Formalización enfocada a operaciones de Tensores en PyTorch

Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

FECHA
2025-10-07

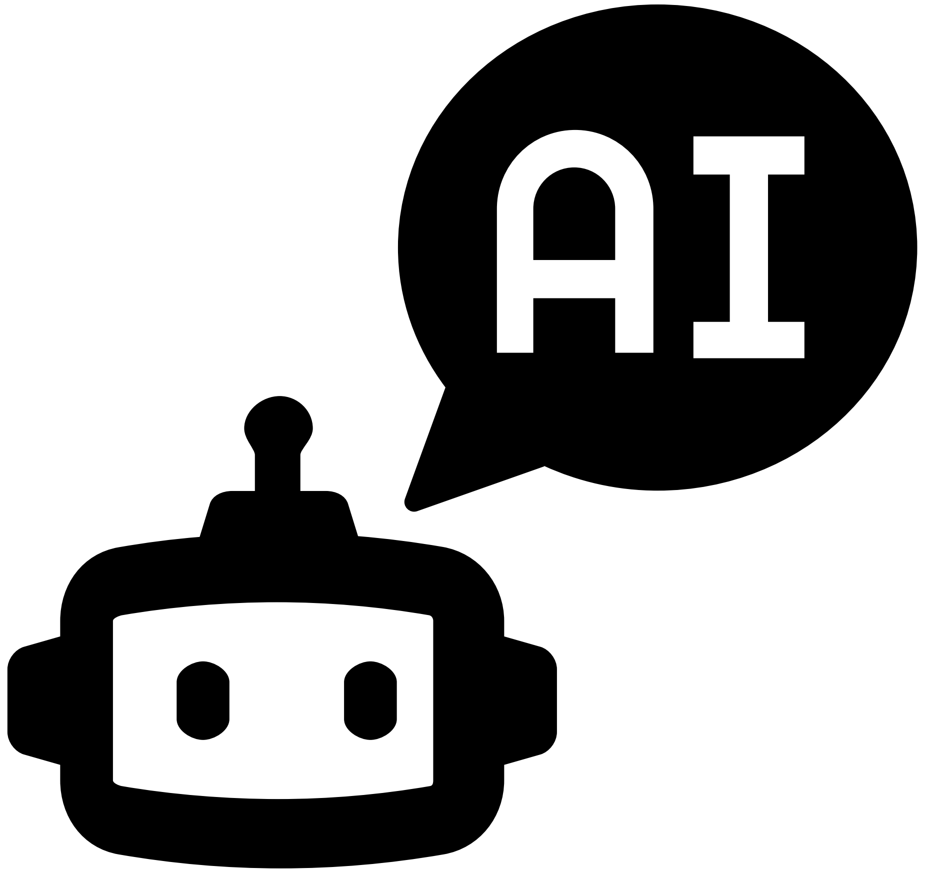
ESTUDIANTE
Jorge Cruces

PROFESORES GUÍAS
Matías Toro
Éric Tanter

COMISIÓN
Luis Mateu
Valentin Barriere

Indice

Avance de la IA y la importancia de Python



- Simplicidad y versatilidad
- Permitir profesionales de distinta áreas desarrollar soluciones
- Un ecosistema amplio de **librerías y frameworks**

Frameworks actuales

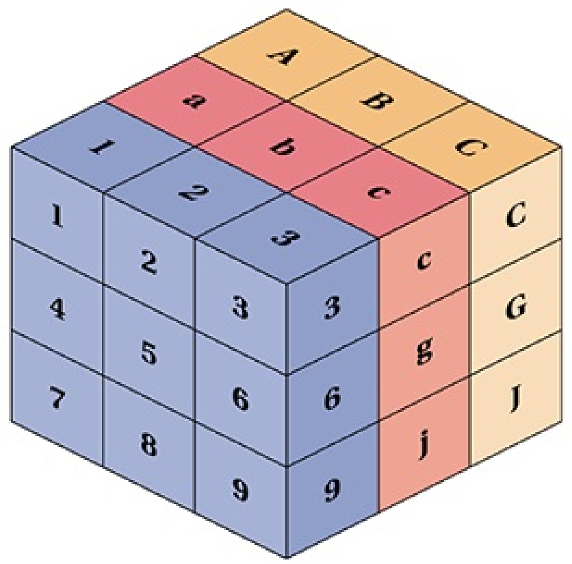
Tensorflow

PyTorch

Tensores

Un tensor es una estructura matemática que generaliza los conceptos de escalares, vectores y matrices a dimensiones superiores.

Permiten representar y manipular eficientemente datos complejos, como **imágenes** y **secuencias de texto**.



Operación valida

$$\begin{bmatrix}1&2\\3&4\\5&6\end{bmatrix} \times \begin{bmatrix}7&8&9\\1&2&3\end{bmatrix} = \begin{bmatrix}9&12&15\\25&32&39\\41&52&63\end{bmatrix}$$

3 x 2 2 x 3 3 x 3

=

Operación invalida

$$\begin{bmatrix}1&2&1\\3&4&1\\5&6&1\end{bmatrix} \times \begin{bmatrix}7&8&9\\1&2&3\end{bmatrix}$$

3 x 3 2 x 3

≠

Requieren que sus dimensiones estén **alineadas** para evitar **errores y asegurar que funcionen correctamente**

Gramática Formal

Una gramática formal es un **conjunto de reglas** que define cómo se pueden generar las cadenas válidas de un lenguaje.

A partir de este concepto, un enfoque clave para el análisis de formas de los tensores consiste en diseñar un **sistema de tipos** que permita razonar sobre ellas, donde las **formas** se representan como anotaciones de tipo.

1. $S \to a$
2. $S \to SS$
3. $aSa \to b$

SMT Solvers

SMT Solvers (Satisfiability Modulo Theories Solvers) son herramientas que permiten verificar si una fórmula lógica es **satisfacible** bajo ciertas teorías matemática.

Una **fórmula** (por ejemplo, una expresión lógica) es **satisfacible** si puede ser verdadera en algún caso.

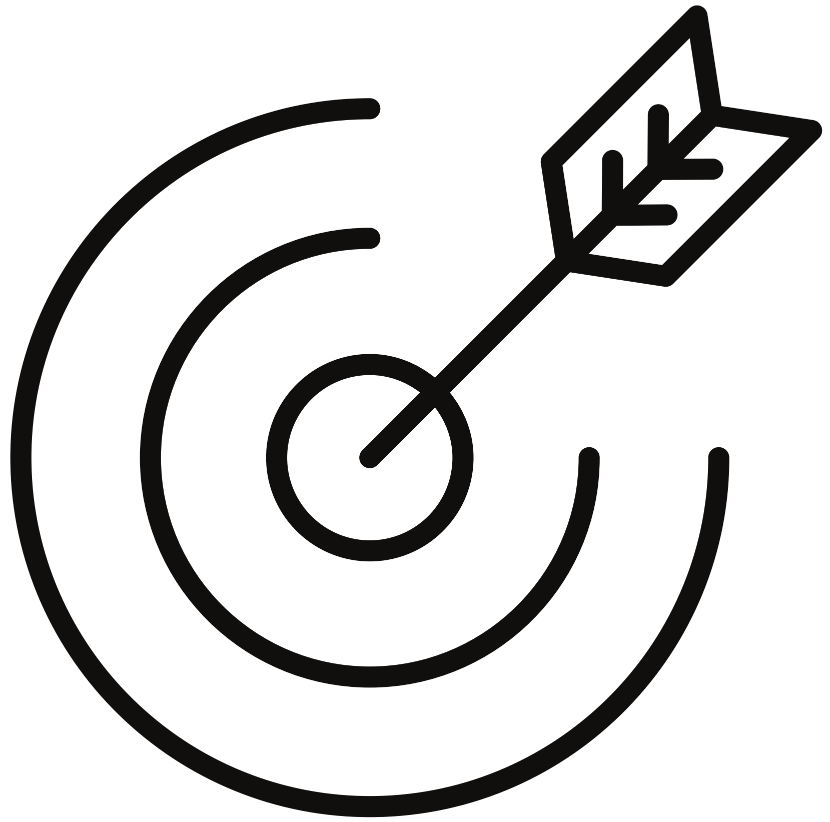
Ejemplo de un programa en PyTorch

```python
# 2 frases, cada una con 4 palabras
data = torch.randn(2, 4, 3)  # [2 frases, 4 palabras, 3 dimensiones]
labels = torch.tensor([1, 0])  # Etiquetas: positiva (1), negativa (0)

class SentimentClassifier(nn.Module):
    def __init__(self):
        super(SentimentClassifier, self).__init__()
        self.fc1 = nn.Linear(4 * 3, 8)  # Aplana: 4 palabras * 3 dim = 12
        self.fc2 = nn.Linear(8, 2)      # Salida: 2 clases (positiva/negativa)

    def forward(self, x):
        batch_size = x.size(0)
        x = x.reshape(batch_size, 4 * 8)  # Reshape: [batch, 4, 3] -> [batch, 12]
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

```
RuntimeError: shape '[2, 32]' is invalid for input of size 24
```

Objetivo General

Desarrollar una **formalización** para las operaciones tensoriales más utilizadas en PyTorch, con el fin de verificar la coherencia dimensional en programas que combinan múltiples operaciones tensoriales.

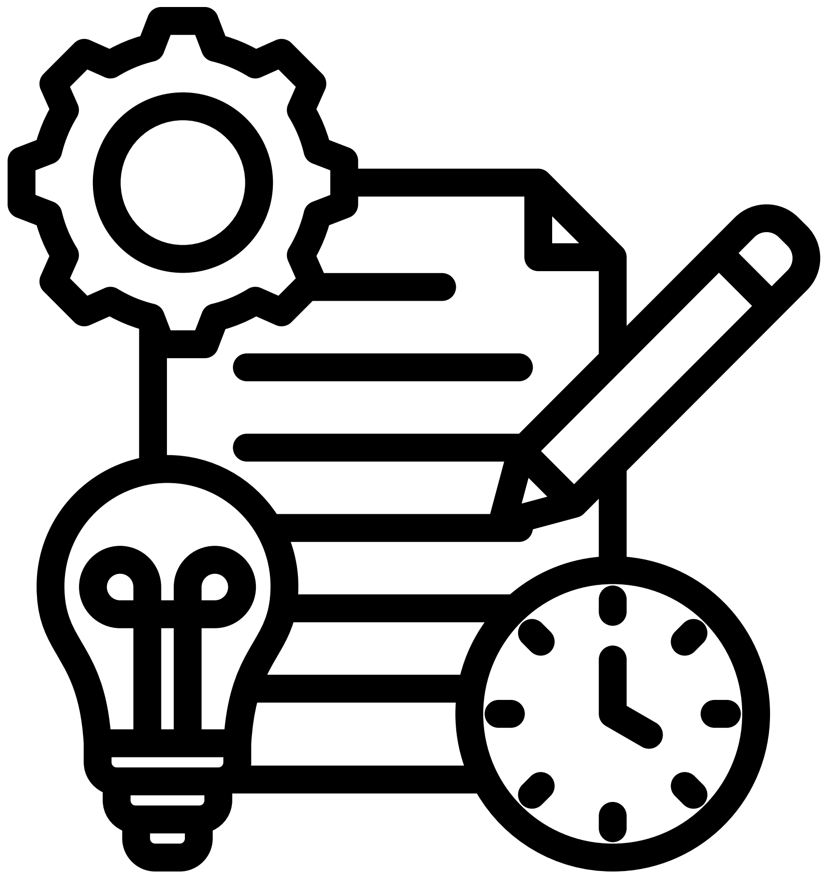


Objetivos Especificos

- Desarrollar una herramienta para el análisis de operaciones con tensores en proyectos de PyTorch.
- Evaluar herramientas y literatura actual del problema.
- Formalizar las restricciones de dimensionalidad en las operaciones más importantes.

Metodologia

1. Recolección de operaciones
2. Revisar literatura y herramientas
3. Diseñar una gramática
4. Evaluar la gramática con un ejemplo concreto



Recolección y Repositorios

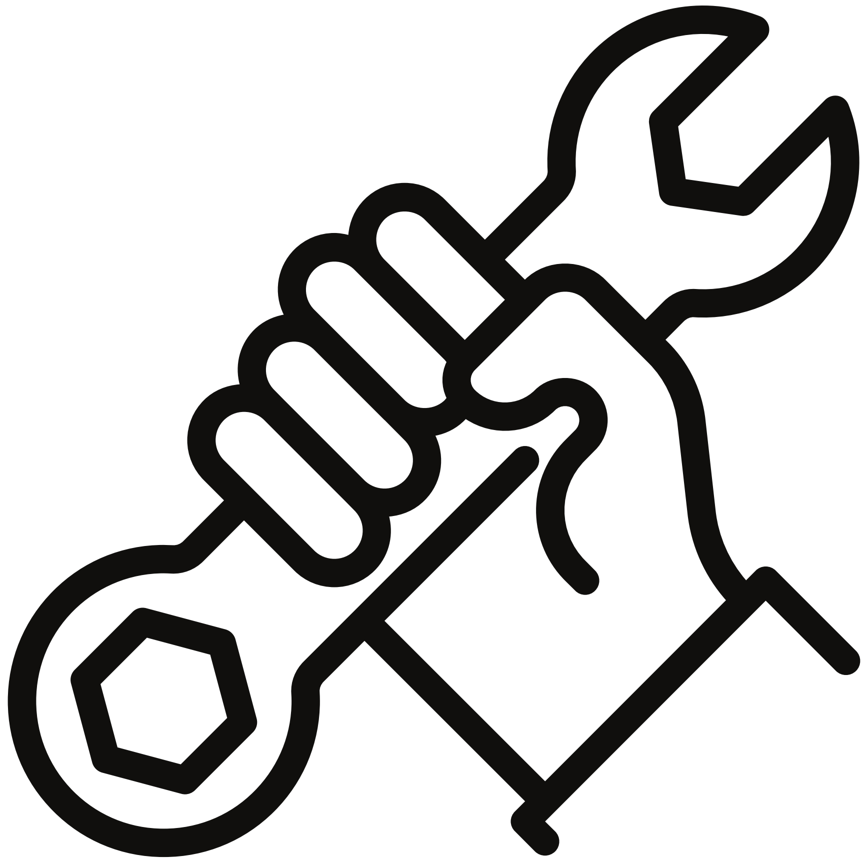
1. BERT
2. Whisper
3. LLama
4. Transformers
5. CC 6205
6. Detectron2
7. YoloV5
8. Segment anything
9. Clip
10. Blip
11. Stable diffusion

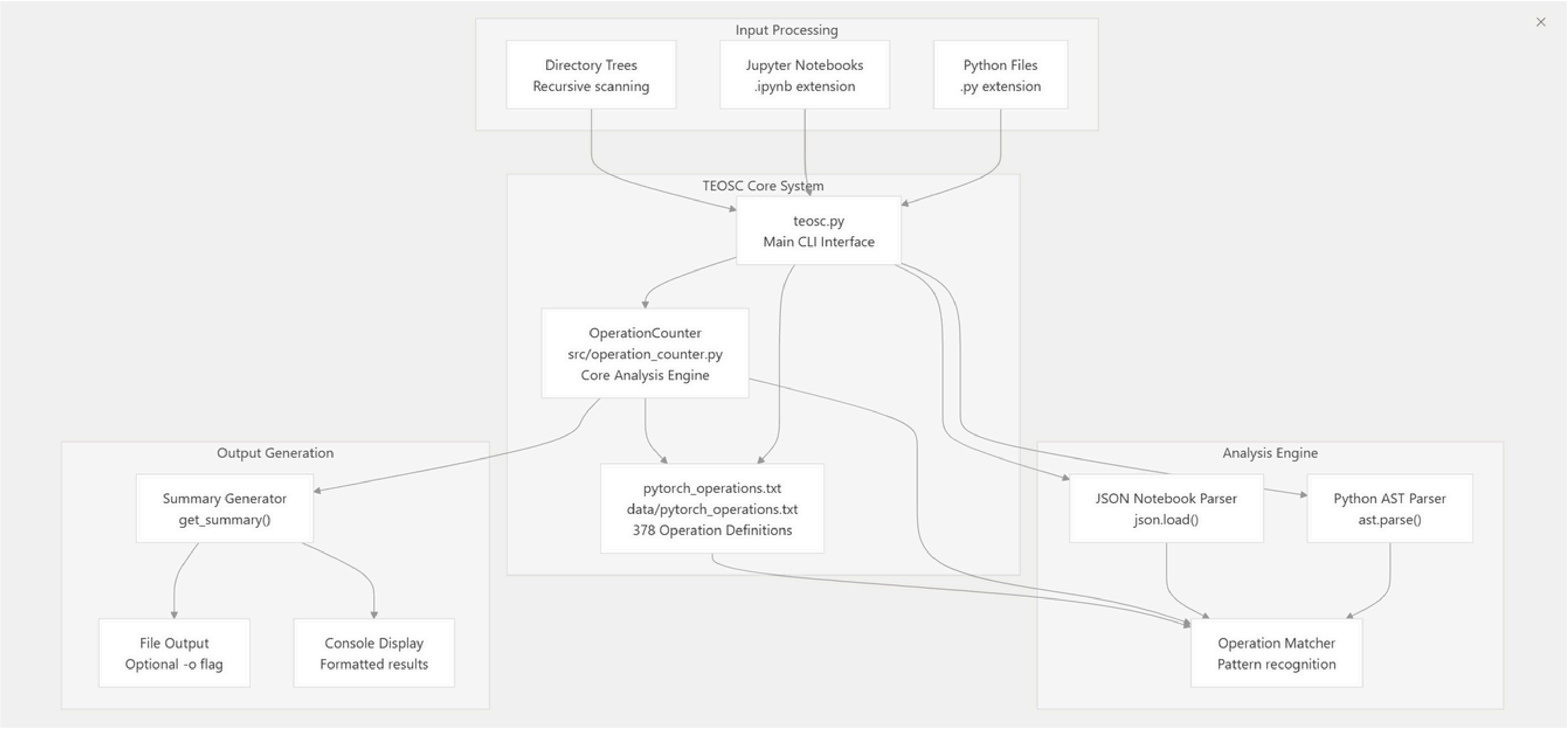
→ Analizar código fuente → Destilar una lista de operaciones más comunes

TEOSC

TEOSC, por su nombre en inglés, **Tensor Operations Static Counter**, es una herramienta que permite contar operaciones de tensores de PyTorch dentro de archivos.

Arquitectura





Ejecución

```
# Analyze a single Python file
python teosc.py path/to/file.py

# Analyze a Jupyter notebook
python teosc.py path/to/notebook.ipynb

# Analyze an entire directory recursively
python teosc.py path/to/directory

# Save results to a file
python teosc.py path/to/file.py -o results.txt
```

Formato de salida

```
[RESULTS] Total Operation Summary:
--------------------------------------------------
Files scanned (3):
  - /path/to/file1.py
  - /path/to/file2.py
  - /path/to/notebook.ipynb
--------------------------------------------------
Operation Count Summary:
tensor: 5
add: 3
matmul: 2
...
--------------------------------------------------
```

TEOSC

- 11 Repositorios
- 4427 Archivos .py y .ipynb
- 185 Operaciones de PyTorch

```
[RESULTS] Total Operation Summary:
--------------------------------------------------
Files scanned (4427):
  - ..\ptoc_repos\BERT\bind_pyt.py
  - ..\ptoc_repos\BERT\create_pretraining_data.py
  - ..\ptoc_repos\BERT\extract_features.py
  - ..\ptoc_repos\BERT\file_utils.py
  - ..\ptoc_repos\BERT\inference.py
  - ..\ptoc_repos\BERT\modeling.py
  - ..\ptoc_repos\BERT\optimization.py
  - ..\ptoc_repos\BERT\run_glue.py
  - ..\ptoc_repos\BERT\run_pretraining.py
  - ..\ptoc_repos\BERT\run_squad.py
  - ..\ptoc_repos\BERT\run_swag.py
  - ..\ptoc_repos\BERT\schedulers.py
```

#	Operación	Frecuencia
1	range	5038
2	view	4375
3	size	4140
4	nn.Linear	3862
5	tensor	3300
6	reshape	3277
7	transpose	3101
8	cat	2357
9	sum	2146
10	split	2059
11	unsqueeze	2012
12	zeros	1907
13	arange	1807
14	ones	1643
15	max	1570
16	nn.LayerNorm	1366
17	squeeze	1306
18	expand	1288
19	any	1284
20	allclose	1224
21	nn.Dropout	1195
22	permute	1152
23	min	951
24	mean	939
25	matmul	852
26	stack	786
27	all	716
28	sqrt	682
29	flatten	661
30	nn.Embedding	660
31	where	595
32	abs	553
33	repeat	551
34	nn.Conv2d	530
35	clamp	484
36	rand	478
37	log	468
38	einsum	447
39	concat	435
40	asarray	422

Lista final de Operaciones

#	Operación	Breve descripción
1	`range`	Crea un tensor con una secuencia de enteros en un rango especificado.
2	`view`	Reinterpreta la memoria del tensor con una nueva forma, sin copiar datos. Requiere que la nueva forma sea compatible con la disposición contigua de los datos en memoria.
3	`reshape`	Cambia la forma de un tensor sin alterar sus datos.
4	`transpose`	Intercambia dos dimensiones de un tensor.
5	`cat`	Concatena una lista de tensores a lo largo de una dimensión específica.
6	`sum`	Calcula la suma de los elementos de un tensor a lo largo de dimensiones especificadas.
7	`split`	Divide un tensor en sub-tensores según tamaños o número de secciones.
8	`unsqueeze`	Añade una dimensión de tamaño uno en la posición especificada.
9	`zeros`	Crea un tensor lleno de ceros con la forma dada.
10	`arange`	Similar a `range`, genera un tensor con valores igualmente espaciados en un intervalo.
11	`ones`	Crea un tensor lleno de unos con la forma dada.
12	`max`	Devuelve el valor máximo y, opcionalmente, el índice a lo largo de una dimensión.
13	`squeeze`	Elimina dimensiones de tamaño uno de un tensor.
14	`expand`	Expande un tensor a una nueva forma sin copiar datos, replicando valores según sea necesario.
15	`any`	Devuelve un tensor con booleanos dependiendo si algún elemento del tensor cumple una condición.

Literatura y herramientas

Revisar como se había abordado este problema anteriormente.

El objetivo es extraer lo más importante de estos trabajos para construir una base sólida que permita diseñar una **solución más robusta y completa**.

Cada herramienta sera evaluada en terminos de

- Capacidad de detectar errores
- Integración con ecosistémas existentes
- Uso práctico en aplicaciónes

Trabajos más relevantes

1.GraTen

- Primer trabajo realizado
- Gramática expresiva
- Tipado gradual y prueba formales

2.Pythia

- Análisis basado en operaciones matemáticas
- Enfoque práctico.
- Trabajo realizado para Tensorflow

3.PyTea

- Cobertura PyTorch y gramática expresiva
- Restricciones se integran de manera explícita dentro de la gramática.
- Integración con Z3

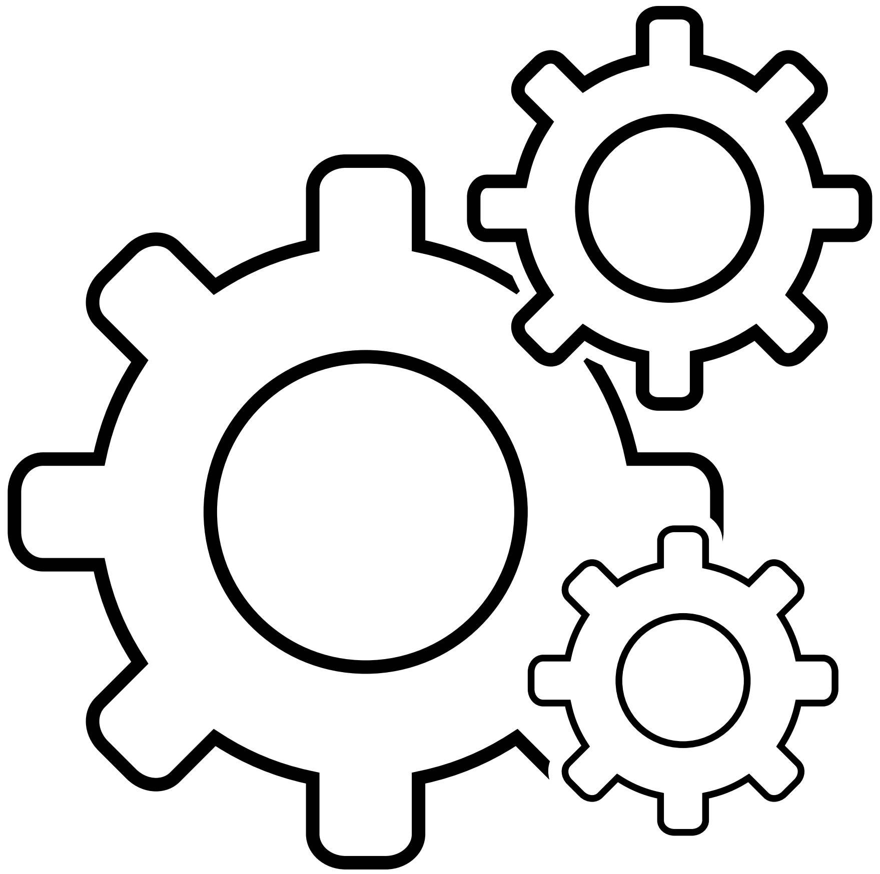
Generalizing Shape Analysis with Gradual Types

- Enfocado en PyTorch
- Gramática expresiva (Se basó la próxima gramática en el diseño inicial presentado en este trabajo)
- Desarrollado durante una pasantía en Meta por la autora **Zeina Migeed**
- Desarrollo industrial
- Uso de solver SMT Z3 de **Microsoft**
- Referencia a trabajos previos

Limitación: acceso restringido (no es público).

Aprendizajes de la literatura y herramientas

- Una buena forma de abordar el problema es a traves de una **gramática**
- Balance entre formalización y lo práctico.
- Ventaja de uso de solvers SMT industriales.
- Relevancia en el contexto de PyTorch



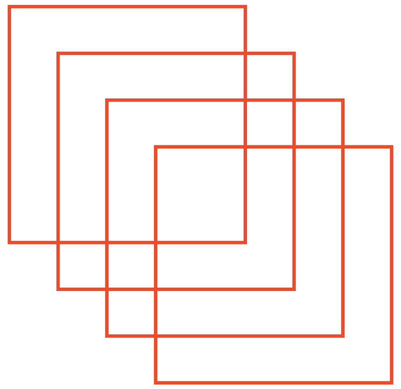
Obtención de restricciones operaciones

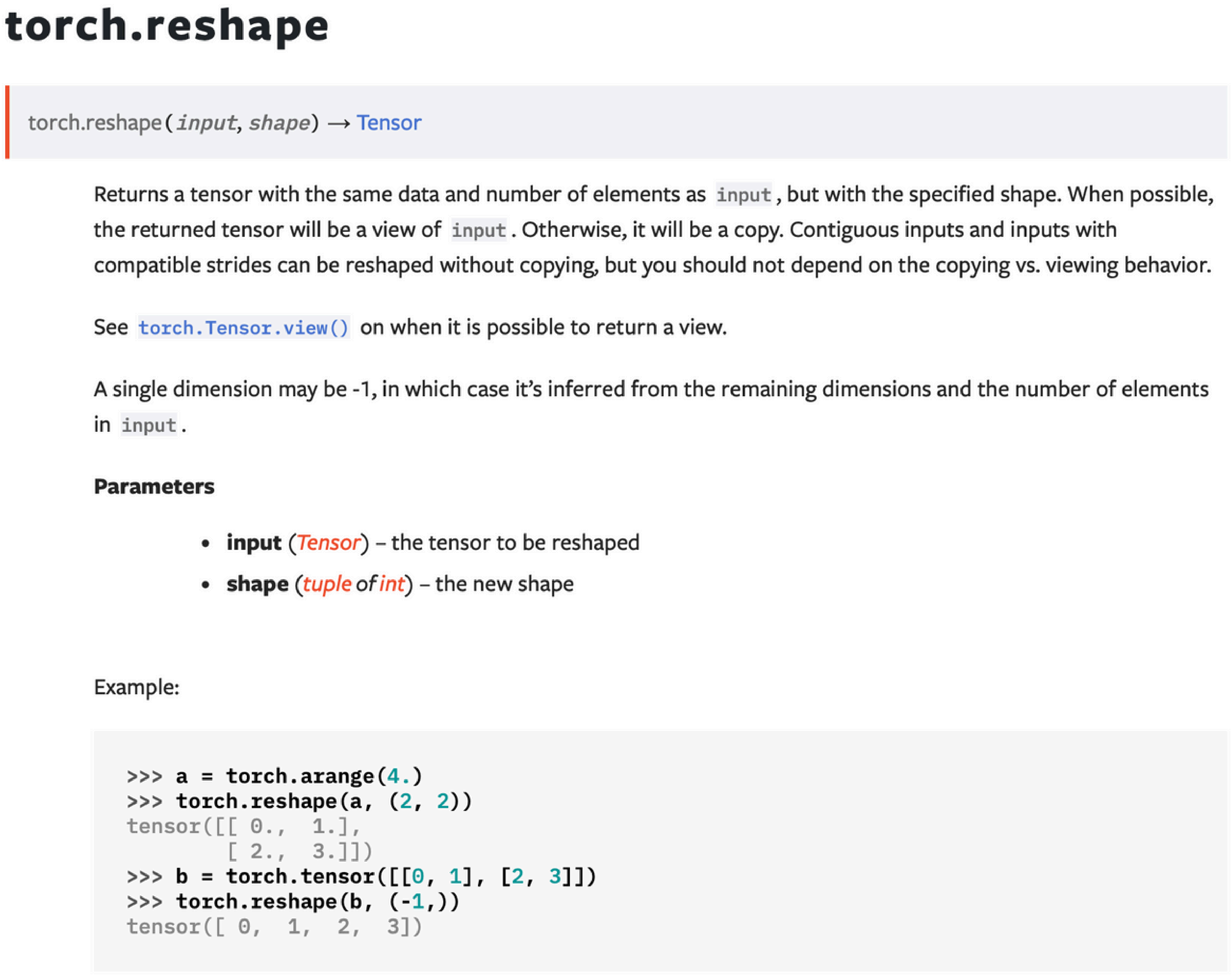
Problema:

Documentación no trae lista de errores o restricciones sino solo su uso

¿Cómo conseguir las restricciones?

1. Obtención de errores para cada operación
2. Deducir restricciones a través de errores





torch.reshape

torch.reshape(*input*, *shape*) → Tensor

Returns a tensor with the same data and number of elements as `input`, but with the specified shape. When possible, the returned tensor will be a view of `input`. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

See `torch.Tensor.view()` on when it is possible to return a view.

A single dimension may be -1, in which case it's inferred from the remaining dimensions and the number of elements in `input`.

Parameters

- **input** (*Tensor*) – the tensor to be reshaped
- **shape** (*tuple of int*) – the new shape

Example:

```
>>> a = torch.arange(4.)
>>> torch.reshape(a, (2, 2))
tensor([[ 0.,  1.],
        [ 2.,  3.]])
>>> b = torch.tensor([[0, 1], [2, 3]])
>>> torch.reshape(b, (-1,))
tensor([ 0,  1,  2,  3])
```

Errores

- Usar más de una dimensión con el valor −1

```
t = torch.zeros(4)
torch.reshape(t, (-1,-1))
>>> RuntimeError: only one dimension can be inferred
```

- Especificar una dimensión con un valor negativo distinto de −1.

```
t = torch.zeros(4)
torch.reshape(t, (-2,2))
>>> RuntimeError: invalid shape dimension -2
```

- Número total de elementos no coincide con el del tensor original.

```
t = torch.zeros(4, 2)
torch.reshape(t, (4, 3))
>>> RuntimeError: shape '[4, 3]' is invalid for input of size 8
```

Restricciones

reshape([x_1, x_2, ..., x_n],(y_1, y_2, ..., y_m)) = [y_1, y_2, ... , y_m]

$$\forall i \in [1..m] : \quad y_i > 0 \ \vee \ y_i = -1$$

y

$$\prod_{i=1}^{n} x_i = \prod_{i=1}^{m} y_i$$

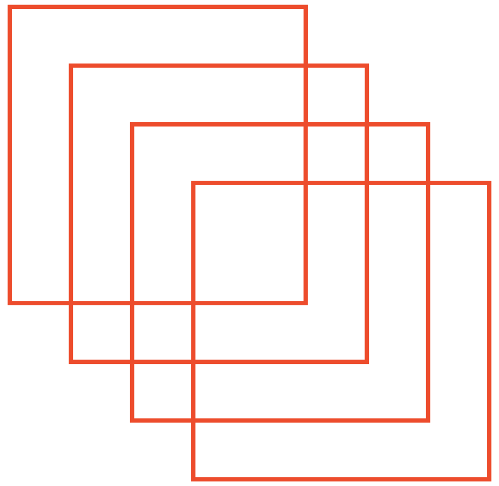
Se decidió trasladar esta lógica a una gramática formal que permitiera describir las **restricciones sobre los tensores en términos de tipos**.

Consideraciones gramática

- **Generalizing Shape Analysis with Gradual Types**: Adopción y extensión de **gramática previa.**
- **PyTea**: Integración de restricciones dentro de la gramática.
- **Pythia:** Enfoque práctico.
- **PyTea y Generalizing:** Uso de Z3 como SMT Solver

Sintaxis

$$
\begin{array}{rl}
 & n, m \in \mathbb{Z} \\
\text{(Declaration)} & decl ::= x : T \\
\text{(Type)} & T ::= \mathbb{B} \mid \mathbb{Z} \mid TT([d_0, ..., d_n]) \mid [T] \\
\text{(Dimension)} & d ::= n \mid x \\
\text{(Constraint)} & C ::= e\ op_c\ e \mid C \wedge C \mid C \vee C \mid \top \\
\text{(Operation Constraint)} & op_c ::= \mid = \mid \neq \mid < \mid > \mid \leq \mid \geq \mid \\
\text{(Operation Expressions)} & op_e ::= + \mid - \mid * \mid / \\
\text{(Operation Unary)} & op_u ::= |\cdot| \mid \lfloor \cdot \rfloor \mid \lceil \cdot \rceil
\end{array}
$$



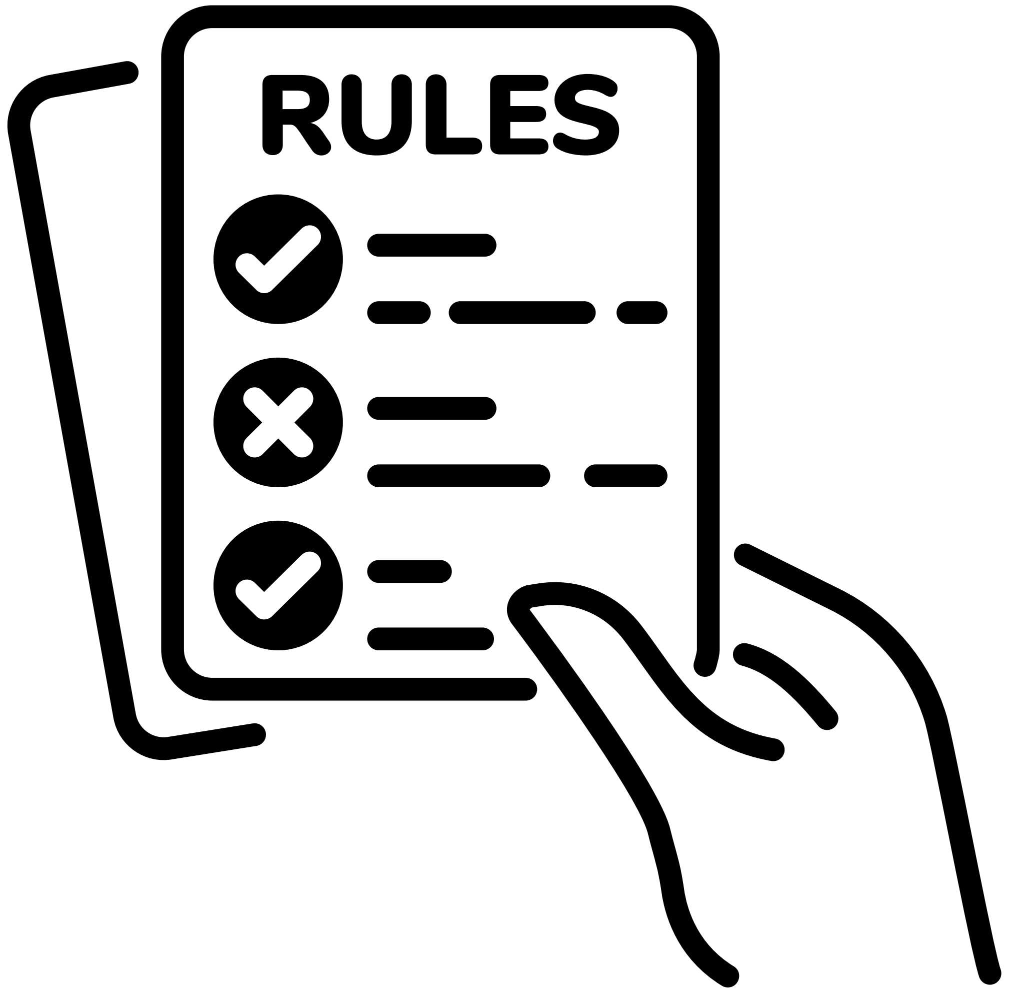
$$
\begin{array}{rl}
\text{(Expression)} & e ::= x \\
 & \mid n \\
 & \mid [e_1, ..., e_n] \\
 & \mid e\ op_e\ e \\
 & \mid op_u\ e \\
 & \mid \text{range}(e_1, e_2) \\
 & \mid \text{range}(e_1, e_2, e_3) \\
 & \mid \text{view}(e_1, e_2) \\
 & \mid \text{reshape}(e_1, e_2) \\
 & \mid \text{transpose}(e, m_1, m_2) \\
 & \mid \text{cat}(e_1, m) \\
 & \mid \text{sum}(e_1, m) \\
 & \mid \text{split}(e, m) \\
 & \mid \text{unsqueeze}(e_1, m) \\
 & \mid \text{zeros}(e) \\
 & \mid \text{arange}(e_1, e_2) \\
 & \mid \text{arange}(e_1, e_2, e_3) \\
 & \mid \text{ones}(e) \\
 & \mid \text{max}(e_1, m) \\
 & \mid \text{squeeze}(e_1, e_2) \\
 & \mid \text{expand}(e_1, e_2) \\
 & \mid \text{any}(e_1, m) \\
\text{(Environment)} & \Gamma ::= \emptyset \mid \Gamma, x : TT([d_0, ..., d_n]) \mid \Gamma, x : \mathbb{Z} = d
\end{array}
$$

Reglas de tipo

- Garantizar coherencia estructural y semántica.
- Reglas formales para verificar **tensores**, **expresiones** y **restricciones**.
- Operaciones válidas solo si cumplen condiciones de tipado.
- Estructura: reglas básicas + reglas específicas para PyTorch.

reshape([4, 2, 3], [4, 6]) ✓

reshape([4, 2, 3], [4, 7]) ✗



Operaciones PyTorch: range, zeros y reshape

$$\text{(OP-RANGE-3)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : \mathbb{Z}; C_1 \quad \Gamma \vdash_1 e_2 : \mathbb{Z}; C_2 \quad \Gamma \vdash_1 e_3 : \mathbb{Z}; C_3 \\ x \text{ free} \\ C_4 = e_3 \neq 0 \\ C_5 = |e_2 - e_1| * e_3 > 0 \\ C_6 = x = \lfloor \frac{e_2 - e_1}{e_3} \rfloor + 1 \\ C = C_1 \wedge\ C_2 \wedge\ C_3 \wedge\ C_4 \wedge\ C_5 \wedge\ C_6\end{array}}{\Gamma \vdash_3 \texttt{range}(e_1, e_2, e_3) : TT([x]); C}$$

$$\text{(OP-ZEROS-INT)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e : \mathbb{Z}; C_1 \\ x \text{ free} \\ C_2 = 0 < e \\ C_3 = x = e \\ C = C_1 \wedge\ C_2 \wedge\ C_3\end{array}}{\Gamma \vdash_3 \texttt{zeros}(e) : TT([x]); C}$$

$$\text{(OP-RESHAPE)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, ..., d'_n]); C_2 \\ C_3 = \prod_{i=1}^{n} d_i = \prod_{i=1}^{m} d'_i \\ C_4 = \bigwedge_{i=1}^{n} (d'_i > 0) \vee (d'_i = -1) \\ C = C_1 \wedge\ C_2 \wedge\ C_3 \wedge\ C_4\end{array}}{\Gamma \vdash_3 \texttt{reshape}(e_1, e_2) : TT([d'_0, ..., d'_n]); C}$$

Operaciones PyTorch: cat, split y unsqueeze

$$
\text{(OP-CAT)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : [TT(d_{0,1}, \ldots, d_{n,1}), \ldots, TT(d_{0,l}, \ldots, d_{n,l})]; C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ x \text{ free} \\ C_2 = \bigwedge_{j=1}^{l} \bigwedge_{\substack{i=0 \\ i \neq m}}^{n} d_{i,1} = d_{i,j} \\ C_3 = x = \sum_{j=1}^{l} d_{m,j} \\ C = C_1 \wedge\ C_2 \wedge\ C_3\end{array}}{\Gamma \vdash_3 \texttt{cat}(e_1, m) : TT([d_0, \ldots, d_{m-1}, x, d_{m+1}, \ldots, d_n]); C}
$$

$$
\text{(OP-SPLIT)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = m > 0 \\ C = C_1 \wedge\ C_2\end{array}}{\Gamma \vdash_3 \texttt{split}(e_1, m) : [TT(d_{0,1}, \ldots, d_{n,1}), \ldots, TT(d_{0,m}, \ldots, d_{n,m})]]; C}
$$

$$
\text{(OP-UNSQUEEZE)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n - 1 \leq m \wedge m < n + 1 \\ C = C_1 \wedge\ C_2\end{array}}{\Gamma \vdash_3 \texttt{unsqueeze}(e_1, m) : TT([d_0, \ldots, d_{m-1}, 1, d_{m+1}, \ldots, d_n]); C}
$$

Limitaciones de la Gramática

- Dependencia de valores numéricos concretos en ciertas operaciones (**any**, **expand** y **unsqueeze**)
- Restricciones en la representación de firmas complejas en **split** (Existen una variante de split con más parametros y se utilizo la versión simple)
- En determinadas operaciones fue necesario incorporar variables libres dentro de las restricciones para completar adecuadamente la definición de la operación

Evaluación

Buscaremos demostrar a través de un **ejemplo** que la gramática puede trasladarse de forma sencilla a una **herramienta práctica**

Se hicieron dos ejemplos con:

- Python
- SMT Solver Z3

1er Ejemplo Unsqueeze

$$\text{(OP-UNSQUEEZE)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n - 1 \leq m \wedge m < n + 1 \\ C = C_1 \wedge\ C_2\end{array}}{\Gamma \vdash_3 \texttt{unsqueeze}(e_1, m) : TT([d_0, \ldots, d_{m-1}, 1, d_{m+1}, \ldots, d_n]); C}$$

2do Ejemplo Reshape

$$\text{(OP-RESHAPE)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, ..., d'_n]); C_2 \\ C_3 = \prod_{i=1}^{n} d_i = \prod_{i=1}^{m} d'_i \\ C_4 = \bigwedge_{i=1}^{n} (d'_i > 0) \vee (d'_i = -1) \\ C = C_1 \wedge\ C_2 \wedge\ C_3 \wedge\ C_4\end{array}}{\Gamma \vdash_3 \texttt{reshape}(e_1, e_2) : TT([d'_0, ..., d'_n]); C}$$

Primer Ejemplo Unsqueeze: código

Para verificar la **satisfacibilidad** se traspasan las restricciones de la operación a la la libreria z3 de python y se revisa si es satisfacible bajo diversos inputs.

$$\text{(OP-UNSQUEEZE)}\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n - 1 \leq m \wedge m < n + 1 \\ C = C_1 \wedge \ C_2\end{array}}{\Gamma \vdash_3 \texttt{unsqueeze}(e_1, m) : TT([d_0, \ldots, d_{m-1}, 1, d_{m+1}, \ldots, d_n]); C}$$

1. Crear solver

```python
def is_unsqueeze_valid(e1, m: int) -> bool:
    # Crear solver Z3
    s = Solver()

    # Variables simbolicas para las dimensiones
    n = len(e1)
    e1_dims = [Int(f"d_{i}") for i in range(n)]
```

2. Agregar restricciones

```python
    # Restriccion C2
    constraint_c2_leq = -n <= m
    constraint_c2_gt: bool = m < (n + 1)
    constraint_c2 = And(constraint_c2_leq, constraint_c2_gt)
    s.add(constraint_c2)
```

3. Ver si es satisfacible

```python
    return s.check() == sat
```

Primer Ejemplo Unsqueeze: resultados

1. Caso Valido

```
Ejemplo 1: unsqueeze([4, 2, 3], 3)
C1: True
C2: -3 <= 2: True
C2: 2 < 4: True
VALID operation.
```

$$\frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \\ \Gamma \vdash_1 m : \mathbb{Z}; \top \\ C_2 = -n-1 \leq m \wedge m < n+1 \\ C = C_1 \wedge \ C_2\end{array}}{\Gamma \vdash_3 \texttt{unsqueeze}(e_1, m) : TT([d_0, \ldots, d_{m-1}, 1, d_{m+1}, \ldots, d_n]); C}$$

2. Operacion invalida: C2 no se cumple

```
Ejemplo 2: unsqueeze([1, 2, 3], -200)
C1: True
C2: -3 <= -200: False
C2: -200 < 4: True
INVALID operation.
```

3. Operacion invalida: C2 no se cumple

```
Ejemplo 3: unsqueeze([1, 2, 3], 10)
C1: True
C2: -3 <= 10: True
C2: 10 < 4: False
INVALID operation.
```

Segundo ejemplo reshape: código

$$\text{(OP-RESHAPE)}\frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, ..., d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, ..., d'_n]); C_2 \\ C_3 = \prod_{i=1}^{n} d_i = \prod_{i=1}^{m} d'_i \\ C_4 = \bigwedge_{i=1}^{n} (d'_i > 0) \vee (d'_i = -1) \\ C = C_1 \wedge\ C_2 \wedge\ C_3 \wedge\ C_4\end{array}}{\Gamma \vdash_3 \texttt{reshape}(e_1, e_2) : TT([d'_0, ..., d'_n]); C}$$

1. Crear solver

```
def is_reshape_valid(e1, e2) -> bool:
    # Crear solver Z3
    s = Solver()

    # Variables simbolicas para las dimensiones
    e1_dims = [Int(f"d_{i}") for i in range(len(e1))]
    e2_dims = [Int(f"d'_{i}") for i in range(len(e2))]
```

2. Agregar restricciones

```
# Restricciones C3: Producto de dimensiones debe ser igual
product_e1 = 1
for i, dim in enumerate(e1):
    product_e1 *= dim

product_e2 = 1
for i, dim in enumerate(e2):
        product_e2 *= dim
# Verifica que el producto de las dimensiones del tensor de entrada sea igual al
producto de las dimensiones objetivo
C_3 = product_e1 == product_e2
s.add(C3)
```

```
# Restricciones C4: Las dimensiones de salida pueden ser positiva o igual a -1
constraints_c4 = []
for i, dim in enumerate(e2):
    constraint = Or(dim > 0, dim == -1)
    constraints_c4.append(constraint)
    s.add(constraint)
```

3. Ver si es satisfacible

```
return s.check() == sat
```

Segundo ejemplo reshape: resultados

1. Caso Valido

```
Ejemplo 1: reshape([4, 2, 3], [4, 6])
C1: True
C2: True
C3: 24 == 24
C4: Or(True, False) ^ Or(True, False)
VALID operation.
```

$$
(\text{OP-RESHAPE})\ \frac{\begin{array}{c}\Gamma \vdash_1 e_1 : TT([d_0, \ldots, d_n]); C_1 \quad \Gamma \vdash_1 e_2 : TT([d'_0, \ldots, d'_n]); C_2 \\ C_3 = \prod_{i=1}^{n} d_i = \prod_{i=1}^{m} d'_i \\ C_4 = \bigwedge_{i=1}^{n} (d'_i > 0) \vee (d'_i = -1) \\ C = C_1 \wedge \ C_2 \wedge \ C_3 \wedge \ C_4\end{array}}{\Gamma \vdash_3 \mathtt{reshape}(e_1, e_2) : TT([d'_0, \ldots, d'_n]); C}
$$

2. Operación invalida: C3 no se cumple

```
Ejemplo 2: reshape([4, 2, 3], [4, 7])
C1: True
C2: True
C3: 24 == 28
C4: Or(True, False) ^ Or(True, False)
INVALID operation.
```

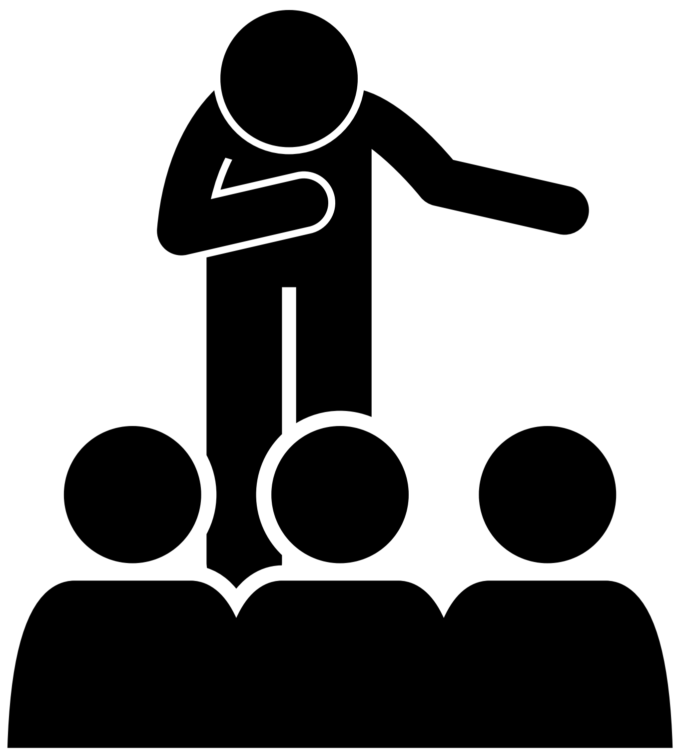
3. Operación invalida: C4 no se cumple

```
Ejemplo 3: reshape([6, 4], [-3, -8])
C1: True
C2: True
C3: 24 == 24
C4: Or(False, False) ^ Or(False, False)
INVALID operation.
```

Conclusión

Este trabajo constituye un **primer avance hacia la detección temprana de errores relacionados con la compatibilidad de dimensiones en operaciones tensoriales** dentro de programas basados en PyTorch

Esta gramática permite formalizar y estructurar el trabajo realizado, facilitando su lectura y posterior implementación.



Trabajo a futuro

- Ampliar cobertura de operaciones
- Gradual Typing
- Agregar soporte a operaciones firmas más complejas
- Implementar en un ambiente práctico ej: typechecker o sistema que requiera rigurosidad en las operaciones tensoriales

Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Formalización enfocada a operaciones de Tensores en PyTorch

FECHA
2025-10-07

ESTUDIANTE
Jorge Cruces

PROFESORES GUÍAS
Matías Toro
Éric Tanter

COMISIÓN
Luis Mateu
Valentin Barriere