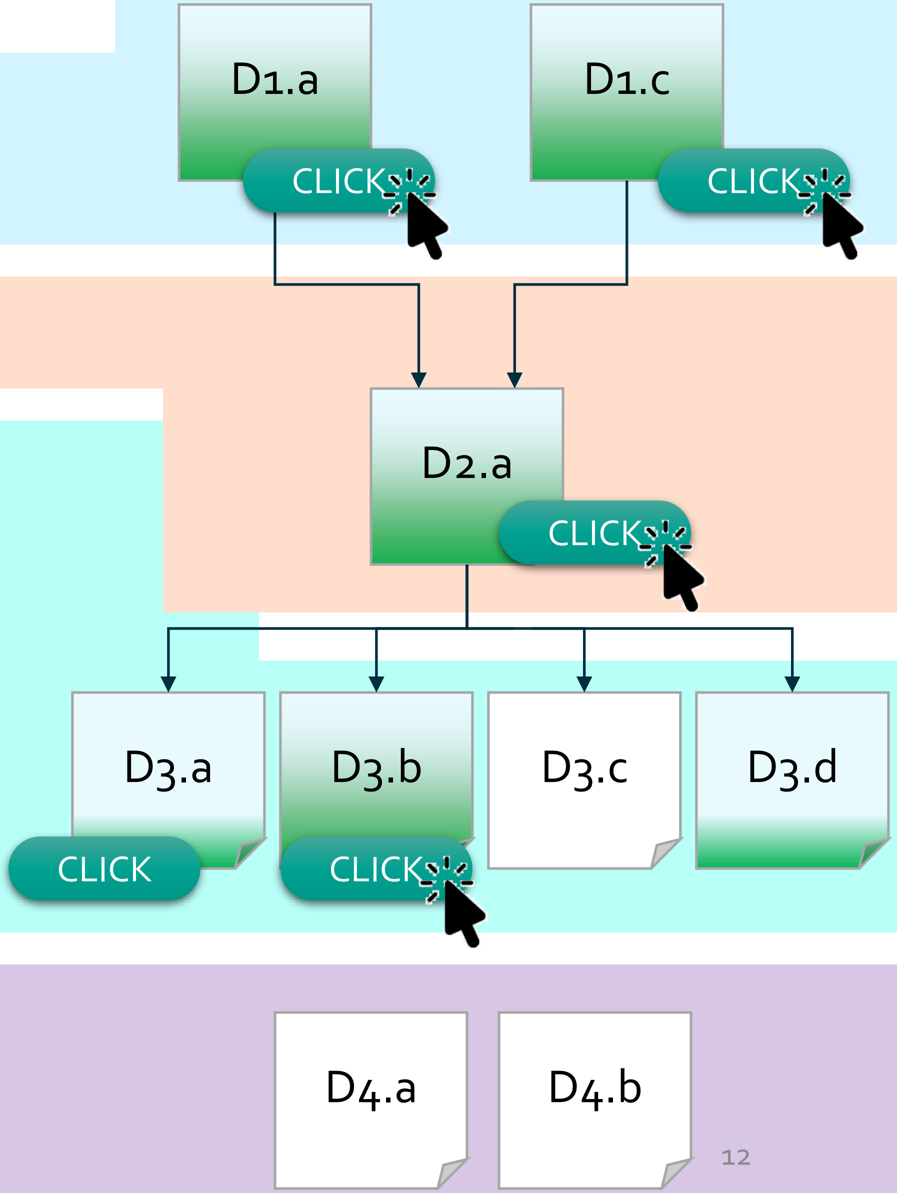
(D3.b) ONNX Safety-related profile – operators (2025/12/31)

(D3.c) ONNX Safety-related profile – format (2025/12/31)

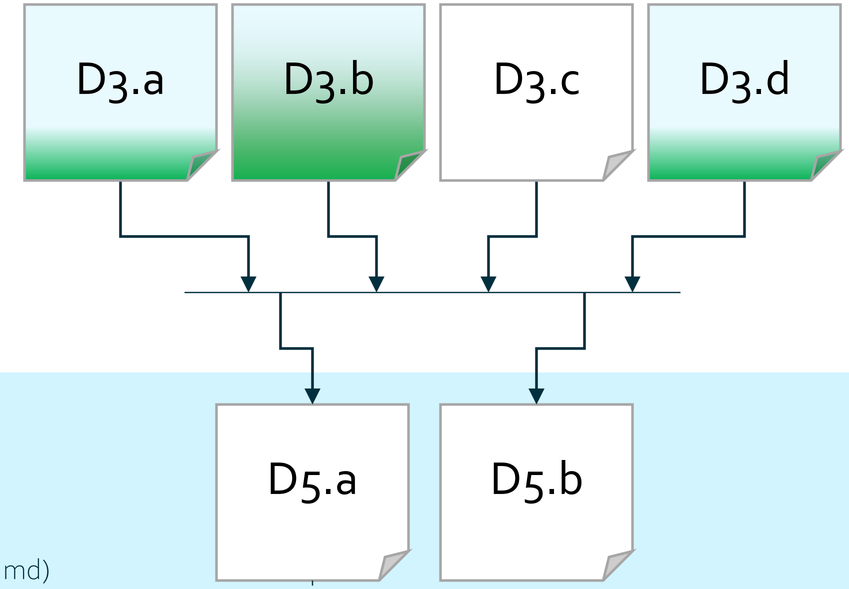
(D3.d) ONNX Safety-related profile reference implementation (2025/12/31)

(D4.a) ONNX Safety-related profile **verification** report

(D4.b) ONNX Safety-related profile **validation** report



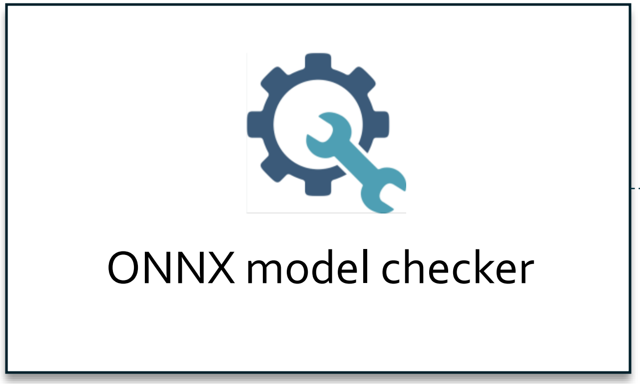
Deliverables
Status



(D5.a) Expression of the **needs** / tool list (2025/01/31)

(D5.b) **Requirements** of tool <tool>(2025/12/31)

(detailed WP is available at https://github.com/ericjenn/working-groups/blob/main/safety-related-profile/documents/sow.md)



Fine.

But is there anything to improve?

ONNX “issues”
ONNX failed conversion survey

- Are there empirical evidences of incompleteness, inconsistencies, etc.?
- Converters fail…
  - See Wenxin Jiang, Arav Tewari, et al, Interoperability in Deep Learning: A User Survey and Failure Analysis of ONNX Model Converters, Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1466–1478,Vien 2024
- … often with the bad mode…
- … but no root cause leading to the spec…

Finding 4. Location: Most failures are in *Node Conversion* (74%).
Finding 5. Symptom: The most common symptoms in DL model converters are *Crash* (56%) and *Wrong Model* (33%).
Finding 6. Causes: *Crashes* are largely due to *Incompatibilities* and *Type Problems*. *Wrong models* are largely due to *Type Problems* and *Algorithmic Errors*.

ONNX "issues"

ONNX github issues

- See discussion https://github.com/onnx/onnx/issues/3651
- See issues labelled **topic: spec clarification**

Describe the bug

When noop_with_empty_axes == 1 & axes is empty, in ONNX spec, it will return input tensor directly.
But in reference in onnx, it is mismatch. it returned np.square of the input tensor

```
Xavier Dupré, 13个月前 | 2 authors (Xavier Dupré and others)
class ReduceSumSquare_18(OpRunReduceNumpy):
    def _run(self, data, axes=None, keepdims=1, noop_with_empty_axes=0):  # type: ignore
        if self.is_axes_empty(axes) and noop_with_empty_axes != 0:  # type: ignore      liqun Fu, 17
            return (np.square(data),)

        axes = self.handle_axes(axes)
        keepdims = keepdims != 0  # type: ignore
```

This is complicated. Agree that there is a mismatch, but is the bug in the specification or implementation?

My personal interpretation is that this is a bug in the specification, not implementation, for the following reason: the attributes serve to define the set of axes being reduced: specifically, it is a flag to allow the empty list to indicate that all axes must be reduced (or that no axes must be reduced). Now, even if zero axes are reduced, it makes sense to compute the square. ReduceSumSquare is not actually a reduction-op: it is a reduction-op Sum applied to the square of the input.

Note: as of 2025, this issue has been corrected.

ONNX “issues”

ONNX github issues

- Rounding and numerical precision
  - `Cast` operator rounding (#3876, #5004)
    - No mention to truncation or rounding…
  - `DequantizeLinear` (#6132)
    - y = (x - x_zero_point) * x_scale , with x and x_zero_point with the same dtype. **What happens if x - x_zero_point is outside the range of dtype?**

  The onnxrutime and reference implementation behave differently.

- Operator semantics
  - `RandomNormal`, `RandomUniform` (# 6408)
    - The operator mentions a seed attribute, but doesn’t said anything about its behavior. If the operator is stateless, the same value will be generated each time it is called. If it is state full, it’ll generate different values, but according to the same sequence.
    - The onnxruntime and reference implementation behave differently.

ONNX "issues"

Laconic and lacunar documentation

Problem: *what is a convolution?*

Conv - 22

↑ Back to top

Summary

The convolution operator consumes an input tensor and a filter, and computes the output.

(Excerpt of ONNX doc.)

"Problem": *What is the value used for padding in a convolution?*

- Uhhh… zero?

ONNX “issues”

Default values

Problem: ONNX operators use attributes that have default values

Attributes

- **auto_pad - STRING** (default is `'NOTSET'`):

  auto_pad must be either NOTSET, SAME_UPPER, SAME_LOWER or VALID. Where default value is NOTSET, which means explicit padding is used.

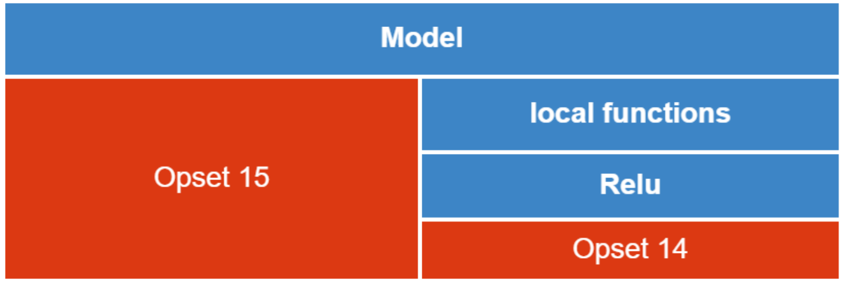
Conv operator

ONNX "issues"

Opset resolution, naming ambiguity

Problem: *ambiguity in opset resolution*

- An ONNX Function is a design artefact used to:
  1. define a composition of operators (ex: Relu Function is defined through Max Operator)
  2. define a composition of Nodes in the Graph as a reusable sub-graph (local function)
- Opsets are referenced in the Model element, and in each Function definition.
- Ex : Model import Opset v15,
  Model local function Relu import Opset v14.



- The Opset resolution is not specified:

// The (domain, name, overload) tuple must be unique across the function protos in this list.
// In case of any conflicts the behavior (whether the model local functions are given higher priority,
// or standard operator sets are given higher priotity or this is treated as error) is defined by
// the runtimes.

From onnx/onnx/onnx-ml.proto at main · onnx/onnx · GitHub , line 498-501

ONNX "issues"

Graph execution order

Problem: "ambiguity" in operator execution

- No functional ambiguity (the function is completely determined by the graph) but…
- …Operator are executed according to dataflow constraints, which determine a partial order…

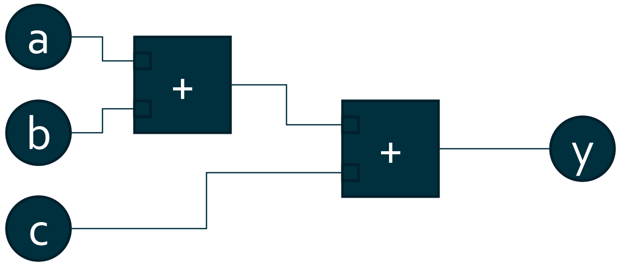
ONNX runtime

- Default execution order uses Graph::ReverseDFS() to generated topological sort
- Priority-based execution order uses Graph::KahnsTopologicalSort with per-node priority

- Note that there is no problem with associativity

$$y = a + b + c \stackrel{?}{=} (a + b) + c \stackrel{?}{=} a + (b + c)$$

Associativity is imposed by the graph



Non determinism and non reproducibility

- The different levels on “non determinism”

- Origin of non-determinism

Non determinism

Some factors

- Algorithmic factors
  - Non deterministic DL layers (e.g. dropout)
  - Weight initialization
  - Batch ordering

Controlable using seeds of RNGs.

- Representation of real numbers
  - When using float numbers, operations become non associative: $(a + b) + c \neq a + (b + c)$
- Effects of data parallelism
  - Use of SIMD acceleration (e.g., SSE or AVX on x86)
    - Impact on accuracy and aggregation order
- Effects of task parallelism
- Implementation of operations
  - E.g., convolution may be implemented
    - Naively as nested loops
    - Using im2col and GEMM
    - Using Winograd method
    - Using FFT

Optimizations

Structural & sparsity
- Pruning & dead-code elimination (graph-aware channel/filter removal)
- Structured sparsity (channel/filter/block/N:M) with sparse kernels
- Low-rank factorization (SVD/Tucker/CP) of Conv/MatMul
-Weight clustering / sharing

Algebraic & constant rewrites
- Constant folding & propagation
- Algebraic simplification (+0, *1, cancel reshapes/transposes)
- Common subexpression elimination (CSE)
- Broadcast/shape canonicalization

Fusion
- BatchNorm folding into Conv/Linear
- Bias/activation fusion (Conv/MatMul + Bias + Activation)
- Elementwise chain fusion (Add→Mul→Activation)
- Concat/Split sinking & merge

Layout & data movement
- Layout propagation/selection (NCHW↔NHWC)
- Transpose sinking/hoisting/merging
- Operator reordering to minimize movement

Quantization & precision
- PTQ/QAT graph rewrite (insert Q/DQ, calibration)
- Precision propagation (FP16/BF16/INT8 mixes)
- Dequant fusion into Conv/MatMul
- Clip/saturation canonicalization

Memory & scheduling
- Memory planning & buffer reuse (liveness, in-place, arenas)
- Operator scheduling to reduce peak memory
- Slice/concat planning & small-tensor coalescing

Control flow & loops
- Loop unrolling/peeling (static bounds)
- Loop-invariant code motion
- Dead-branch elimination (constant condition folding)
- Scan/While fusion (per-step elementwise fusion)

Domain-specific fusions
- CNNs: Conv-BN-Activation, DWConv-BN-Activation, Residual-Add fusion
- Transformers: QKV/attention block fusion, Gemm-Bias-GELU, RoPE/pos-enc fusion

Numerical stability rewrites
- Stable softmax (subtract max)
- Log/exp pairing rules
- Epsilon folding into norms

Partitioning & autotuning
- Subgraph partitioning/placement (delegate to TensorRT/NNAPI/etc.)
- Compile-/build-time autotuning hooks (tactic/algorithm selection)

Meta-optimizations
- Operator canonicalization to fusion-friendly normal form
- Shape inference & specialization (static fast paths)
- Duplicate constant pooling/deduplication

- BatchNorm folding into Conv/Linear

BEFORE

$$\mathrm{BN}(y_c) = \gamma_c \frac{y_c - \mu_c}{\sqrt{\sigma_c^2 + \varepsilon}} + \beta_c$$

$$y_c = W_c * x + b_c$$

AFTER

$$W'_c = s_c W_c, \qquad b'_c = s_c b_c + t_c$$

$$s_c = \frac{\gamma_c}{\sqrt{\sigma_c^2 + \varepsilon}}, \qquad t_c = \beta_c - s_c \mu_c$$

Optimizations

Structural & sparsity
- Pruning & dead-code elimination (graph-aware channel/filter removal)
- Structured sparsity (channel/filter/block/N:M) with sparse kernels
- Low-rank factorization (SVD/Tucker/CP) of Conv/MatMul
-Weight clustering / sharing

Algebraic & constant rewrites
- Constant folding & propagation
- Algebraic simplification (+0, *1, cancel reshapes/transposes)
- Common subexpression elimination (CSE)
- Broadcast/shape canonicalization

Fusion
- BatchNorm folding into Conv/Linear
- Bias/activation fusion (Conv/MatMul + Bias + Activation)
- Elementwise chain fusion (Add→Mul→Activation)
- Concat/Split sinking & merge

Layout & data movement
- Layout propagation/selection (NCHW↔NHWC)
- Transpose sinking/hoisting/merging
- Operator reordering to minimize movement

Quantization & precision
- PTQ/QAT graph rewrite (insert Q/DQ, calibration)
- Precision propagation (FP16/BF16/INT8 mixes)
- Dequant fusion into Conv/MatMul
- Clip/saturation canonicalization

Memory & scheduling
- Memory planning & buffer reuse (liveness, in-place, arenas)
- Operator scheduling to reduce peak memory
- Slice/concat planning & small-tensor coalescing

Control flow & loops
- Loop unrolling/peeling (static bounds)
- Loop-invariant code motion
- Dead-branch elimination (constant condition folding)
- Scan/While fusion (per-step elementwise fusion)

Domain-specific fusions
- CNNs: Conv-BN-Activation, DWConv-BN-Activation, Residual-Add fusion
- Transformers: QKV/attention block fusion, Gemm-Bias-GELU, RoPE/pos-enc fusion

Numerical stability rewrites
- Stable softmax (subtract max)
- Log/exp pairing rules
- Epsilon folding into norms

Partitioning & autotuning
- Subgraph partitioning/placement (delegate to TensorRT/NNAPI/etc.)
- Compile-/build-time autotuning hooks (tactic/algorithm selection)

Meta-optimizations
- Operator canonicalization to fusion-friendly normal form
- Shape inference & specialization (static fast paths)
- Duplicate constant pooling/deduplication

(chatGPT generated)

- Stable softmax (subtract max)

BEFORE

$$\mathrm{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

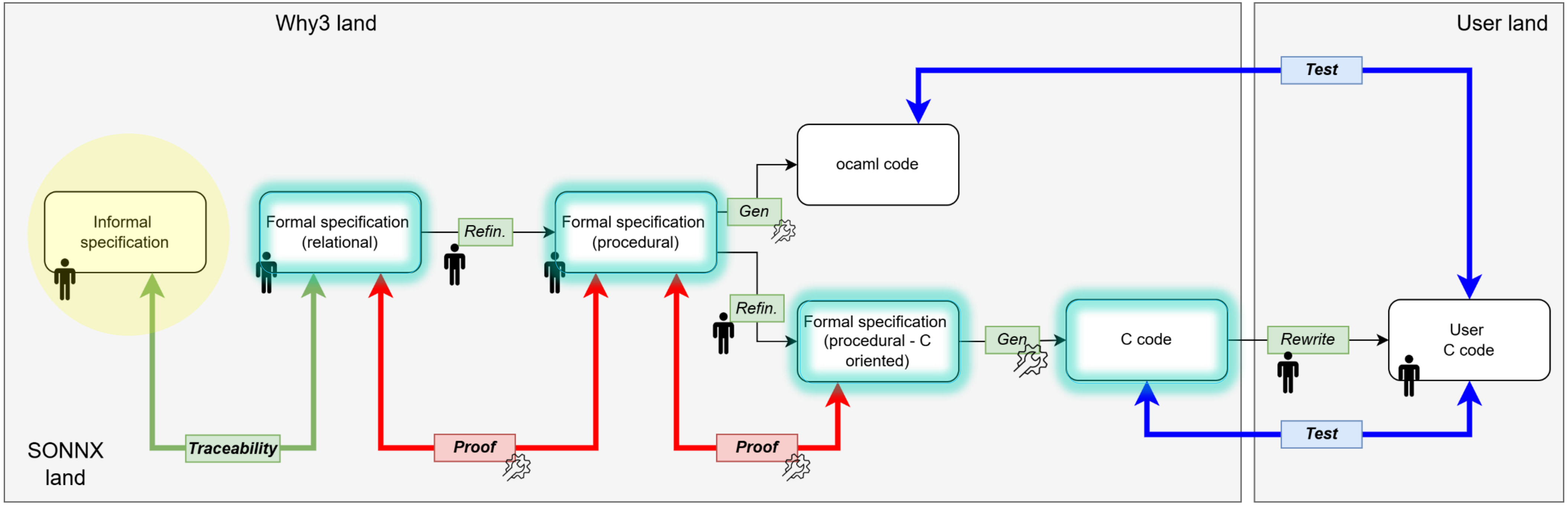
AFTER

$$\mathrm{softmax}(\mathbf{z})_i = \frac{e^{\,z_i - m}}{\sum_{j=1}^{n} e^{\,z_j - m}}$$

Back on track…

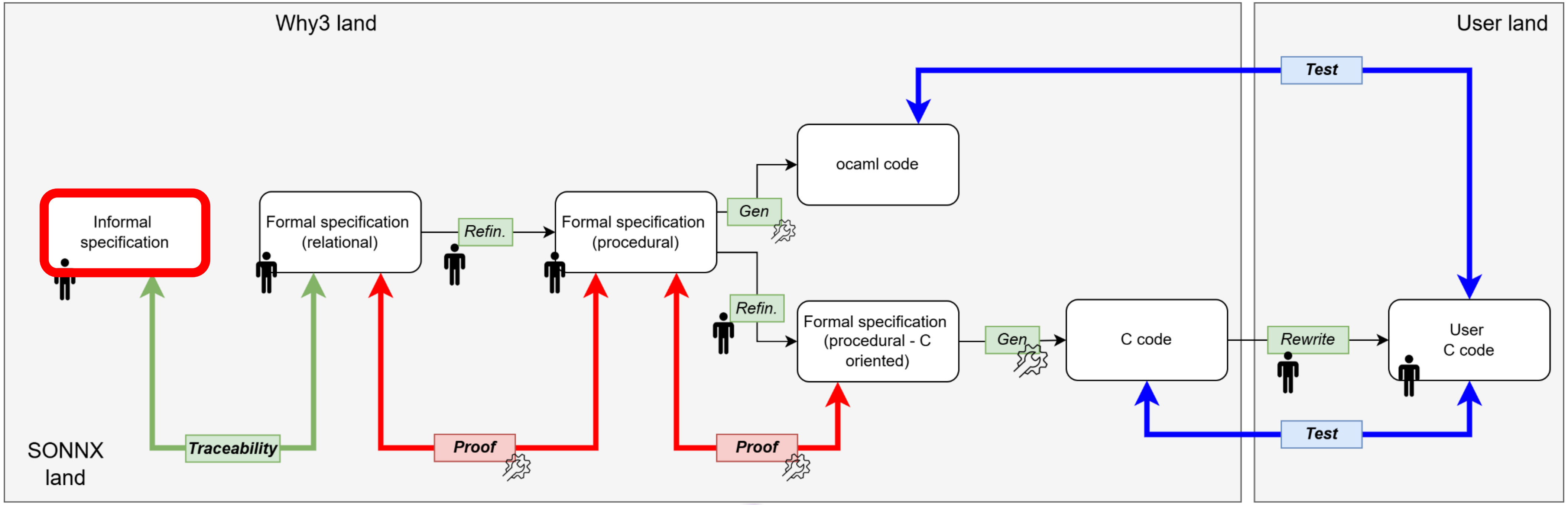
Formal specification and verification

Overview



Formal specification and verification

Information specification

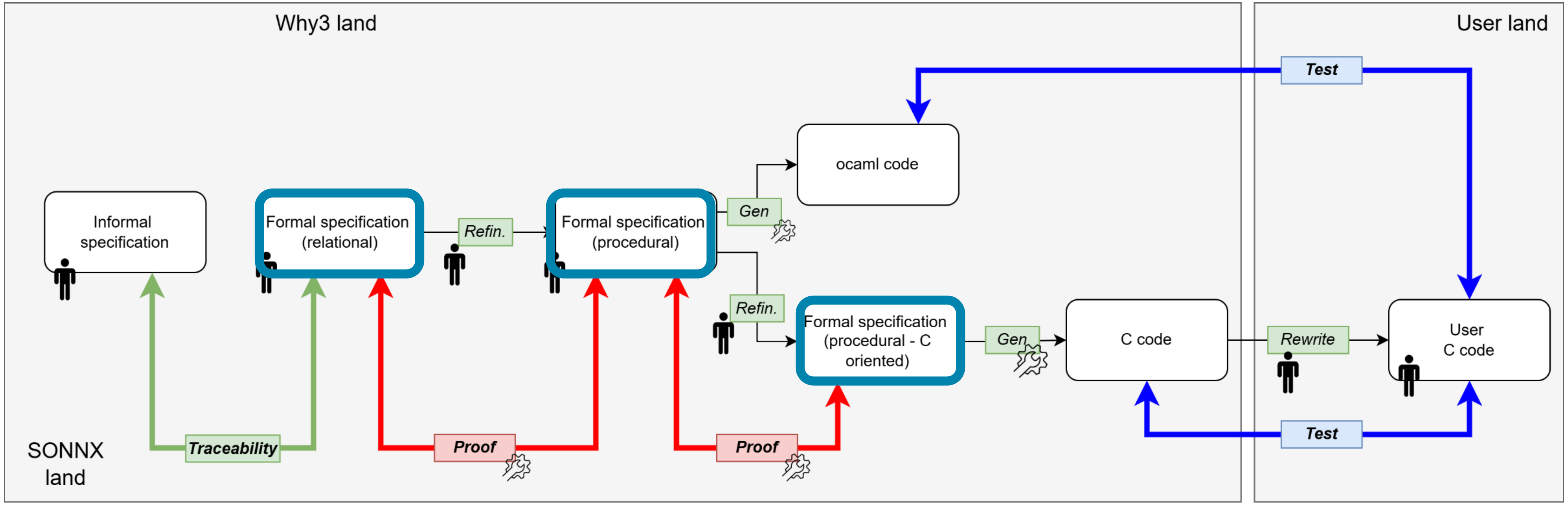


- Informal specification
  - Aka "documentation" / "user manual"

Formal specification and verification

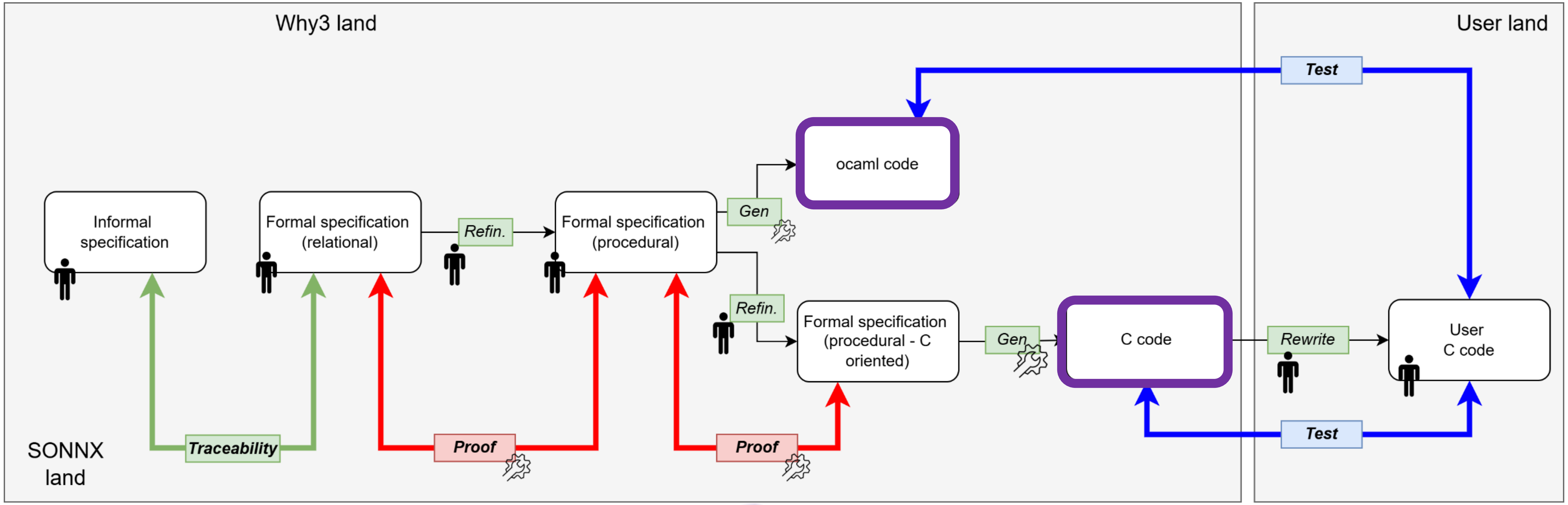
Formal specification

traceable



- Formal specification
  - Specification for developers (?)
  - Step towards the reference implementation

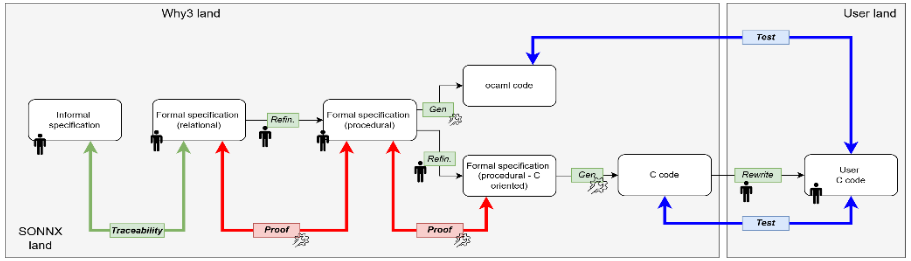
Formal specification and verification
Reference implementation

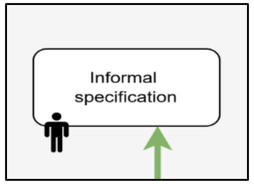


- Reference implementation
  - Interim ocaml code
  - Final C code

Formal specification and verification

The **conv** operator – Informal specification





`conv` operator

Contents

- `Convolution` operator for type real.

`Conv` `(real)`

Signature

`Y = conv(X,W,[B])` where

- `X` : input tensor
- `W` : convolution kernel
- `B` : optional bias
- `Y` : output tensor

Restrictions

The following restrictions apply to the `conv` operator for the SONNX profile:

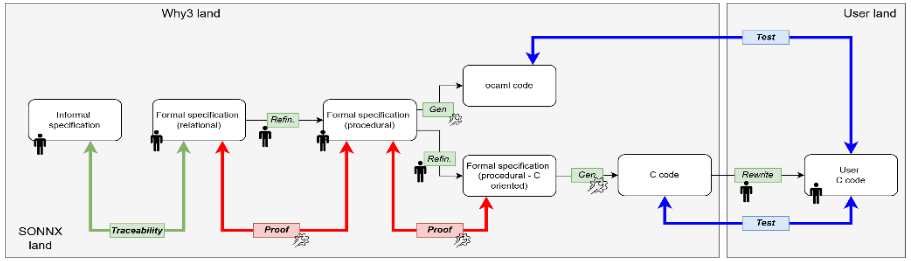
Restriction	Statement	Origin
`R1`	Input tensor `X` has 2 spatial axes	Transient
`R2`	Attribute `auto_pad` is set to `NOTSET`	No default values
`R3`	Attribute `group` is set to 1 (standard convolution) or to the number of channels of the input tensor `X` (depthwise convolution)	Transient

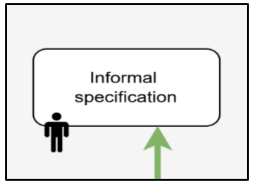
Simplification of the WG's work

Development assurance

Formal specification and verification

The **conv** operator – Informal specification





Informal specification

Operator `conv` computes the convolution of the input tensor `X` with the kernel `W` and adds bias `B` to the result. Two types of convolutions are supported: *standard convolution* and *depthwise convolution*.

Standard convolution

A *standard convolution* applies a kernel (also called "filter") to the input tensor, aggregating information accross both spatial axes and channels. For a given output channel, the kernel operates accross all input channels and all contributions are summed to produce the output. This corresponds to the case where attribute `group` = 1.

The mathematical definition of the operator is given hereafter:

$$Y[b,c,m,n] = \sum_{i=0}^{dW_1-1} \sum_{j=0}^{dW_2-1} \sum_{z=0}^{dW_3-1} (X_p[b,i,m \cdot \text{strides}[0]+j, n \cdot \text{strides}[1]+z] \cdot W_d[c,i,j,z]) + B_b[c]$$

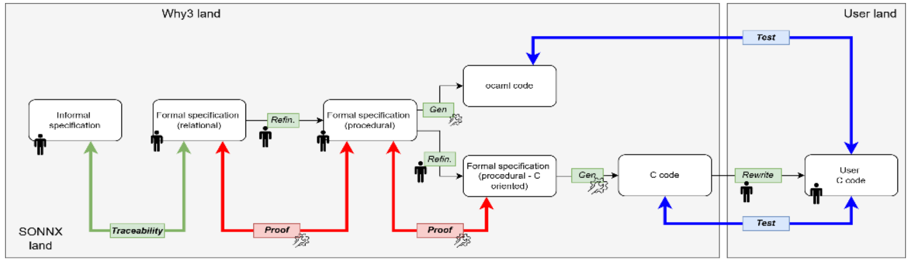
Simple, "naïve" formulation

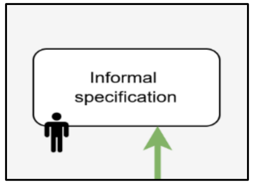
Where

- $b \in [0, dY_0 - 1]$ is the batch index. dY_0 is the batch size of output `Y`
- $c \in [0, dY_1 - 1]$ is the data channel. dY_1 is the number of data channels of output `Y`
- $m \in [0, dY_2 - 1]$ is the index of the first spatial axis of output `Y`
- $n \in [0, dY_3 - 1]$ is the index of the second spatial axis of output `Y`
- dW_1 is the number of feature maps of kernel `W`
- dW_2 is the size of the first spatial axis of kernel `W`
- dW_3 is the size of the second spatial axis of kernel `W`
- `strides` is an attribute of the operator. It will be described later in this section.
- $X_p = \text{pad}(X, pads)$ is the padded version of the input tensor `X`. Function pad applies zero-padding as specified by the pads attribute (see ONNX `Pad` operator).
- $W_d = \text{dilation}(W, dilations)$ is the dilated version of the kernel `W`. Function dilation expands the kernel by inserting spaces between its elements as specified by the dilations attribute. Its definition is given later.
- $B_b = \text{broadcast}(B, (dY_0, dY_1, dY_2, dY_3))$ is the broadcasted version of bias `B`. Function broadcast replicates the bias value across the spatial dimensions and batch dimension of the output `Y`. It takes as argument the bias `B` and the shape of output `Y`. Its definition is given later.

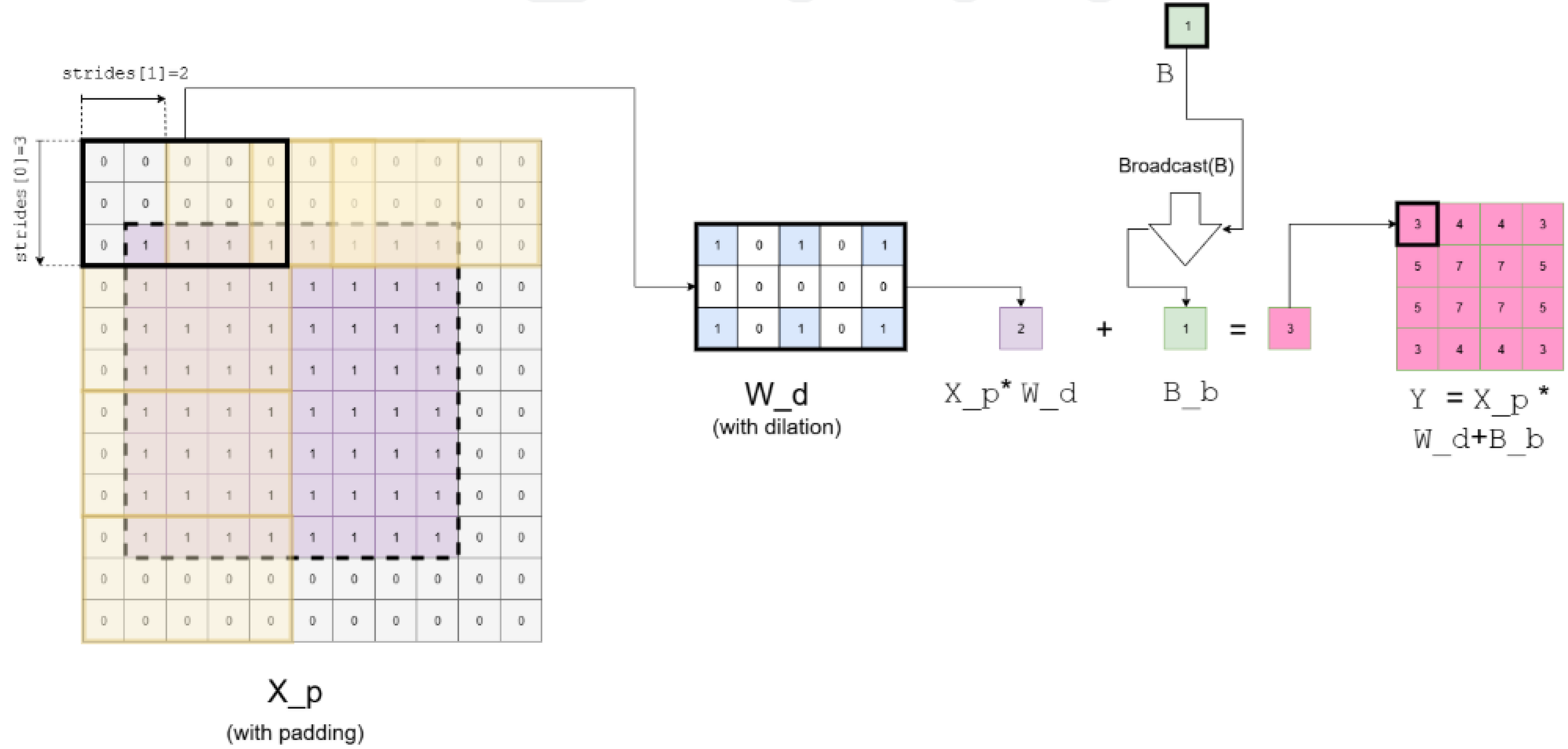
Formal specification and verification

The **conv** operator – Informal specification



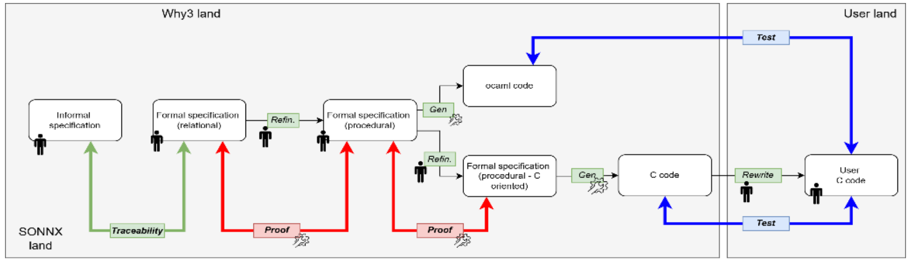


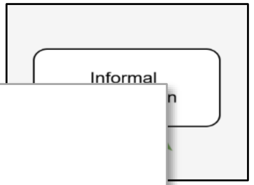
Finally, the following figure illustrates operator `Conv` applied on input `X` with kernel `W` and bias `B` :



Formal specification and verification

The **conv** operator – Informal specification





`X` : tensor of real

Tensor `X` is the input tensor on which convolution with kernel `W` is computed.

The sha…

- dX…
- dX…
- dX…

Constrai…

- `C1` : Number of spatial axes of tensor `X`
  - Statement: The number of spatial axes of tensor `X` is 2. `R1`
  - Rationale: This restriction is introduced to reduce the specification effort. It matches the industrial use cases considered in the profule.

Rationale: This restriction is introduced to reduce the specification effort. It matches the industrial use cases

- `C2`…
- `C3` : Consistency between the shape of tensors `X` , `W` , `Y` and attributes `pads` , `dilations` and `strides`
  - Statement:
    - $\left\lfloor \frac{alpha - ((dilations[0] \cdot dW_2 - 1) + 1)}{strides[0]} \right\rfloor + 1 = dY_2$ with $alpha = dX_2 + pads[0] + pads[2]$

    and

    - $\left\lfloor \frac{beta - ((dilations[1] \cdot dW_3 - 1) + 1)}{strides[1]} \right\rfloor + 1 = dY_3$ with $beta = dX_3 + pads[1] + pads[3]$
- `C4`…

Numerical errors
Approach

- Neural networks tend to be robust to numerical errors
  - Training on real data introduces much larger noise than rounding
  - Networks learn to operate under variability.
  - Non-exact optimization (stoch. grad. desc. , dropout,...) naturally "desensitizes" the model to small perturbations.
  - Continuous activations (ReLU, Sigmoid, etc.) ensure local smoothness almost everywhere.
  - Batch normalization and weight regularization improve conditioning and scale stability.

Numerical errors

Approach

- We propose a lower bound on the error for any value in the input domain, for any implementation complying with IEEE 754
  - This is not necessarily the smallest error but
    - The effort to express the formula remains acceptable
    - The complexity of the formula remains tractable
    - The verification of the property remains achievable
- For a restriction of the input domain, the error may be smaller
- The SONNX reference implementation will (?) comply with this constraint
- More efficient implementations may violate this constraint. In that case, the implementer has to provide its own precision requirement, following the structure of the provided formula.
- A tool will be used to demonstrate that the accuracy constraint is satisfied

Numerical errors
Example: the **add** operator

Numerical Accuracy

If tensor A_{err} is the numerical error of `A`, tensor B_{err} is the numerical error of `B`, let us consider $C_{\text{err}}^{\text{propag}}$ the propagated error of `Add` and $C_{\text{err}}^{\text{intro}}$ the introduced error of `Add`. Hence the numerical error of `C`, $C_{\text{err}} = C_{\text{err}}^{\text{propag}} + C_{\text{err}}^{\text{intro}}$.

Error propagation

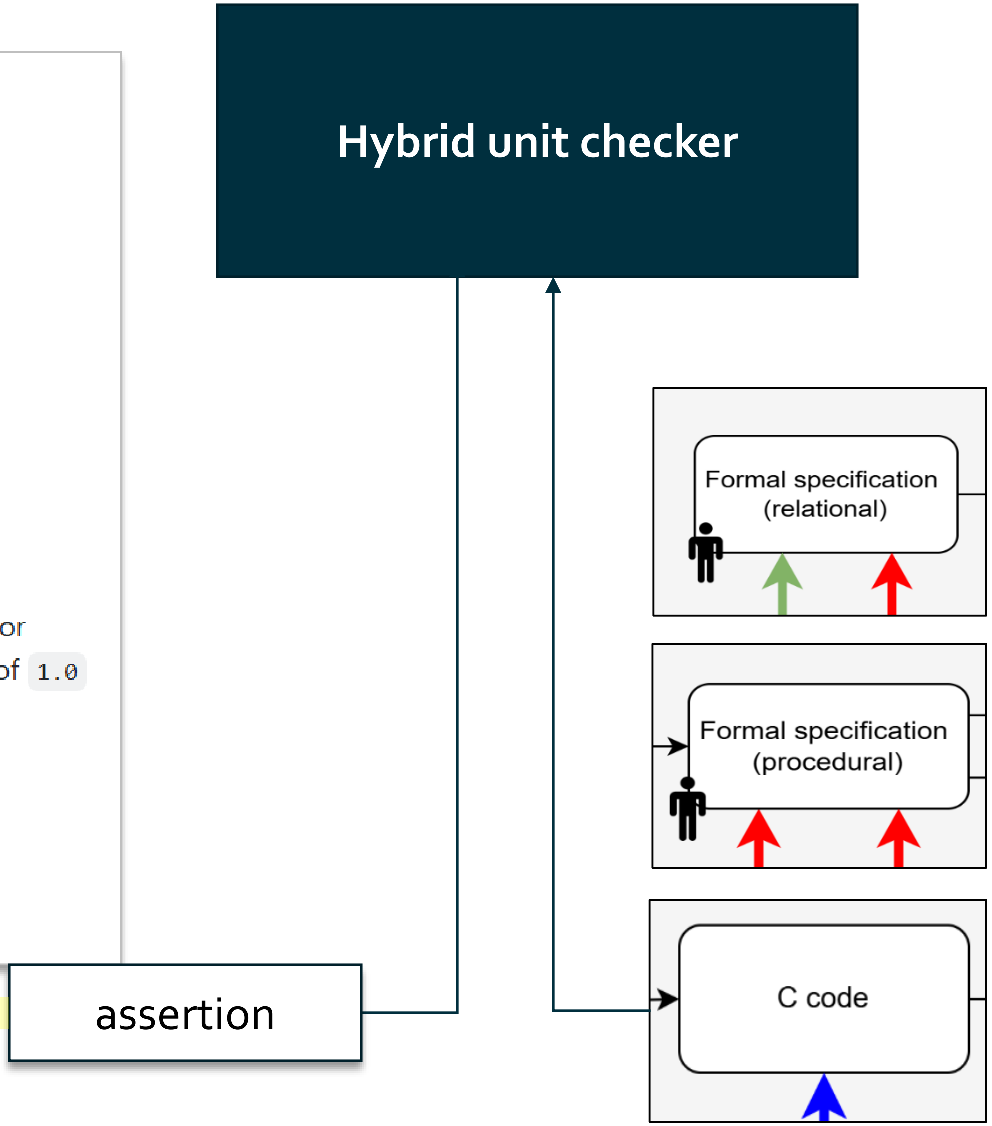
For every indexes $I = (i_0, i_1, \ldots, i_n)$ over the axes,

- $C_{\text{err}}^{\text{propag}}[I] = A_{\text{err}}[I] + B_{\text{err}}[I]$

Error introduction - floating-point IEEE-754 implementation

The error introduced by the `Add` operator shall be bound by the semi-ulp of the addition result for every tensor component for a normalized result. For a hardware providing m bits for floating-point mantissa, the semi-ulp of `1.0` is $2^{-(m+1)}$. Hence, for every indexes $I = (i_0, i_1, \ldots, i_n)$ over the axes,

- $\left|C_{\text{err}}^{\text{intro}}[I]\right| \leq \max\left(\left|A[I] + B[I] + A_{\text{err}}[I] + B_{\text{err}}[I]\right| \times 2^{-(m+1)}, \frac{\text{denorm-min}}{2}\right)$
- $\left|C_{\text{err}}^{\text{intro}}[I]\right| \leq \max\left(\left|A_{\text{float}}[I] + B_{\text{float}}[I]\right| \times 2^{-(m+1)}, \frac{\text{denorm-min}}{2}\right)$
- $\left|C_{\text{err}}^{\text{intro}}[I]\right| \leq \max\left(\left|A[I] + B[I]\right| \times \frac{2^{-(m+1)}}{1 - 2^{-(m+1)}}, \frac{\text{denorm-min}}{2}\right)$

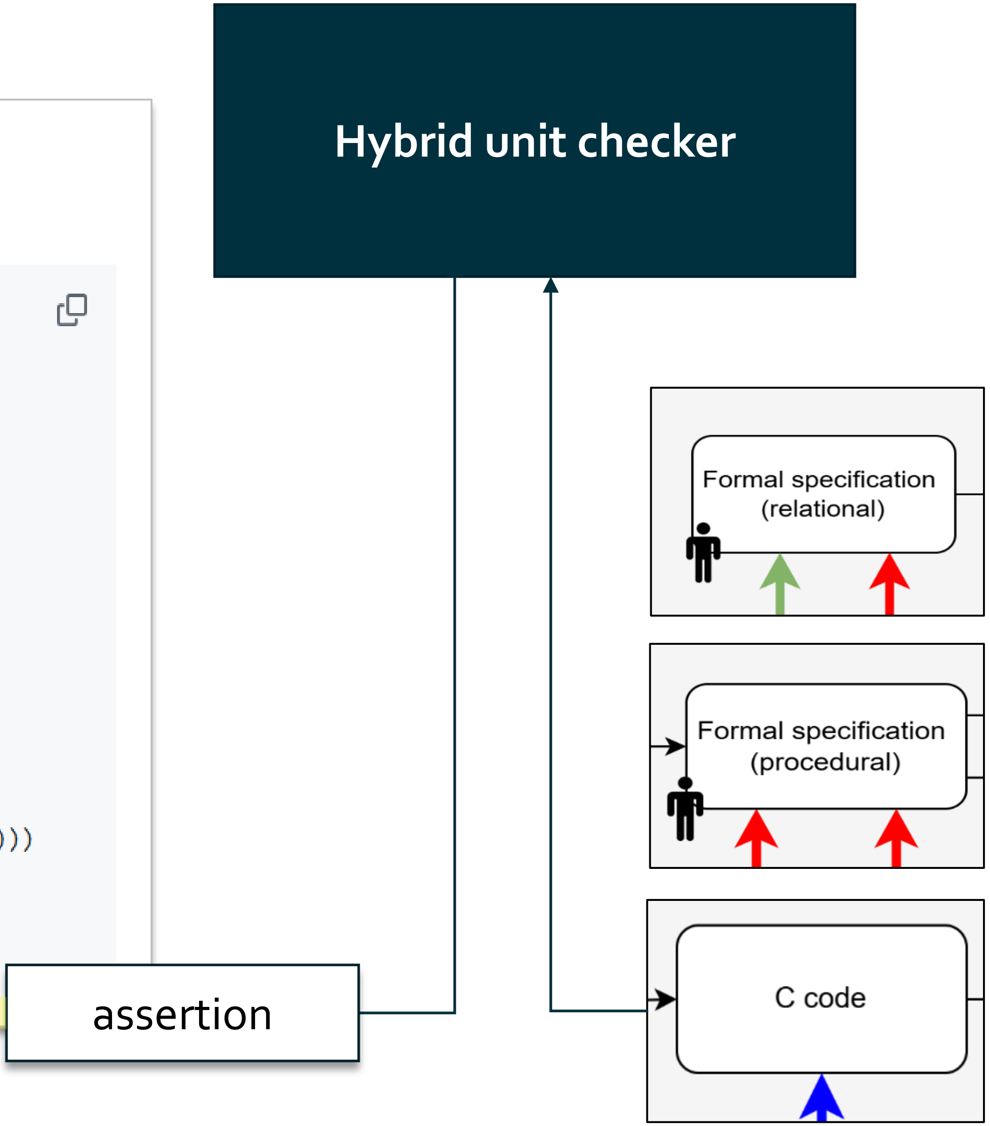


Numerical errors
Example: the **add** operator

Unit verification - floating-point IEEE-754 implementation

A symbolic inference of the error over the tensor components should ensure the above properties.

```
Tensor<SymbolicDomainError> A, B;

/* A and B symbolic initialization */

template <typename TypeFloat>
std::function<TypeFloat (decltype(A.indexes()))>
  result = [&A, &B](decltype(A.indexes()) list_of_indexes)
    { return A[list_of_indexes] + B[list_of_indexes]; }

for (auto i : A.indexes()) {
   SymbolicDomainError a = A[i];
   SymbolicDomainError b = B[i];
   SymbolicDomainError c = result(i);
   assert(std::abs(c.err - a.err - b.err) <= std::max(std::abs(a.float + b.float)*(pow(2.0LD, -(m+1))),
        (real) std::numeric_limits<decltype(a.float)>::denorm_min() / 2.0);
}
```



Numerical errors
Bibliography

- Error evaluation is done using the fldlib developed by Franck Vevrine (CEA LIST) available at https://github.com/fvedrine/fldlib .
- Papers

  - [1] F. Védrine, M. Jacquemin, N. Kosmatov, and J. Signoles, 'Runtime Abstract Interpretation for Numerical Accuracy and Robustness', in *Verification, Model Checking, and Abstract Interpretation*, vol. 12597, F. Henglein, S. Shoham, and Y. Vizel, Eds, in Lecture Notes in Computer Science, vol. 12597. , Cham: Springer International Publishing, 2021, pp. 243–266. doi: 10.1007/978-3-030-67067-2_12.
  - [2] G. Boussu, N. Kosmatov, and F. Védrine, 'A Case Study on Numerical Analysis of a Path Computation Algorithm', *Electron. Proc. Theor. Comput. Sci.*, vol. 411, pp. 126–142, Nov. 2024, doi: 10.4204/EPTCS.411.8., available at https://arxiv.org/pdf/2411.14372

Failure conditions

- How to specify error conditions
- Examples

```
y=Add(a: int32, b: int32)
```

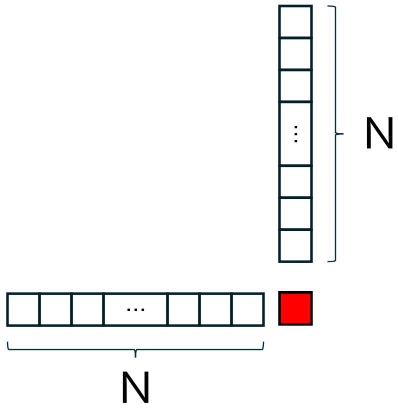
$$-2^{32} \leq \ a + b \leq 2^{32} - 1$$

Or, more conservatively,

$$-2^{31} + 1 \leq \ a < 2^{31} \text{ and } -2^{31} + 1 \leq b < 2^{31}$$

- For matrix multiplcation (e.g., **`MatMulInteger`**), a precondition can be expressed on the shape of the tensors

$$\mathrm{N} > \frac{2^{32} - 1}{128^2} \approx \ 133141.5$$



Failure conditions
The add operator

ONNX

Add (real, real)

$$Y[i] = A[i] + B[i]$$

Add (float, float)

$$Y[i] = A[i] + B[i]$$

Add (int, int)

For unsigned values (type `UINTn`):

$$Y[i] = \begin{cases} A[i] + B[i] - k.2^n & \text{if } A[i] + B[i] > 2^n - 1 \\ A[i] + B[i] & \text{otherwise} \end{cases}$$

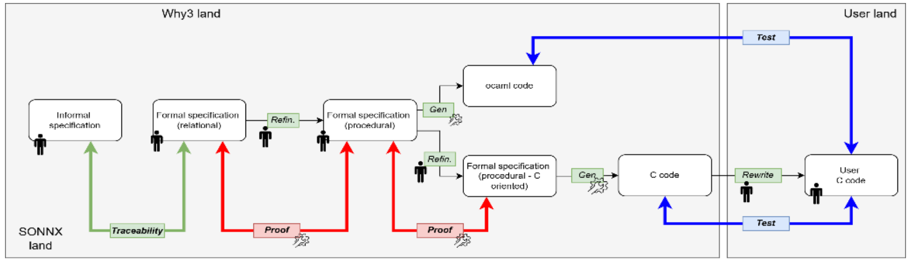
For signed values (type `INTn`):

$$Y[i] = \begin{cases} A[i] + B[i] - k_1.2^n & \text{if } A[i] + B[i] > 2^{n-1} - 1 \\ A[i] + B[i] + k_2.2^n & \text{if } A[i] + B[i] < -2^{n-1} \\ A[i] + B[i] & \text{otherwise} \end{cases}$$

To be checked…

Formal specification and verification

From informal to implementation



Informal specification

Review

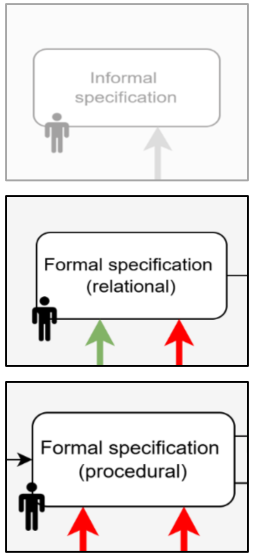
Formal specification (level 1)

(Automatic) Proof

Formal specification (level 2)

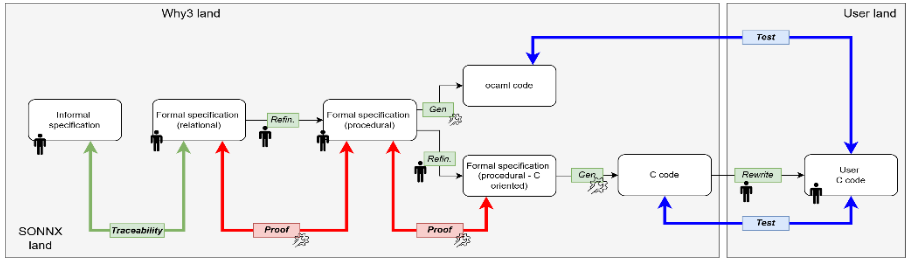
(Automatic) generation

C Code



Formal specification and verification

The **conv** operator – Formal specification: abstract **tensors**

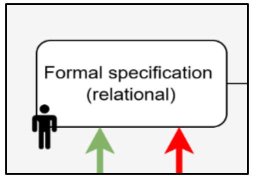


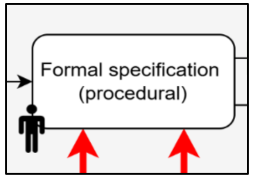

```
(** Formalization of Tensor *)
module Tensor
  use int.Int
  use map.Map
  use list.List
  use Range

  type data 'a = map (list int) 'a

  type tensor 'a = {
    dims : list int ;
    data : data 'a ;
    background : 'a ; (* default value, or value for 0-dimensions tensor *)
  }
```

```
(** Constant Tensor *)
let ghost function const (v : 'a) (bg : 'a) (ds : list int): tensor 'a
  ensures { result.dims = ds }
  requires { positive ds }
  ensures { result.background = bg }
  ensures { forall k. valid k ds -> result k = v }
  (*proof*)
  = { dims = ds ; data = pure { fun k -> if valid k ds then v else bg } ; background = bg }
  (*qed*)

(** Constant & Null *)
goal zero_is_const: forall e : 'a, ds. positive ds -> zero e ds == const e e ds

end
```


```
type tensor 'a = {
  dims : list int ;
  data : data 'a ;
  background : 'a ; (* default value, or value for 0-dimensions tensor *)
}
```

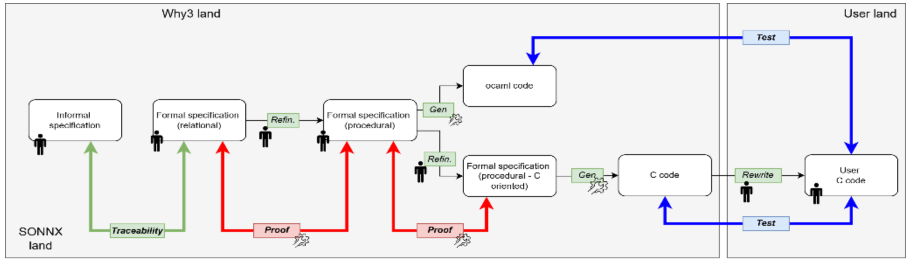

A abstract map from an index to a value





Formal specification and verification

The **conv** operator – Formal specification: concrete **tensors**




```
type farray = ptr float

type ctensor = {
  t_rank : int32 ;
  t_dims : iarray ;
  t_data : farray ;
}

function tensor_dim (t : ctensor) : list int
function tensor_size (t : ctensor) : int = vo
predicate valid_index (k : list int) (t : ctensor) = valid k (tensor_dim t)
predicate empty_tensor (t : ctensor) = t.t_r

predicate valid_tensor (t : ctensor) =
  dimension t.t_dims t.t_rank /\
  valid_range t.t_data 0 (tensor_size t) /\
  writable t.t_data

 let ctensor_add (a b r : ctensor)
     requires { valid_tensor a }
     requires { valid_tensor b }
     requires { valid_tensor r }
     requires { tensor a ~= tensor b ~= tensor r }
     ensures { tensor r = opadd (tensor a) (tensor b) }
```

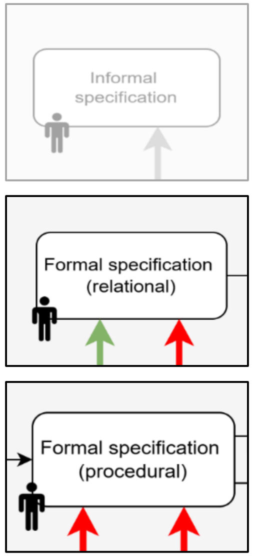

```
type ctensor = {
  t_rank : int32 ;
  t_dims : iarray ;
  t_data : farray ;
}
```


A concrete array containing the values


```
let ctensor_add (a b r : ctensor) =
    requires { valid_tensor a }
    requires { valid_tensor b }
    requires { valid_tensor r }
    requires { tensor a ~= tensor b ~= tensor r }
    ensures { tensor r = opadd (tensor a) (tensor b) }
```

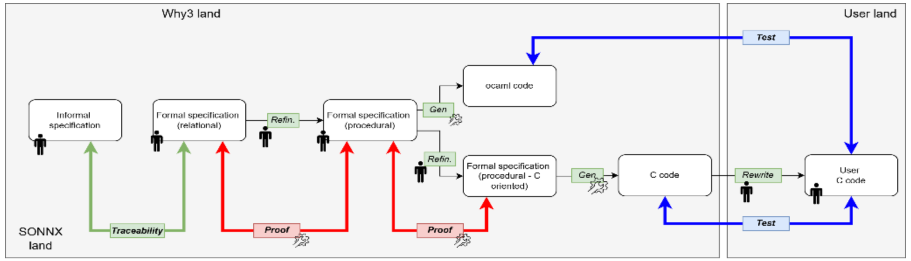

The concrete addition

The abstract addition



Formal specification and verification

The **conv** operator – Formal specification: the operation

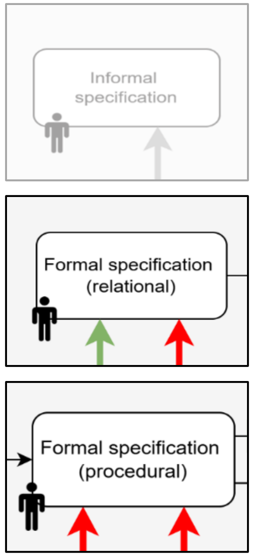



```
let function conv2d_int (x: tensor int) (w: tensor int) (b: option (tensor int))
                         (strides pads dilations: seq int)
                         (group_val: int)
                         (auto_pad_is_not_set: bool)
                         : tensor int

  Ops4D.n_dim w > 0 }
  requires { Ops4D.n_dim x > 0 }

  (* --- Attribute Sequence Length Requirements --- *)
  requires { Seq.length strides = 2 }
  requires { Seq.length pads = 4 }
  requires { Seq.length dilations = 2 }

     ONNX Profile Restrictions ---
  requires { group_val = 1 }
  requires { auto_pad_is_not_set }

  (* --- Conditional Bias Tensor Constraints --- *)
  requires { match b with
             | None -> true
             | Some b_tensor ->
               dim b_tensor = 1
           end
         }
```

- `C1` : Number of spatial axes of tensor `X`
  - Statement: The number of spatial axes of tensor `X` is 2. `R1`
  - Rationale: This restriction is introduced to reduce the specifica considered in the profule.

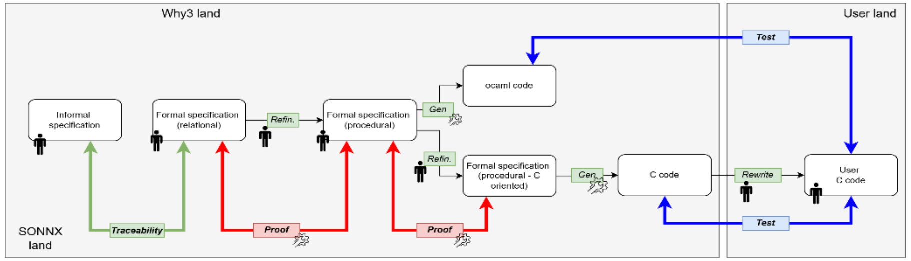
The formal expression of constraints

The informal expression of constraints



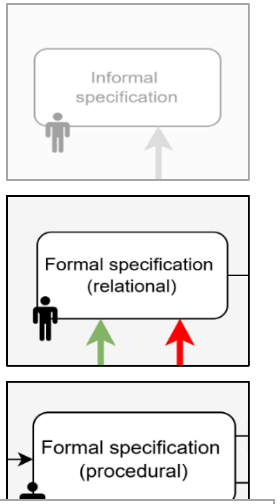
Formal specification and verification

The **conv** operator – Formal specification: the operation




```
let function conv2d_int (x: tensor int) (w: tensor int) (b: option (tensor int))
                        (strides pads dilations: seq int)
                        (group_val: int)
                        (auto_pad_is_not_set: bool)
                        : tensor int
```

```
  =
    let res_shape = conv2d_output_shape x w strides pads dilations in
    let res_value_func = conv2d_output_value x w b strides pads dilations res_shape in
    { shape = res_shape; value = res_value_func }

end
```

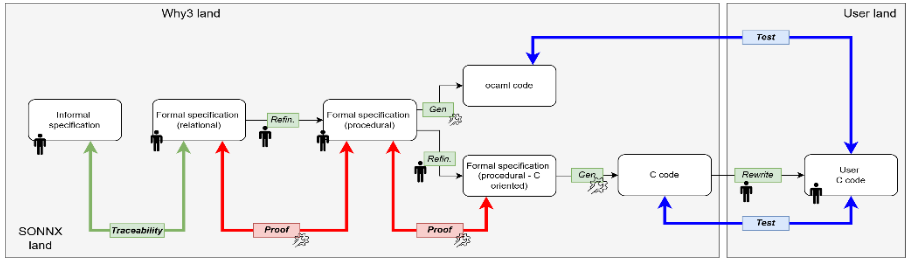



```
let function conv2d_output_shape (x w: tensor int)
                                 (strides pads dilations: seq int) : Shape.shape
```

```
let function conv2d_output_value (x w: tensor int) (b: option (tensor int))
                                 (strides pads dilations: seq int)
                                 (out_shape_param: Shape.shape)
                                 : (Index.index -> int)
```

Formal specification and verification

The **conv** operator – Formal specification: the operation



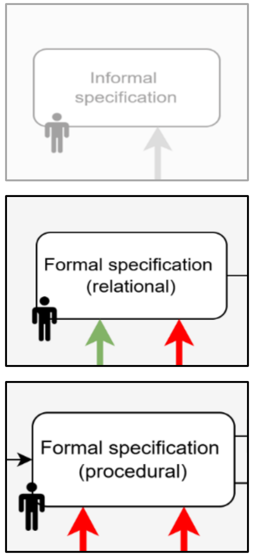

```
let function conv2d_output_value (x w: tensor int) (b: option (tensor int))
                                 (strides pads dilations: seq int)
                                 (out_shape_param: Shape.shape)
                                 : (Index.index -> int)
```

The formal specification of the operation

```
let rec function sum_over_c_in (x_tensor w_tensor: tensor int)
                               (pads_attr dilations_attr strides_attr: seq int)
                               (n_for_x c_out_for_w h_out_val w_out_val c_in_iter_for_both: int) : int
```

```
let rec function sum_over_kh (x_tensor w_tensor: tensor int)
                             (pads_attr dilations_attr strides_attr: seq int)
                             (n_for_x c_out_for_w c_in_for_both h_out_val w_out_val kh_iter_for_w: int) : int
```

```
let rec function sum_over_kw (x_tensor w_tensor: tensor int)
                             (pads_attr dilations_attr strides_attr: seq int)
                             (n c_out c_in h_out w_out kh kw_iter: int) : int
```



The informal specification of the operation

$$Y[b,c,m,n] = \sum_{i=0}^{dW_1-1}\sum_{j=0}^{dW_2-1}\sum_{z=0}^{dW_3-1}\left(X_p[b,i,m\cdot \mathrm{strides}[0]+j,n\cdot \mathrm{strides}[1]+z]\cdot W_d[c,i,j,z]\right) + B_b[c]$$

Formal specification and verification

The **conv** operator – Formal specification: the operation

```
let ctensor_conv2d (x : ctensor) (w : ctensor) (b : ctensor)
                   (pad_top pad_bottom pad_left pad_right : int32)
                   (dil_h dil_w : int32) (str_h str_w : int32)
                   (output : ctensor): unit  =
```

```
(* Postcondition: The concrete tensor 'output' must be equal to the ghost specification of Conv2D. *)
   ensures { tensor output = OPConv2d.opconv2d (tensor x) (tensor w) (tensor b)
                                                (Int32.to_int pad_top) (Int32.to_int pad_bottom)
                                                (Int32.to_int pad_left) (Int32.to_int pad_right)
                                                (Int32.to_int dil_h) (Int32.to_int dil_w)
                                                (Int32.to_int str_h) (Int32.to_int str_w) }
```

```
for n = 0 to n_batches - 1 do (* Loop over Batch dimension (N) *)
    invariant { true }
    for m = 0 to m_out - 1 do (* Loop over Output Channels (M) *)
        invariant { true }
        for oh = 0 to h_out - 1 do (* Loop over Output Height (oH) *)
            invariant { true }
            for ow = 0 to w_out - 1 do (* Loop over Output Width (oW) *)
                invariant { true }
                (* Initialisation de l'accumulateur pour le produit scalaire (dot product) *)
                let conv_sum = ref (f64 0.0) in
                                        .../...
```

The concrete specification of the operation

Formal specification and verification

The **conv** operator – Code generation

```
void ctensor_conv2d(struct ctensor x, struct ctensor w, struct ctensor b,
                    int32_t pad_top, int32_t pad_bottom, int32_t pad_left,
                    int32_t pad_right, int32_t dil_h, int32_t dil_w,
                    int32_t str_h, int32_t str_w, struct ctensor output) {
  int32_t n_batches, c_in, h_in, w_in, m_out, kh, kw, h_out, w_out,
          pad_top_i, pad_left_i, dil_h_i, dil_w_i, str_h_i, str_w_i;
  int32_t* x_coords;
  int32_t* w_coords;
  int32_t* b_coords;
  int32_t* output_coords;
  int32_t n, o, m, o1, oh, o2, ow, o3, c, o4, k_h, o5, k_w, o6, ih, iw,
```

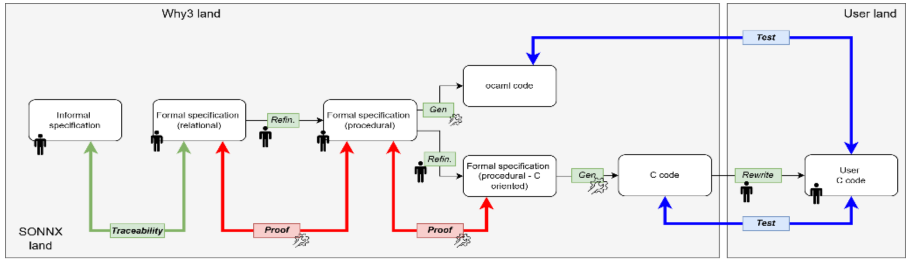
```
  if (x_coords && (w_coords && (b_coords && output_coords))) {
    o = n_batches - 1;
    if (0 <= o) {
      for (n = 0; ; ++n) {
        o1 = m_out - 1;
        if (0 <= o1) {
          for (m = 0; ; ++m) {
            o2 = h_out - 1;
            if (0 <= o2) {
              for (oh = 0; ; ++oh) {
                o3 = w_out - 1;
                if (0 <= o3) {
                  for (ow = 0; ; ++ow) {
                    conv_sum = ((double) 0.0);
```

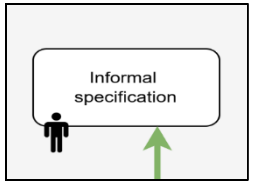
Formal specification and verification
Bibliography

- Formal specification is done using the WhyML language
- Formal verification is done using the Why3 and Why3find toolset
  - Why3: https://www.why3.org
  - Why3find: https://git.frama-c.com/pub/why3find
    - The installation of Why3Find is described at https://github.com/ericjenn/working-groups/blob/ericjenn-srpwg-wg1/safety-related-profile/tools/why3_faq.md
  - Papers:

    - [1] J.-C. Filliâtre and A. Paskevich, 'Why3 — Where Programs Meet Provers', in *Programming Languages and Systems*, vol. 7792, M. Felleisen and P. Gardner, Eds, in Lecture Notes in Computer Science, vol. 7792. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 125–128. doi: 10.1007/978-3-642-37036-6_8. Available at https://inria.hal.science/hal-00789533
    - [2] Loïc Correnson. Packaging proofs with Why3find. 35es Journées Francophones des Langages Applicatifs (JFLA 2024), Jan 2024, Saint-Jacut-de-la-Mer, France., available at https://cea.hal.science/hal-04407129v1

Formal specification and verification
Other cases: the **`graph`**





Graph

- `[T01a]` A graph contains a set of nodes
- `[T01b]` A graph contains a set of tensors that are inputs and outputs of the nodes
  - Some of those tensors are inputs (resp. outputs) of the graph, i.e, their values are set (resp. returned) before (resp. after) executing the graph

Nodes

- `[T03a]` A node refers to an operator
  - An operator may be referred to by multiple nodes
- `[T03b]` There is a 1-to-1 mapping between the set of inputs and outputs of a node and the set of inputs and outputs of its associated operator `[R1]`.
  - Note that is is a restriction with respect to the ONNX standard that allows fewer inputs or outputs when the omitted input or output is optional.

Tensors

- `[T02b]` A tensor is an object that can hold a value or be uninitialized
- `[T02a]` A tensor is identified by a unique identifier within a graph

Operators

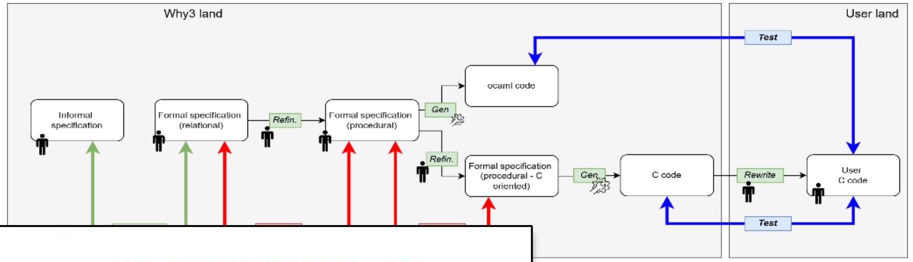
- `[T04a]` An operator specifies a relation (a function) between a set of input parameters and a set of outputs parameters.
  - Input and output parameters (resp. output) are free variables that can be bound to tensors using nodes
  - An operator has at least one output

Execution Semantics

- `[T05a]` A node is executable if all its input tensors are initialized
- `[T05b]` Executing a node means assigning values to output tensors such that the inputs-outputs relation specified by the operator holds
- `[T05c]` All executable nodes are executed
- `[T05d]` An executable node is executed only once
- `[T05e]` A tensor is assigned at most once (Single Assignment)

Formal specification and verification

Other cases: the **graph**



Graph

- `[T01a]` A graph contains a set of nodes
- `[T01b]` A graph contains a set of tensors that are inputs and outputs of the nodes
  - Some of those tensors are inputs (resp. outputs) of the graph, i.e, their values are set (resp. returned) before (resp. after) executing the graph

Nodes

- `[T03a]` A node refers to an operator
  - An operator may be referred to by multiple nodes
- `[T03b]` There is a 1-to-1 mapping between the set of inputs and outputs of a node and the set of inputs and outputs of its associated operator `[R1]`.
  - Note that is is a restriction with respect to the ONNX standard that allows fewer inputs or outputs when the omitted input or output is optional.

Tensors

- `[T02b]` A tensor is an object that can hold a value or be uninitialized
- `[T02a]` A tensor is identified by a unique identifier within a graph

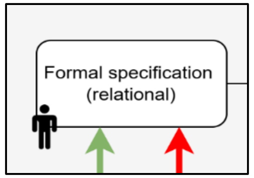
Operators

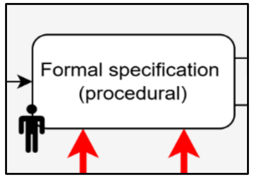
- `[T04a]` An operator specifies a relation (a function) between a set of input parameters and a set of outputs parameters.
  - Input and output parameters (resp. output) are free variables that can be bound to tensors using nodes
  - An operator has at least one output

```
type graph = {
    gi: list tensor_id;          (* graph inputs *)
    go: list tensor_id;          (* graph outputs *)
    gt: list tensor_id;     (* graph tensors *)
    gn: list node;           (* graph nodes *)
}
```

```
type node = {
    ope: operator; (* The operator referred to by the node *)
    oi: list tensor_id; (* Input tensors, position-wise *)
    ou: list tensor_id; (* Output tensors, position-wise *)
}
    invariant {
        length oi = length ope.opi /\
        length ou = length ope.opo
    }
    by
    {
        ope= { name = ""; opi = Nil; opo = Nil };
        oi = Nil;
        ou = Nil;
    }
```

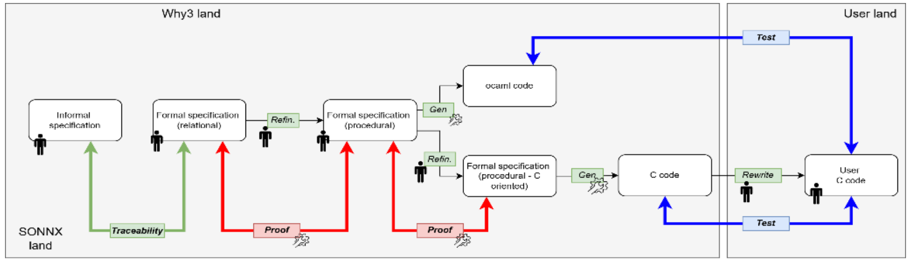
```
type operator = {
    name: string; (* name of the operator *)
    opi: list shape;  (* input shapes *)
    opo: list shape;  (* output shapes *)
}
```



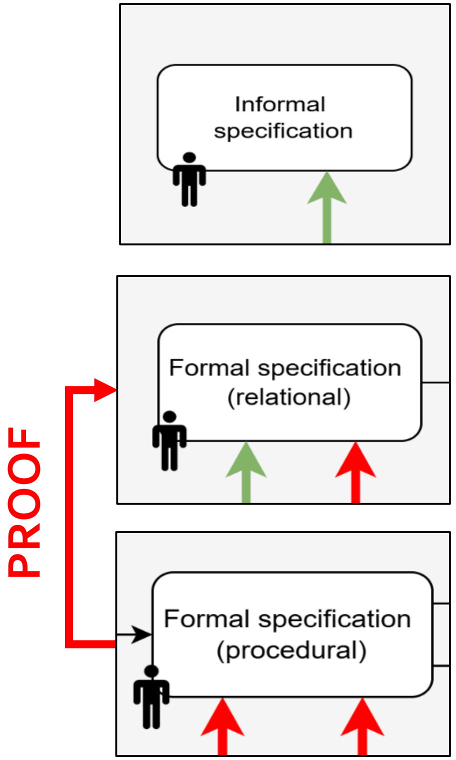


Deliverables

Other cases: the **graph**




```
Partial (4/5) (23ms) (alt-ergo 4) (split_vc 2) (depth 2)
let rec assign_list (s: graph_state) (l: lis_map) : graph_state
    (* A tensor shall only appear once in the state *)

    (* A tensor shall only appear once in an assignment *)
    requires { forall t: tensor_id. t_appears_at_most_once t l }

    (* The assignment is correct  *)
    ensures { forall t, v . Mem.mem (t, v) l ->
        my_map_get_logic result t =  v }
    variant { l }
=
    match l with
    | Nil -> s  (* Nothing to assign, the state does not change *)
    | Cons (t, v) xs ->
        (* Assign the value and continue with the rest of the assignment list *)
        let s' = my_map_set s t v in
            (* Tensor t is correctly assigned *)
            assert {  my_map_get_logic s' t =  v } ;
            assume { NumOcc.num_occ t (project xs) = 0  } ;
            assume { forall t'. t_appears_at_most_once t' xs } ;
            let s'' = assign_list s' xs in
                assume { forall t'', v'' . Mem.mem (t'', v'') xs -> my_map_get_logic s' t'' =  v'' } ;
                s''
    end
```



Formal specification and verification

Some questions

Add — Trivial:
a review is sufficient

sqrt — There is a simple property:

$$sqrt(x) \times sqrt(x) = x$$

conv — Non trivial:
we specify the algorithm

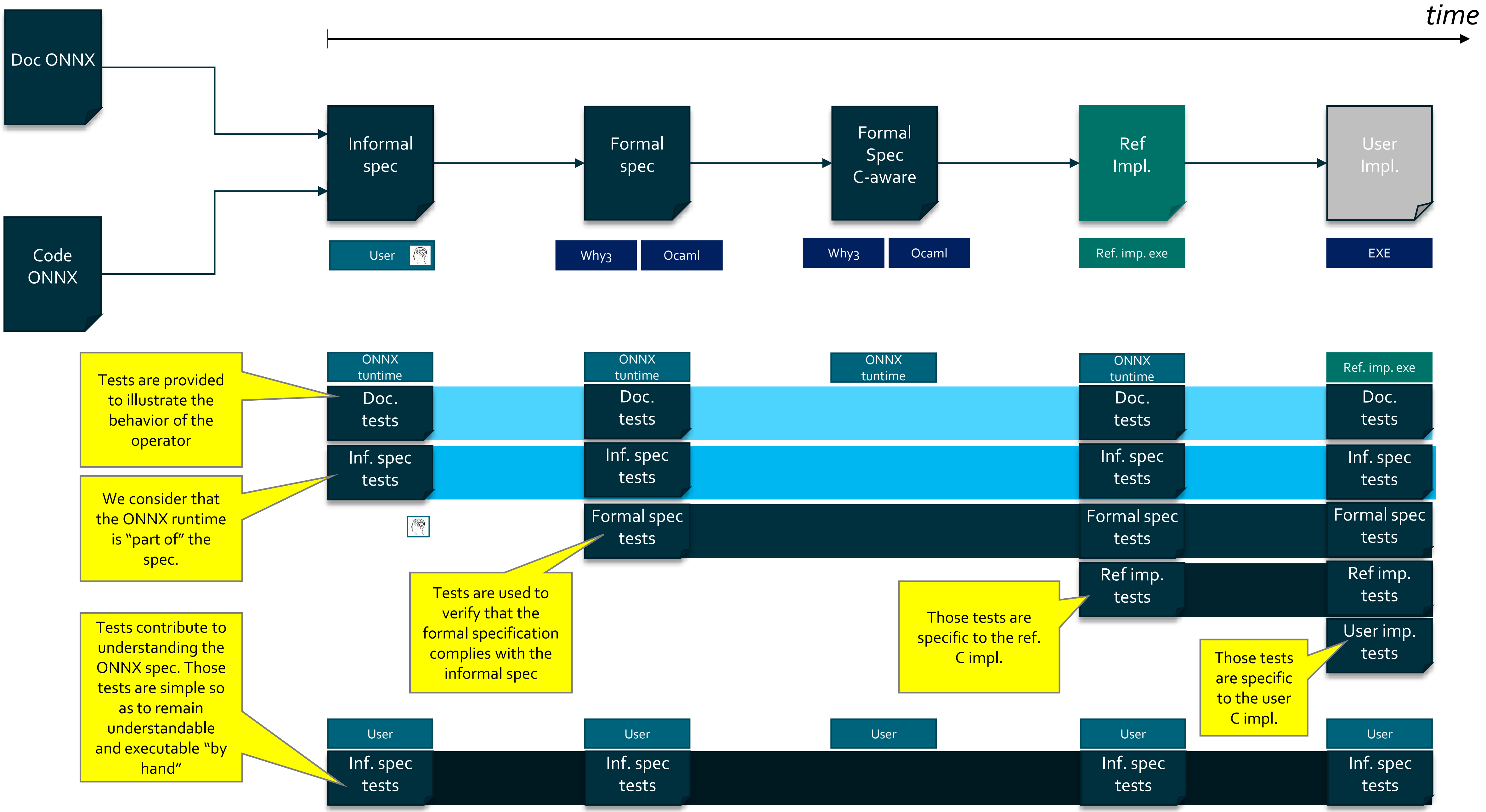
exp → log

$$\log(\exp(x)) = x$$

Testing

- "Testing" the documentation
- Back-to-back testing of implementation

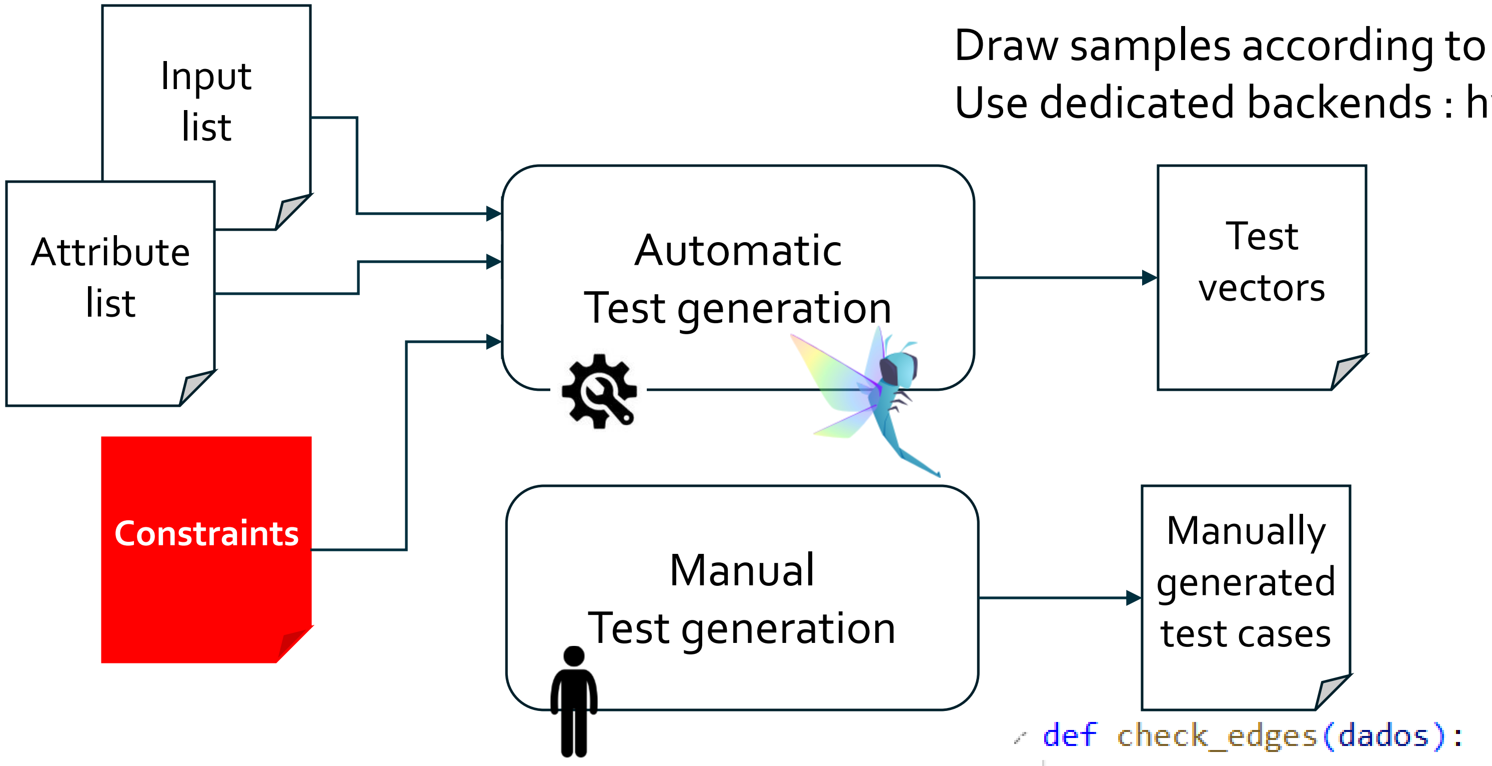
Tests



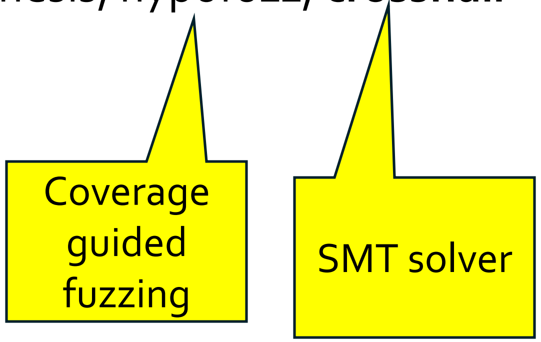
time
Doc ONNX
Code ONNX
Informal spec
Formal spec
Formal Spec C-aware
Ref Impl.
User Impl.
User
Why3
Ocaml
Why3
Ocaml
Ref. imp. exe
EXE
ONNX tuntime
ONNX tuntime
ONNX tuntime
ONNX tuntime
Ref. imp. exe
Doc. tests
Inf. spec tests
Formal spec tests
Ref imp. tests
User imp. tests
User
Tests are provided to illustrate the behavior of the operator
We consider that the ONNX runtime is "part of" the spec.
Tests contribute to understanding the ONNX spec. Those tests are simple so as to remain understandable and executable "by hand"
Tests are used to verify that the formal specification complies with the informal spec
Those tests are specific to the ref. C impl.
Those tests are specific to the user C impl.

Testing

Generating test vectors with Hypothesis



Draw samples according to some distribution
Use dedicated backends : hypothesis, hypofuzz, **crosshair**



ONNX

Testing

Generating test vectors with Hypothesis

```python
# Dilations [C3] -> X [C3]
dilation_max = []
for i in range(num_spatial_axes_w):
    if w_spatial_axis[i] > 1:
        dilation_max.append(math.floor((spatial_dimension_values_w_max[i] - 1) /
                                       (w_spatial_axis[i] - 1)))
    else:
        dilation_max.append(inputs_atributes['dilation_unlimited_max'])

inputs_atributes['dilation_max'].append(dilation_max)
# Dilations [C2]
dilations = [draw(st.integers
                  (min_value=inputs_atributes['dilation_min'],
                  max_value=dilation_max[i])) for i in range(num_spatial_axes_w)]

      = (dilations[0] * (w_spatial_axis[0] - 1)) + 1
gamma = (dilations[1] * (w_spatial_axis[1] - 1)) + 1

# y spatial dimension calculations
# When auto_pad is NOTSET, pads are explicit
# X [C3]
dy2 = math.floor(((alpha - (theta)) / strides[0])) + 1
dy3 = math.floor(((beta - (gamma)) / strides[1])) + 1
```

(C3) is one of the constraints given in the informal specification

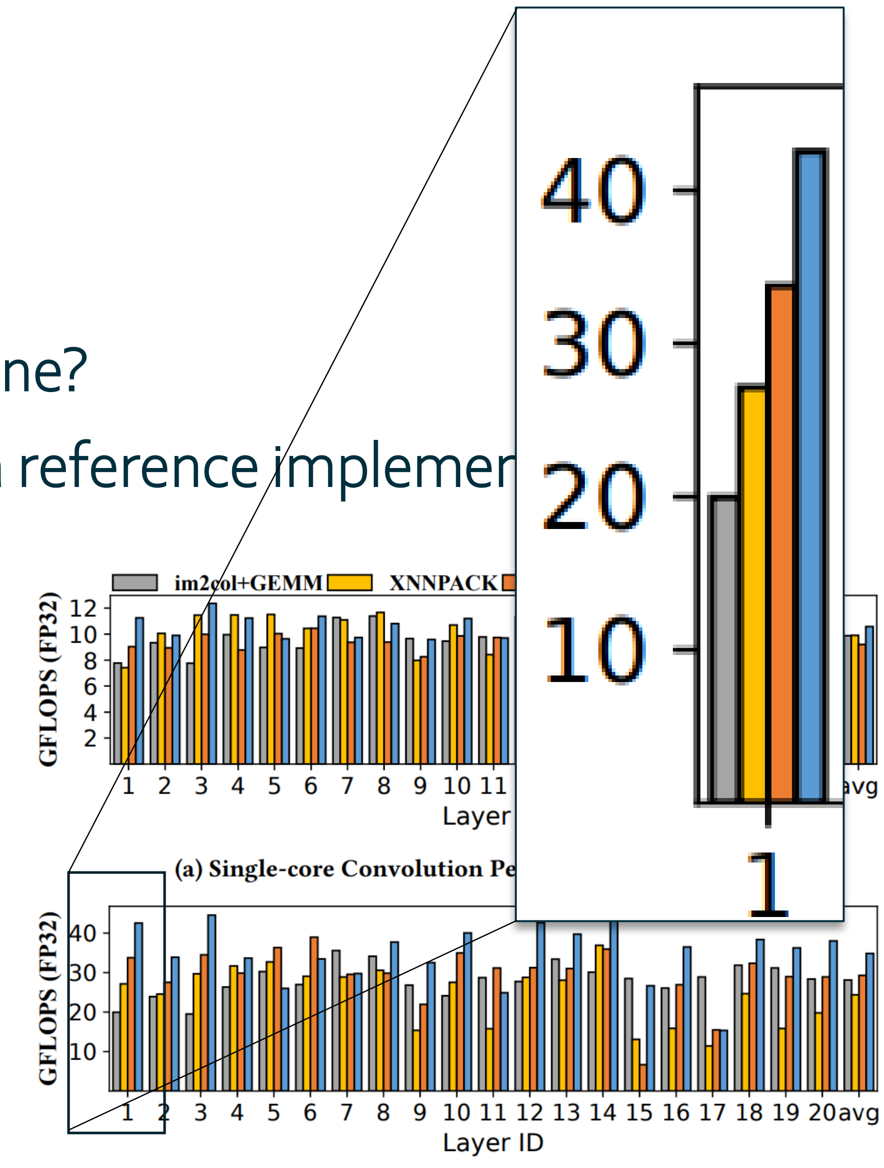
Sampling strategy

“Propagation” of constraints

Reference implementation
The Performance Question

- Performance comparison CPU vs. GPU
- How *slow* is a naïve implementation vs. an optimized one?
- How to test the correctness of an implementation *vs.* a reference implemen

- Naïve implementation
  - Simple, traceable to the specification but very slow…
  - Performance
- Tests shall be conducted on small tensors / kernels



(b) 4-cores Convolution Performance on RPi 4

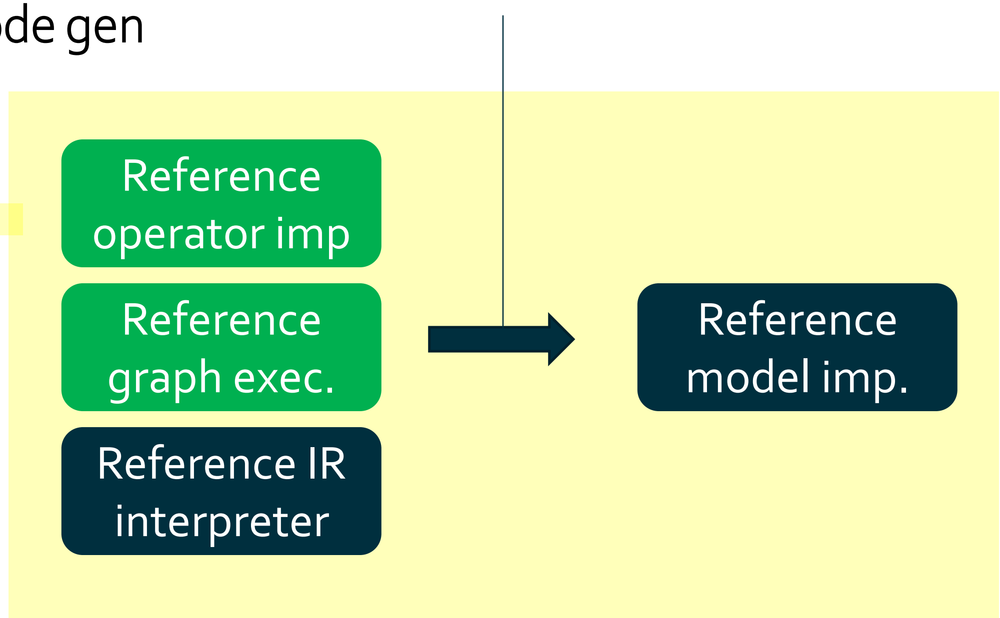
Conclusion
Where are we? What's next?

- Where are we...
  - First drafts to be consolidate / completed...

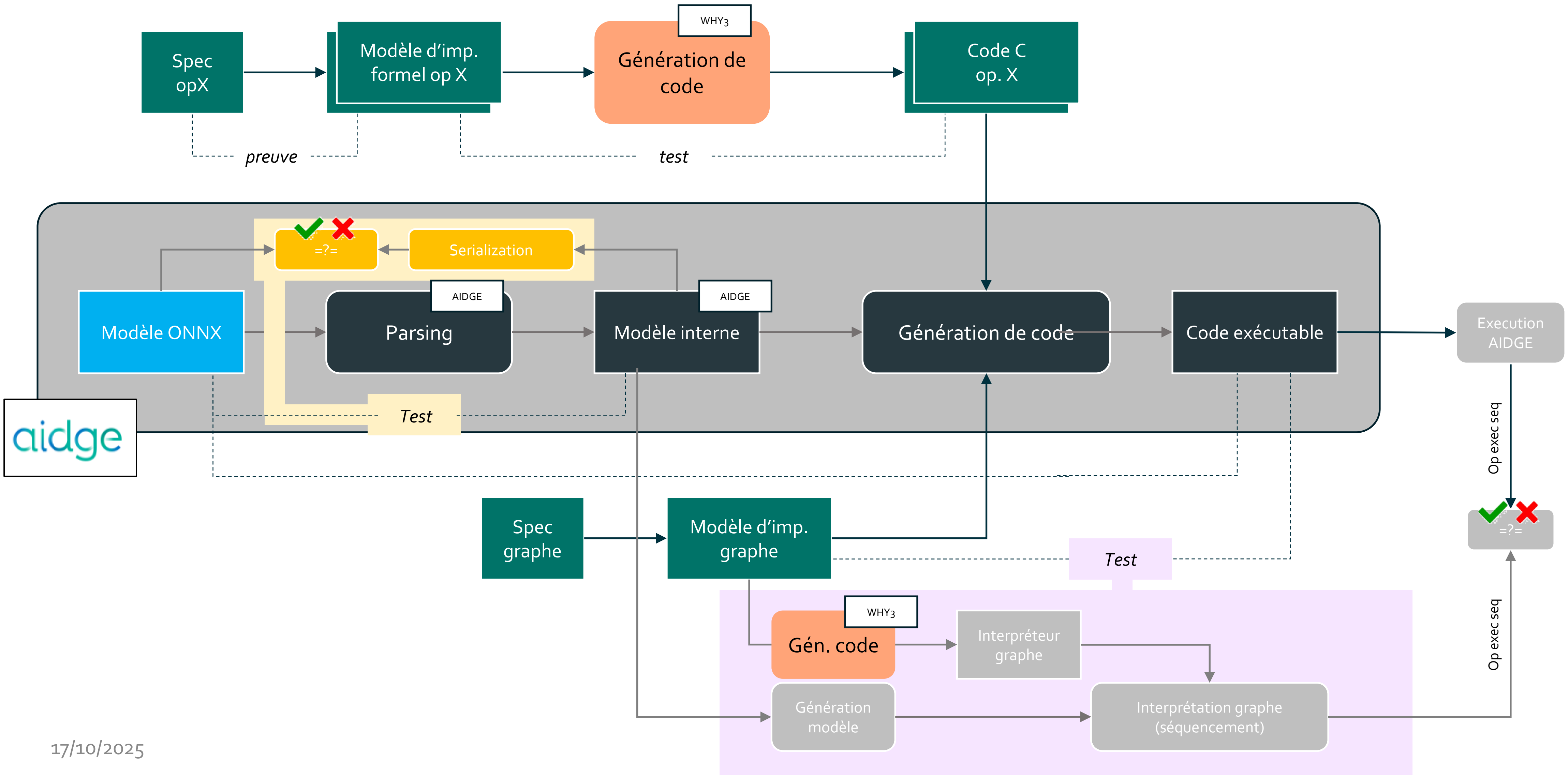
- What's next...
  - Completion of operator informal and formal spec + proof + code gen
  - Completion graph spec + proof + code gen
  - Generation of tests

  - Integration in the ONNX ecosystem...

- Integration to the aidge platform



Integration in AIDGE



Contacts

- Eric JENN (eric.jenn@irt-saintexupery.com)
- Jean SOUYRIS (jean.souyris@airbus.com)

- To join the mailing list, send a message to:
  onnx-sonnx-workgroup+subscribe@lists.lfaidata.foundation