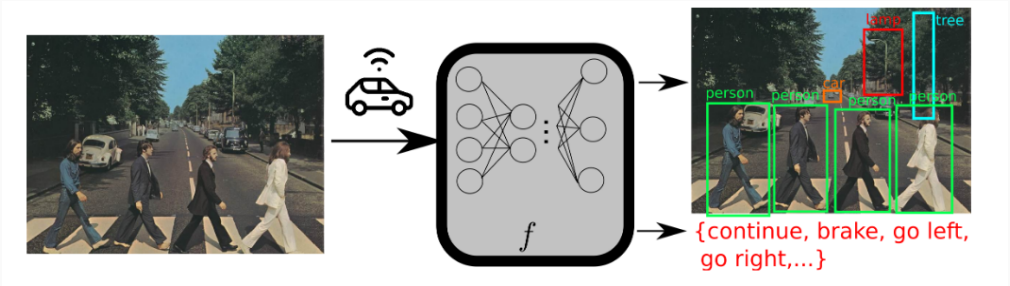
Spécifier et vérifier des propriétés de sûreté sur de l'IA

Journée commune des GDRs RADIA et GPL 2024

Michele Alberti (CEA LIST) : michele.alberti@cea.fr
Julien Girard-Satabin (CEA LIST) : julien.girard2@cea.fr
Alban Grastien (CEA LIST) : alban.grastien@cea.fr
François Bobot (CEA LIST) : francois.bobot@cea.fr
Zakaria Chihani (CEA LIST) : zakaria.chihani@cea.fr

Enjeux spécifiques de la preuve d’IA connexioniste

Des difficultés épistémiques pour la spécification



Qu'est-ce qu'une spécification d'image de piéton ? Inspécifiable formellement, donc on utilise de l'apprentissage machine.

Un de-facto standard peu satisfaisant

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)

(declare-const Y_0 Real)
(declare-const Y_1 Real)
(declare-const Y_2 Real)
(declare-const Y_3 Real)
(declare-const Y_4 Real)

; Unscaled Input 0 : (36000, 60760)
(assert (≤ X_0 0.679857769))
(assert (≥ X_0 0.268978427))

; Unscaled Input 1 : (0.7, 3.141592)
(assert (≤ X_1 0.499999896))
(assert (≥ X_1 0.11140846))

; Unscaled Input 2 : (-3.141592, -3.1315920000000004)
(assert (≤ X_2 -0.498408347))
(assert (≥ X_2 -0.499999896))

; Unscaled Input 3 : (900, 1200)
(assert (≤ X_3 0.5))
(assert (≥ X_3 0.227272727))

; Unscaled Input 4 : (600, 1200)
(assert (≤ X_4 0.5))
(assert (≥ X_4 0.0))

; unsafe if coc is ¬ minimal
(assert (or
  (and (≤ Y_1 Y_0))
  (and (≤ Y_2 Y_0))
  (and (≤ Y_3 Y_0))
  (and (≤ Y_4 Y_0))
))
```

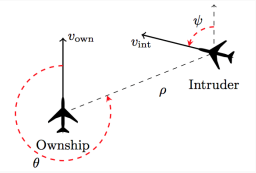
Exemple de propriété fonctionnelle exprimée dans le langage de spécification VNN-Lib [Dem+23] utilisé dans la compétition internationale de vérification de réseaux de neurones (VNN-Comp). Doit s'accompagner d'un réseau au format ONNX.

Limites

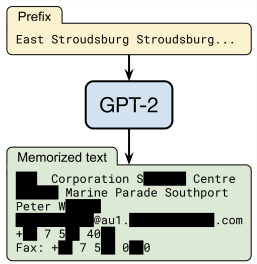
Exemple	Limite	Propriété désirable
$\forall x \in \mathrm{E} \subseteq \mathbb{R}^d.\epsilon \in \mathbb{R}.f_1(x)f_2(x) \leq \epsilon$	Spécifier des propriétés sur différents réseaux ?	*Composabilité des preuves*
$\forall(x, x^{'}), g(f, x, x^{'}) \in \mathscr{D}, f(x) = f(x^{'})$	Comment définir et exprimer des fonctions ? Des ensembles mathématiques ?	*expressivité*

Des programmes difficiles à maîtriser...

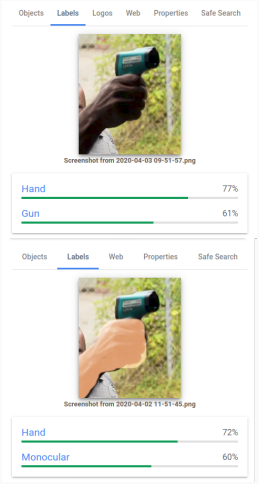
Robustesse sur des petites perturbations [Mad+17] et des transformations sémantiques [Bal+19] ?



Prouver des propriétés fonctionnelles [Kat+17] ?



Respect de la vie privée (en bonus avec les images [Car+23]) ?



Fairness[Urb+19] et qualifi-

... et beaucoup d'effort pour les analyser

- PyRAT (développé chez nous)[LLG24]
- Marabou [Kat+19]
- Neurify

... et beaucoup d'effort pour les analyser

- PyRAT (développé chez nous)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19 ; Mül+21]
- $\alpha - \beta$-Crown [Wan+21]
- NNV [a]
- FaceLattice [Yan+21]

... et beaucoup d'effort pour les analyser

- PyRAT (développé chez nous)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19 ; Mül+21]
- $\alpha - \beta$-Crown [Wan+21]
- NNV [a]
- FaceLattice [Yan+21]
- Facet-Vertex incidence [b]
- ReluDiff [PWW20]
- Peregrinn [KFS21]

---

a. https://github.com/verivital/nnv
b. https://github.com/Shaddadi/Facet-Vertex-FFNN

... et beaucoup d'effort pour les analyser

- PyRAT (développé chez nous)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19 ; Mül+21]
- $\alpha - \beta$-Crown [Wan+21]
- NNV [a]
- FaceLattice [Yan+21]
- Facet-Vertex incidence [b]
- ReluDiff [PWW20]
- Peregrinn [KFS21]

- Oval [a]
- nnenum [Bak21]
- Libra [Urb+19]

a. https://github.com/verivital/nnv
b. https://github.com/Shaddadi/Facet-Vertex-FFNN

... et beaucoup d'effort pour les analyser

- PyRAT (développé chez nous)[LLG24]
- Marabou [Kat+19]
- Neurify
- ERAN [Sin+19 ; Mül+21]
- $\alpha - \beta$-Crown [Wan+21]
- NNV [a]
- FaceLattice [Yan+21]
- Facet-Vertex incidence [b]
- ReluDiff [PWW20]
- Peregrinn [KFS21]

a. https://github.com/verivital/nnv
b. https://github.com/Shaddadi/Facet-Vertex-FFNN

- Oval [a]
- nnenum [Bak21]
- Libra [Urb+19]
- MIPVerify [TXT19]
- Planet [Ehl17]
- Sherlock [Dut+17]
- ZoPE[Str+22]
- DNNV [SED21]
- Veritex [b]
- Verinet and Venus [c]

a. https://github.com/oval-group/oval-bab
b. https://github.com/Shaddadi/veritex
c. https://github.com/vas-group-imperial/VeriNet

(crédit : Bill Wurtz)

Des niches écologique pour les prouveurs

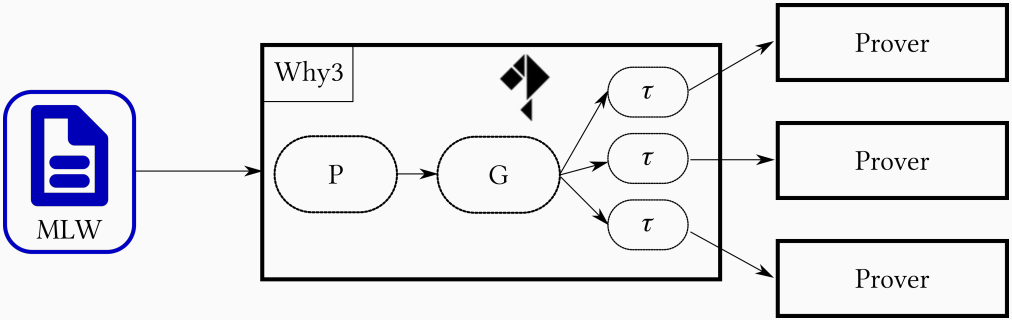
Spécialisations

1. par propriétés : robustesse locale, propriété fonctionnelles, *fairness*;
2. par techniques : Satisfaction Modulo Théorie (SMT), Programmation par Contrainte (CP), interprétation abstraite;

Ne pas se perdre dans la forêt

Un langage de spécification plus riche

Why3 à la rescousse



La plateforme de vérifications de programmes [FP13] Why3. Repose sur un langage de spécification et de programmation (WhyML), un parser (P), ungénérateur d'obligations de preuves (G), et des transformations (τ)

Spécification du langage

$$
\begin{array}{lll}
\langle decl \rangle & \texttt{type}\ \langle tId \rangle = \langle type \rangle \\
& \mid \texttt{predicate}\ \langle id \rangle \\
& \quad \langle binder \rangle^* = \langle expr \rangle \\
& \mid \texttt{function}\ \langle id \rangle \\
& \quad \langle binder \rangle^*\ \langle spec \rangle^* = \langle expr \rangle \\
\langle type \rangle & \langle tId \rangle \\
& \mid \langle type \rangle \to \langle type \rangle \\
& \mid (\langle type \rangle, \ldots, \langle type \rangle) \\
& \mid \texttt{vector}\ \langle type \rangle \\
& \mid \texttt{int} \mid \texttt{bool} \mid \texttt{float} \mid \texttt{string} \\
& \mid \texttt{model} \\
\langle binder \rangle & \langle id \rangle \mid (\langle id \rangle : \langle type \rangle) \\
\langle spec \rangle & \texttt{requires}\ \{\langle expr \rangle\} \\
& \mid \texttt{ensures}\ \{\langle expr \rangle\} \\
\langle bop \rangle & \leq \mid \geq \mid < \mid > \\
& \mid + \mid - \mid \times \mid / \\
& \mid \wedge \mid \vee \mid \to
\end{array}
$$

$$
\begin{array}{lll}
\langle expr \rangle & \langle id \rangle \\
& \mid \langle built\text{-}in \rangle \\
& \mid \langle expr \rangle \langle expr \rangle \\
& \mid ((expr), \ldots, (expr)) \\
& \mid \texttt{let}\ \langle id \rangle = \langle expr \rangle\ \texttt{in} \\
& \mid \texttt{if}\ \langle expr \rangle\ \texttt{then}\ \langle expr \rangle \\
& \quad \texttt{else}\ \langle expr \rangle \\
& \mid \langle expr \rangle \langle bop \rangle \langle expr \rangle \\
& \mid \texttt{forall} \langle binder \rangle . \langle expr \rangle \\
& \mid \texttt{exists} \langle binder \rangle . \langle expr \rangle \\
& \mid \texttt{not} \langle expr \rangle \\
& \mid \texttt{i} \in \texttt{Integer} \\
& \mid \{\texttt{true}, \texttt{false}\} \in \texttt{Boolean} \\
& \mid \texttt{f} \in \texttt{Float} \mid \texttt{s} \in \texttt{String}
\end{array}
$$

$$
\begin{array}{lll}
\langle built\text{-}in \rangle & \texttt{read\_model}\ \langle expr \rangle \\
& \mid \texttt{length}\ \langle expr \rangle \\
& \mid \texttt{has\_length}\ \langle expr \rangle\ \langle expr \rangle \\
& \mid \langle expr \rangle [\langle expr \rangle] \\
& \mid \langle expr \rangle @@ \langle expr \rangle
\end{array}
$$

Grammaire pour le langage de CAISAR, très largement redevable de WhyML. Nos additions sont encadrées.

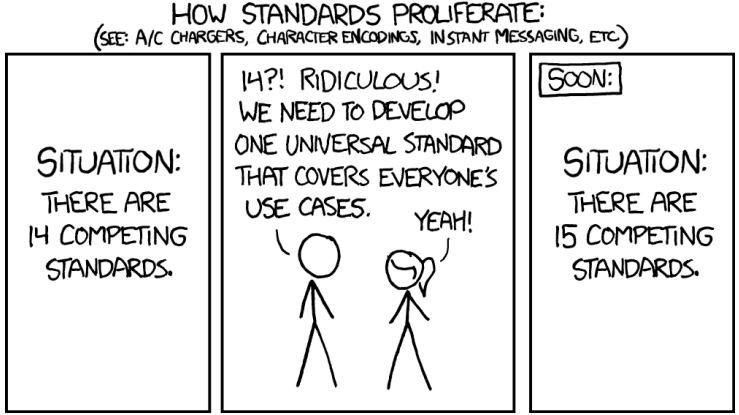
Exemple

```
theory ACASXU_P3
 constant distance_to_intruder : int = 0;
 [...]
 predicate valid_input (i: input) =
  (0.0 :t) .≤ i[distance_to_intruder]
    .≤ (60760.0 :t)
  ∧ .− pi .≤ i[angle_to_intruder] .≤ pi
  ∧ .− pi .≤ i[intruder_heading] .≤ pi
  ∧ (100.0 :t) .≤ i[speed] .≤ (1200.0 :t)
  ∧ (0.0 :t) .≤ i[intruder_speed]
    .≤ (1200.0 :t)
 constant nn_1_1: nn =
  read_neural_network "net.onnx" ONNX

 predicate is_min (o: vector t) (i: int) =
  forall j: int. 0 ≤ j < 5 → i ≠ j → o[i] .≤ o[j]

 goal P3_1_1:
  forall i: vector t.
   has_length i 5 → valid_input i →
   let is_coc_min = is_min (nn_1_1@@i) 0 in
   ¬ is_coc_min

end
```



crédit : Randall Munroe

Outillage pour la réécriture

État actuel de la vérification

Les prouveurs état de l'art sont efficaces dans leur niche, mais fonctionnent sur peu de classes de problèmes

Exemple

Soient deux réseaux nn_1 et nn_2, $x \in \mathbb{R}^2$, $\epsilon \in \mathbb{R} << 1$. Soit $r = nn_2(nn_1(x_1), x_1 + \epsilon) + nn_1(x_0)$. On souhaite garantir $r > 0$.

Exemple

Soient deux réseaux nn_1 et nn_2, $x \in \mathbb{R}^2$, $\epsilon \in \mathbb{R} << 1$. Soit $r = nn_2(nn_1(x_1), x_1 + \epsilon) + nn_1(x_0)$. On souhaite garantir $r > 0$.

Impossible d'exprimer cette propriété dans le langage d'entrée des gagnants de la VNN-Comp 202X

Notre approche

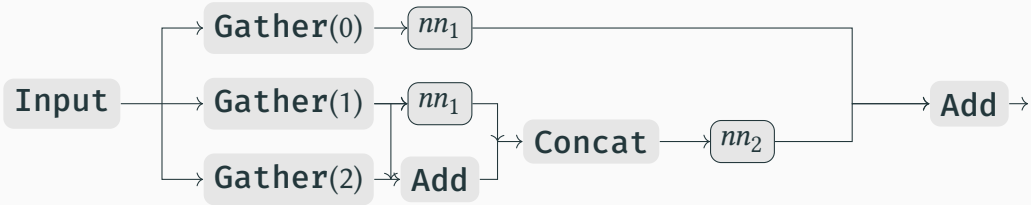
Transformer

$$\forall x_0, x_1, \epsilon.\ \mathrm{H}(x_0, x_1, \epsilon) \to nn_2@@(nn_1@@(x_1), x_1 + \epsilon) + nn_1@@(x_0) > 0$$

en

$$\forall x_0, x_1, \epsilon.\ \mathrm{H}(x_0, x_1, \epsilon) \to nn_3@@(x_0, x_1, \epsilon) > 0$$

1. distribuer x_0 et x_1 sur des évaluations séparées de nn_1 ;
2. effectuer l'addition de ϵ à x_1 ;
3. créer un nœud combinant $nn_1@@(x_1)$ et $x_1 + \epsilon$;
4. effectuer le calcul de nn_1 et nn_2 sur leurs entrées respectives ;
5. ajout des sorties.



`Gather`(0) (resp. `Gather`(1)) extrait x_0 (resp. x_1) et `Gather`(2) extrait ϵ depuis le nœud d'entrée. Le premier nœud `Add` calcule $x_1 + \epsilon$. Les nœuds nn_1 nn_2 représentent les flots de contrôle de nn_1 et nn_2 inlinés. Enfin, `Concat` prépare les deux entrées requis par nn_2.

Réécriture automatique de graphe

Intégration d'une partie de la spécification WhyML dans le flot de contrôle du réseau

1. support des hyperpropriétés (plusieurs réseaux/traces);
2. génération automatique d'un réseau intégrant la propriété;

Réécritures pour chaque prouveur

```
(**PyRAT, SMTLIB-ish specification **)
;; Goal P3
(assert (<= Y_0 Y_1))
(assert (<= Y_0 Y_2))
(assert (<= Y_0 Y_3))
(assert (<= Y_0 Y_4))

[...]

(**Marabou, prover-specific specification**)
x2 >= 0.49338032358484307238200017309281975030899047851562 5
x3 >= 0.29999999999999998889776975374843459576368331909179687 5
x4 >= 0.29999999999999998889776975374843459576368331909179687 5
+y0 -y1 <= 0
+y0 -y2 <= 0
+y0 -y3 <= 0
+y0 -y4 <= 0
```

Séparation en conjonctions de buts pour les prouveurs qui ne supportent pas la disjonction (oui, ça existe...)

Outils de vérification supportés

Calcul SMT

1. Marabou [Kat+19]
2. Z3 [dB08]
3. CVC5 [Bar+22]
4. Alt-Ergo [Con+18],

Interprétation abstraite

1. PyRAT [LLG24]
2. $\alpha - \beta$–CROWN [Wan+21]
3. Saver [RZ19]

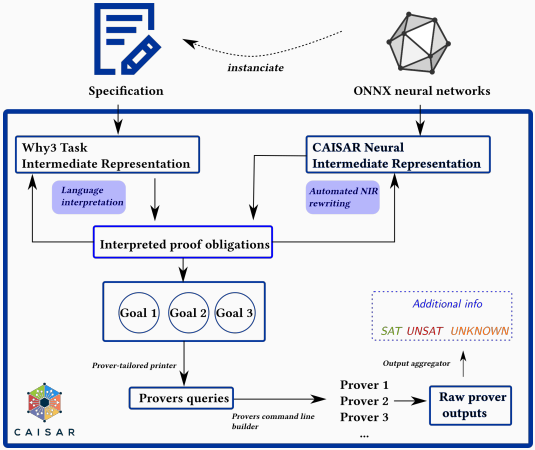
Support d'approches basées test métamorphique (AIMOS [Lem+23])

Support des réseaux de neurones, machines à vecteur de support, machine à boosting de gradient

Ce qu'on a ajouté

1. constructions d'algèbre linéaire classique ;
2. gestion des ensembles d'apprentissage, propriétés non "réellement" universelles ;
3. traducteur du flot de contrôle du programme (réseaux de neurones, SVM, arbres de décision, machine de boosting) vers une représentation intermédiaire et du SMTLIB "classique" ;
4. moteur d'interprétation pour faciliter la décharge de preuve aux prouveurs ;

Architecture de la plateforme



Limitations existantes

- la réécriture impose un surcoût sur certains prouveurs ;
- langage encore centré autours du programme : permettre la génération automatique de WhyML depuis un ODD ?
- plus généralement, comment s'intégrer avec les méthodologies et outils de formalisation des exigences (Papyrus) ?

Travaux en cours

1. composition de la vérification ;
2. spécification des transformations sur les donnée (*pre-processing*) ;
3. explications de décisions [WWB23] ;
4. support du *S-ONNX* ;
5. API Python et communication en mode serveur pour permettre l'intégration (des CI, on peut espérer) ;

Travaux en cours au sein du projet DeepGreen

aidge

DEEPGREEN

Vérification de propriétés telles que l'iso-fonctionnalité entre réseaux sur plusieurs cibles matérielles ($nn_1(x) = nn_2(x)$), qualification de la perte de performance ($\forall x \in \mathscr{D}, \|nn_1(x) - nn_2(x)\| \leq \epsilon$), robustesse face aux transformations sémantiques

CAISAR

Site web : `https://caisar-platform.com/`
Logiciel libre (LGPLv2) : `https://git.frama-c.com/pub/caisar`
Rapport technique : `https://hal.science/hal-03687211`
Offres : `https://caisar-platform.com/positions`

CDD et post-docs à pourvoir ! julien.girard2@cea.fr

La maturation de CAISAR est partiellement financée par les projets PRISSMA, Confiance.ai et Deepgreen

Références

[Bak21] Stanley Bak. **“Nnenum : Verification of ReLU Neural Networks with Optimized Abstraction Refinement”.** In : *NASA Formal Methods*. Sous la dir. d’Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz et Ivan Perez. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2021, p. 19-36. isbn : 978-3-030-76384-8. doi : 10.1007/978-3-030-76384-8_2 (cf. p. 7-11).

Bibliography ii

[Bal+19] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr et Martin Vechev. **“Certifying Geometric Robustness of Neural Networks”.** In : *Advances in Neural Information Processing Systems*. T. 32. Curran Associates, Inc., 2019. URL : https://papers.nips.cc/paper/2019/hash/f7fa6aca028e7ff4ef62d75ed025fe76-Abstract.html (visité le 19/05/2022) (cf. p. 6).

Bibliography iii

[Bar+22] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli et Yoni Zohar. **"cvc5 : A Versatile and Industrial-Strength SMT Solver".** In : *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*. Sous la dir. de Dana Fisman et Grigore Rosu. T. 13243. Lecture Notes in Computer Science. Springer, 2022, p. 415-442. DOI : 10.1007/978-3-030-99524-9_24. URL : https://doi.org/10.1007/978-3-030-99524-9%5C_24