Towards end-to-end ASP computation*

Katsumi Inoue

National Institute of Informatics

TAASP 2025 (TU Wien Informatics, November 25, 2025)

* Joint work with **Taisuke Sato** and **Akihiro Takemura**
(to appear in *Neurosymbolic Artificial Intelligence*)

Outline

1. Introduction: Towards Trustworthy AI
2. Background: Algebraic Approaches to Logic Programming
3. Main: A Framework for Differentiable ASP
4. Supplementary: Tools for Differentiable ASP (unpublished)

Towards Robust Symbolic Reasoning

- **Symbolic reasoning** has been used
  - to derive *logical consequences* of knowledge bases (represented in *logical formulas*);
  - to compute *satisfiable assignments* of specifications (represented as *constraints*).

- Symbolic reasoning ensures the **correctness** of computation in terms of *consistency*, *soundness* and *completeness*, supposing that given knowledge and input data are correct.
- Symbolic reasoning is *explainable* and *interpretable*, which gives a foundation of XAI.
- Logical knowledge and derived theorems can be stored and reused.

- The bottleneck exists in obtaining correct knowledge.
- Reasoning algorithms lack scalability and are not tolerant to noise.
- We often need huge commonsense as background knowledge.

⇨ These weakness could be covered by combining with Machine Learning methods.

Integrating KR and ML for Trustworthy AI

Symbolic/Discrete Space

❖**Knowledge Representation and Reasoning (KR)**

- ***Interpretability***
- ***Explainability***

LLM

Neurosymbolic AI

Bridging Two Spaces

❖Linear-algebraic logic programming
❖Differentiable logic reasoning and learning
❖Incorporating constraints into ML systems

Numeric/Continuous Space

❖**Machine Learning (ML)**

- ***Robustness***
- ***Scalability***

Applications

❖Object detection
❖VQA, NLI, Robotics
❖Biology, Physics, etc.

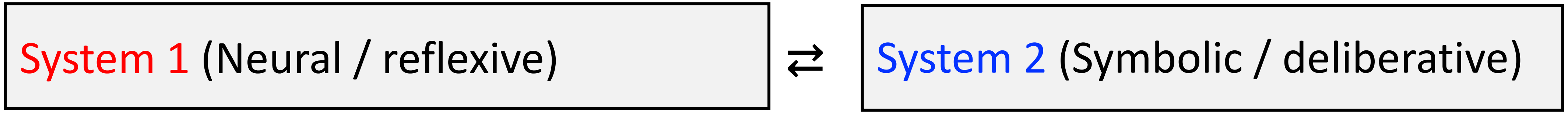
High-speed algebraic computation on GPUs

Discrete/Symbolic ⇄ Continuous/Numeric

- Domains: Boolean, multi-valued/discrete, continuous
- Constraint types: logical, Pseudo-Boolean, linear, non-linear, differential
- Spectrum of search algorithms:
    - Complete: systematic, DPLL, CDCL
    - Local search (grid search): greedy, mixed random walk
    - Large neighborhood search (LNS): several neighborhood definitions
    - Continuous search: cost-minimization, differentiable
- Varieties of optimization methods:
    - Combinatorial: intractable, greedy randomized
    - Continuous: iterative, gradient, Newton
    - Cross-entropy, Evolutional, Quantum, etc.
- Multi-variate time-series data as input
- Multiple variables can be handled simultaneously: Array computing
- Applications to many areas, e.g., XAI, Edge AI, CPS, biology

Neuro(-)symbolic AI (NeSy)

- The popularity of *neuro-symbolic* approaches has been on the rise in recent years, e.g., Artur Garcez et al. (2019); Gary Marcus (2022).
- The goal is to integrate “the two most fundamental aspects of intelligent cognitive behavior” (Leslie Valiant, 2003):
  - the ability to learn from experience, and
  - the ability to reason from what has been learned.
- Analogies have also been drawn with dual process theories in psychology (Daniel Kahneman, 2011; Francesca Rossi, 2022).

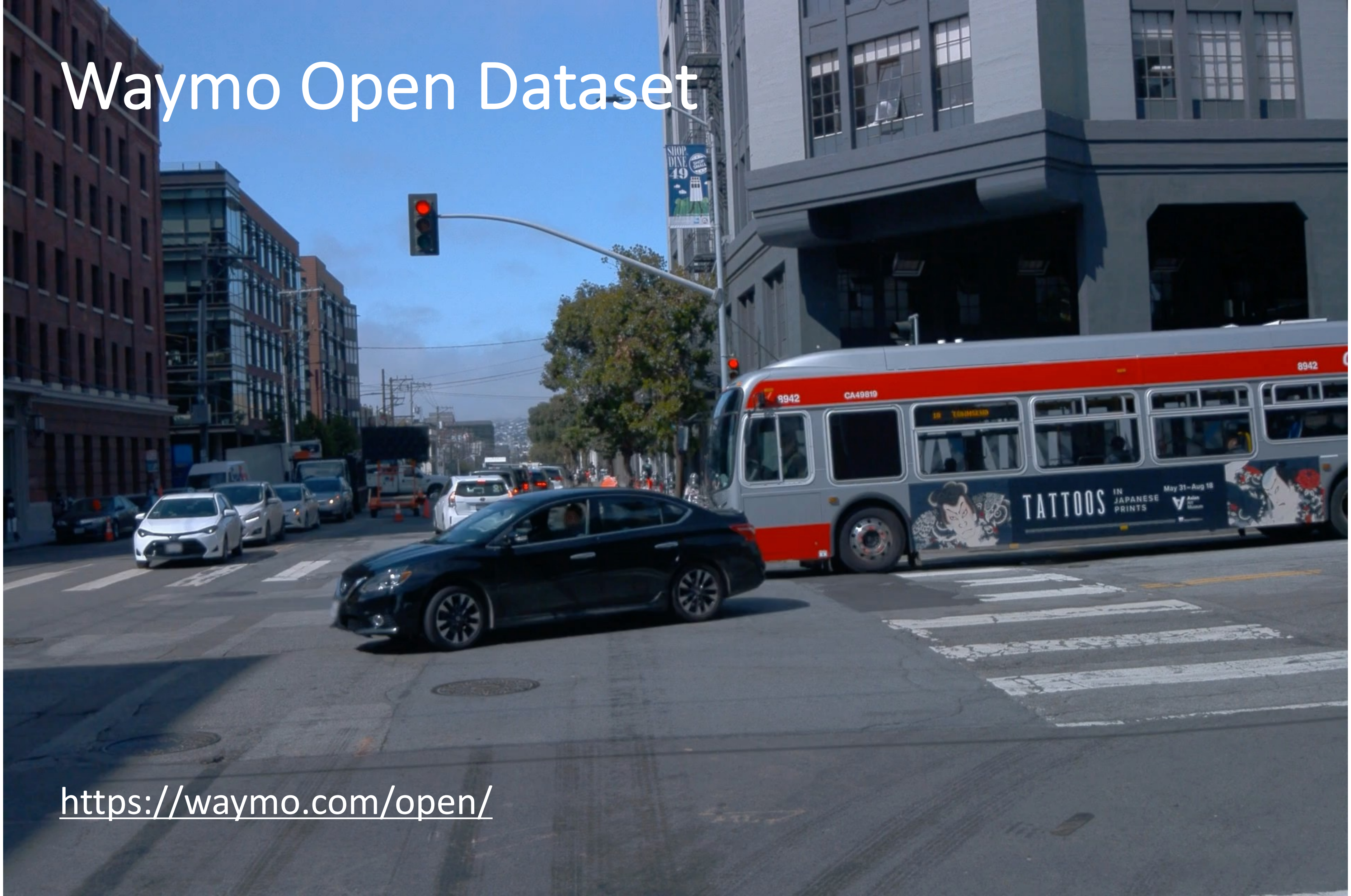


System 1 (Neural / reflexive) ⇄ System 2 (Symbolic / deliberative)

Neurosymbolic reasoning and learning

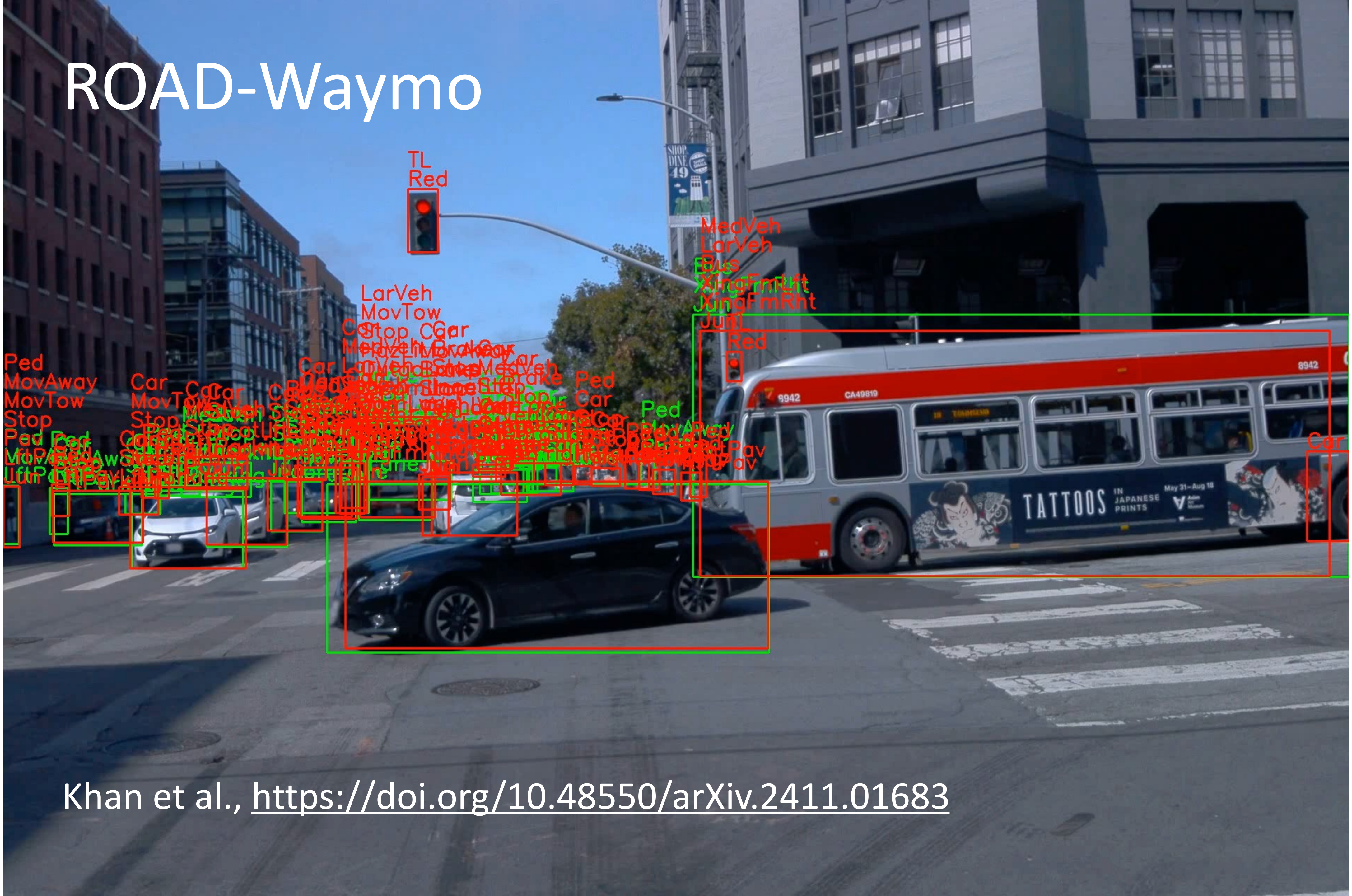
- Explainable models for black-box learning systems
    - symbolic rule extraction from neural networks
    - construction of logic circuits that simulate machine learning systems
- Hybrid systems (popular in NeSy)
    - neural pattern recognition followed by symbolic problem solving
    - verification of machine learning outputs by symbolic reasoning
    - neural pattern recognition enhanced/constrained with symbolic reasoning
- Embedding symbolic knowledge in vector spaces
    - knowledge graph embedding
    - program syntheses, neural/differentiable programming
    - neuro-symbolic reasoning: theorem proving, logic programming, answer set programming, abduction, etc.
    - large language models

Waymo Open Dataset



https://waymo.com/open/

ROAD-Waymo



Khan et al., https://doi.org/10.48550/arXiv.2411.01683

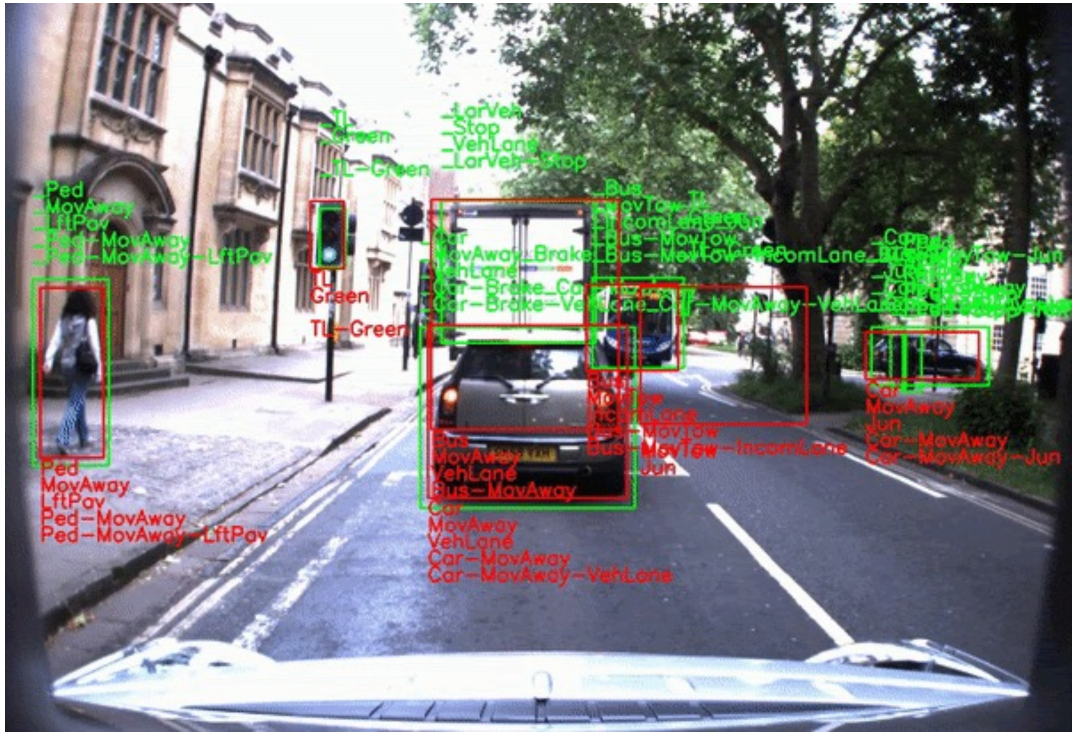
ROAD-R: Autonomous Driving with Requirements

- Multi-label Object Recognition
  - Agent detection

    (*Pedestrian, Car, Cyclist, Emergency-Vehicle* etc.)
  - Action detection

    (*Turning-right, Moving-away, Pushing-objects,* etc.)
  - Location detection

    (*Vehicle-lane, Right-pavement, Bus-stop,* etc.)



https://sites.google.com/view/road-r/dataset

- Requirements (= hard logical constraints)[1]
  - *A traffic light cannot move.*
  - *A traffic light cannot be red and green at the same time.*
  - *If an agent is crossing, it is either a pedestrian or a cyclist.*

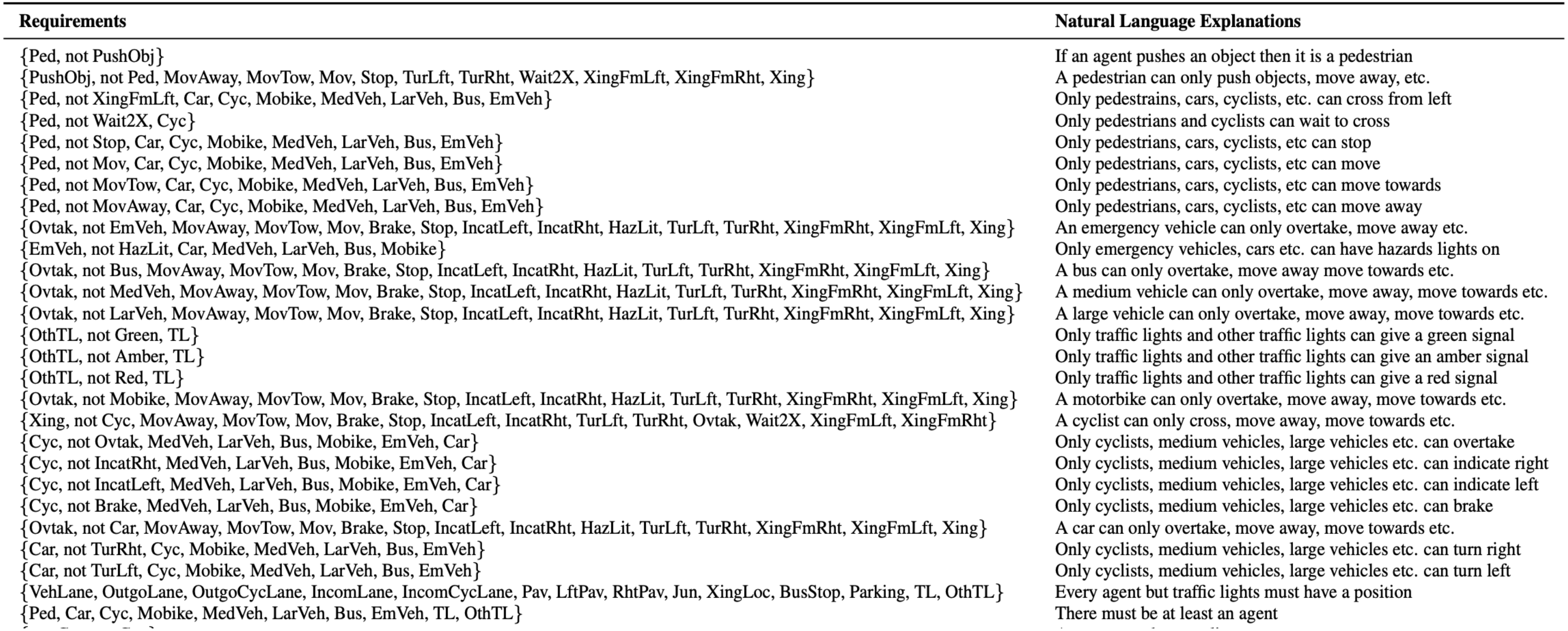
- Methods: Extend pre-trained recognition model, use Partial Weighted MaxSAT

- ROAD-R Challenge for NeurIPS 2023:
  - NII Team Results[2]: Task 2 (supervised) **Won**, Task 1 (semi-supervised) 3rd

[1] Eleonora Giunchiglia, et al.: ROAD-R: the autonomous driving dataset with logical requirements. *Machine Learning*, 112 (2022)

[2] S. Moriyama, K. Watanabe, K. Inoue, A. Takemura: MOD-CL: Multi-label Object Detection with Constrained Loss. arXiv (2024)

Logical Constraints in ROAD-R (all hard constraints)



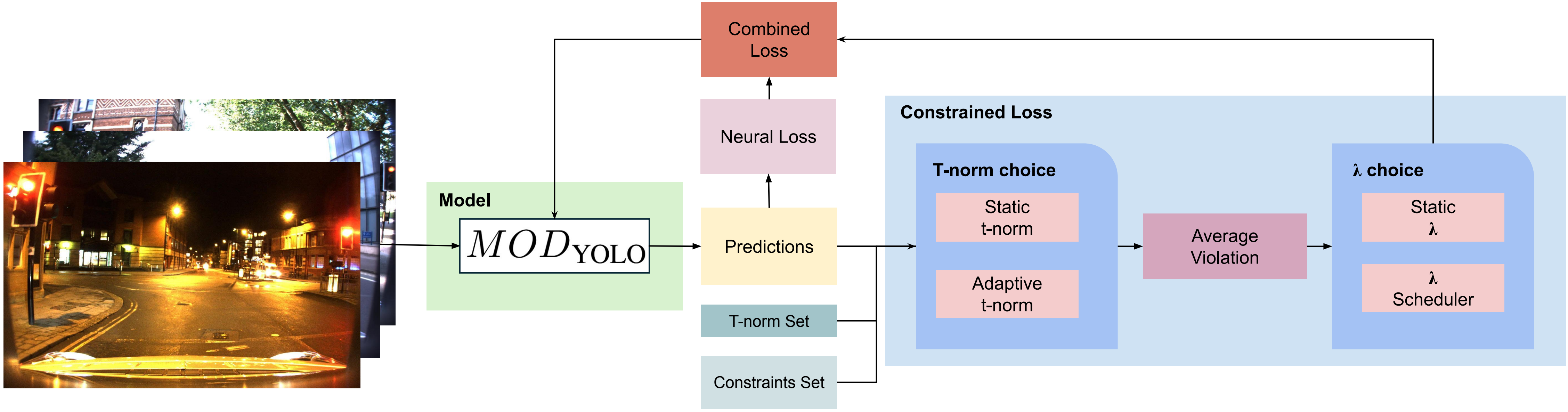
Requirements	Natural Language Explanations
{Ped, not PushObj}	If an agent pushes an object then it is a pedestrian
{PushObj, not Ped, MovAway, MovTow, Mov, Stop, TurLft, TurRht, Wait2X, XingFmLft, XingFmRht, Xing}	A pedestrian can only push objects, move away, etc.
{Ped, not XingFmLft, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrains, cars, cyclists, etc. can cross from left
{Ped, not Wait2X, Cyc}	Only pedestrians and cyclists can wait to cross
{Ped, not Stop, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can stop
{Ped, not Mov, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move
{Ped, not MovTow, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move towards
{Ped, not MovAway, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only pedestrians, cars, cyclists, etc can move away
{Ovtak, not EmVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	An emergency vehicle can only overtake, move away etc.
{EmVeh, not HazLit, Car, MedVeh, LarVeh, Bus, Mobike}	Only emergency vehicles, cars etc. can have hazards lights on
{Ovtak, not Bus, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A bus can only overtake, move away move towards etc.
{Ovtak, not MedVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A medium vehicle can only overtake, move away, move towards etc.
{Ovtak, not LarVeh, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A large vehicle can only overtake, move away, move towards etc.
{OthTL, not Green, TL}	Only traffic lights and other traffic lights can give a green signal
{OthTL, not Amber, TL}	Only traffic lights and other traffic lights can give an amber signal
{OthTL, not Red, TL}	Only traffic lights and other traffic lights can give a red signal
{Ovtak, not Mobike, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A motorbike can only overtake, move away, move towards etc.
{Xing, not Cyc, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, TurLft, TurRht, Ovtak, Wait2X, XingFmLft, XingFmRht}	A cyclist can only cross, move away, move towards etc.
{Cyc, not Ovtak, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can overtake
{Cyc, not IncatRht, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can indicate right
{Cyc, not IncatLeft, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can indicate left
{Cyc, not Brake, MedVeh, LarVeh, Bus, Mobike, EmVeh, Car}	Only cyclists, medium vehicles, large vehicles etc. can brake
{Ovtak, not Car, MovAway, MovTow, Mov, Brake, Stop, IncatLeft, IncatRht, HazLit, TurLft, TurRht, XingFmRht, XingFmLft, Xing}	A car can only overtake, move away, move towards etc.
{Car, not TurRht, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only cyclists, medium vehicles, large vehicles etc. can turn right
{Car, not TurLft, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh}	Only cyclists, medium vehicles, large vehicles etc. can turn left
{VehLane, OutgoLane, OutgoCycLane, IncomLane, IncomCycLane, Pav, LftPav, RhtPav, Jun, XingLoc, BusStop, Parking, TL, OthTL}	Every agent but traffic lights must have a position
{Ped, Car, Cyc, Mobike, MedVeh, LarVeh, Bus, EmVeh, TL, OthTL}	There must be at least an agent

Challenges:

1. Can these constraints help learning with small amount of training data?
2. How can hard constraints be 100% satisfied using neurosymbolic methods?

Adaptive Object Detection for ROAD-R/Waymo

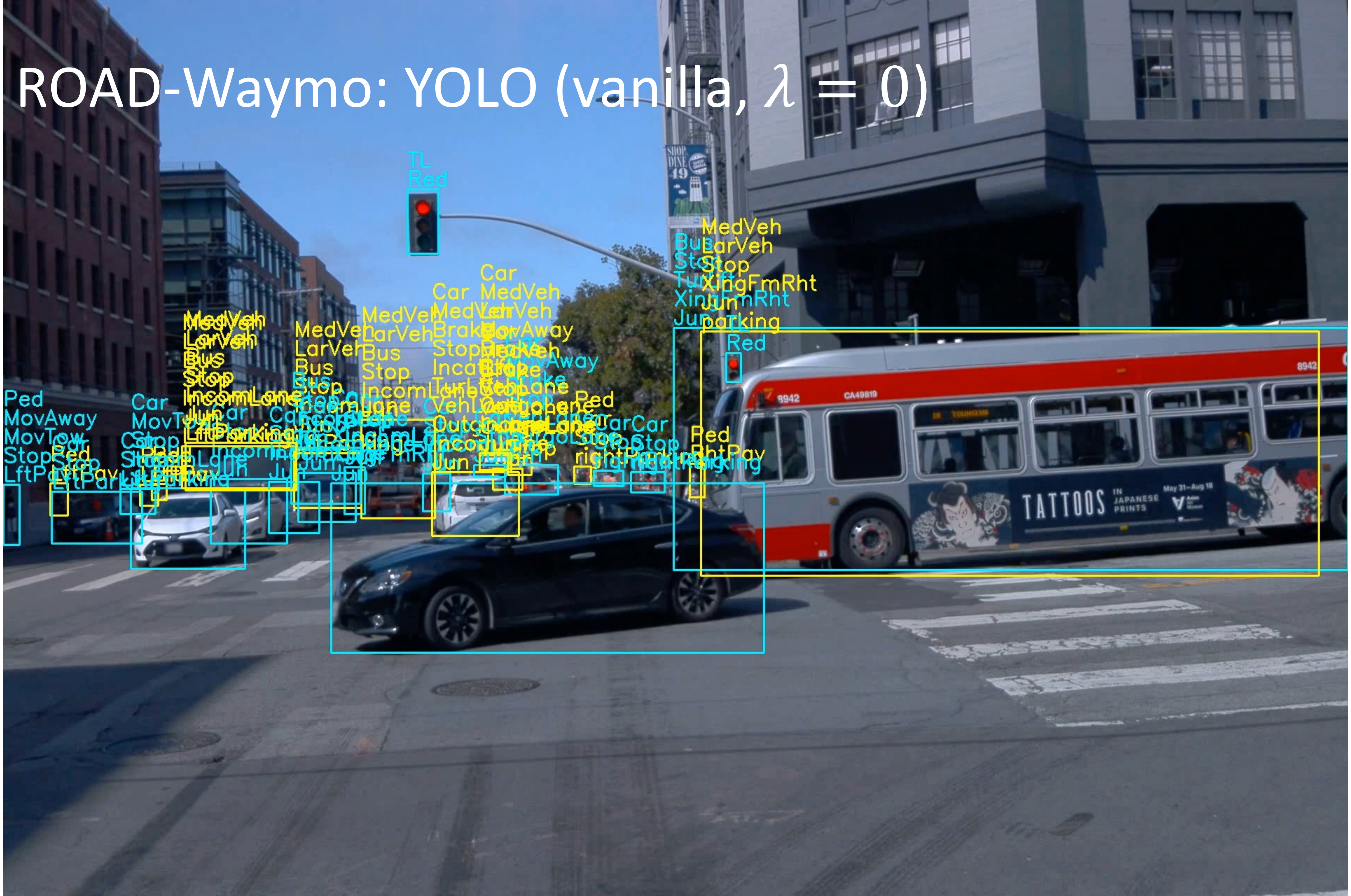
(T. Eiter, N. Higuera, K. Inoue, S. Moriyama, *NeurIPS 2025*)



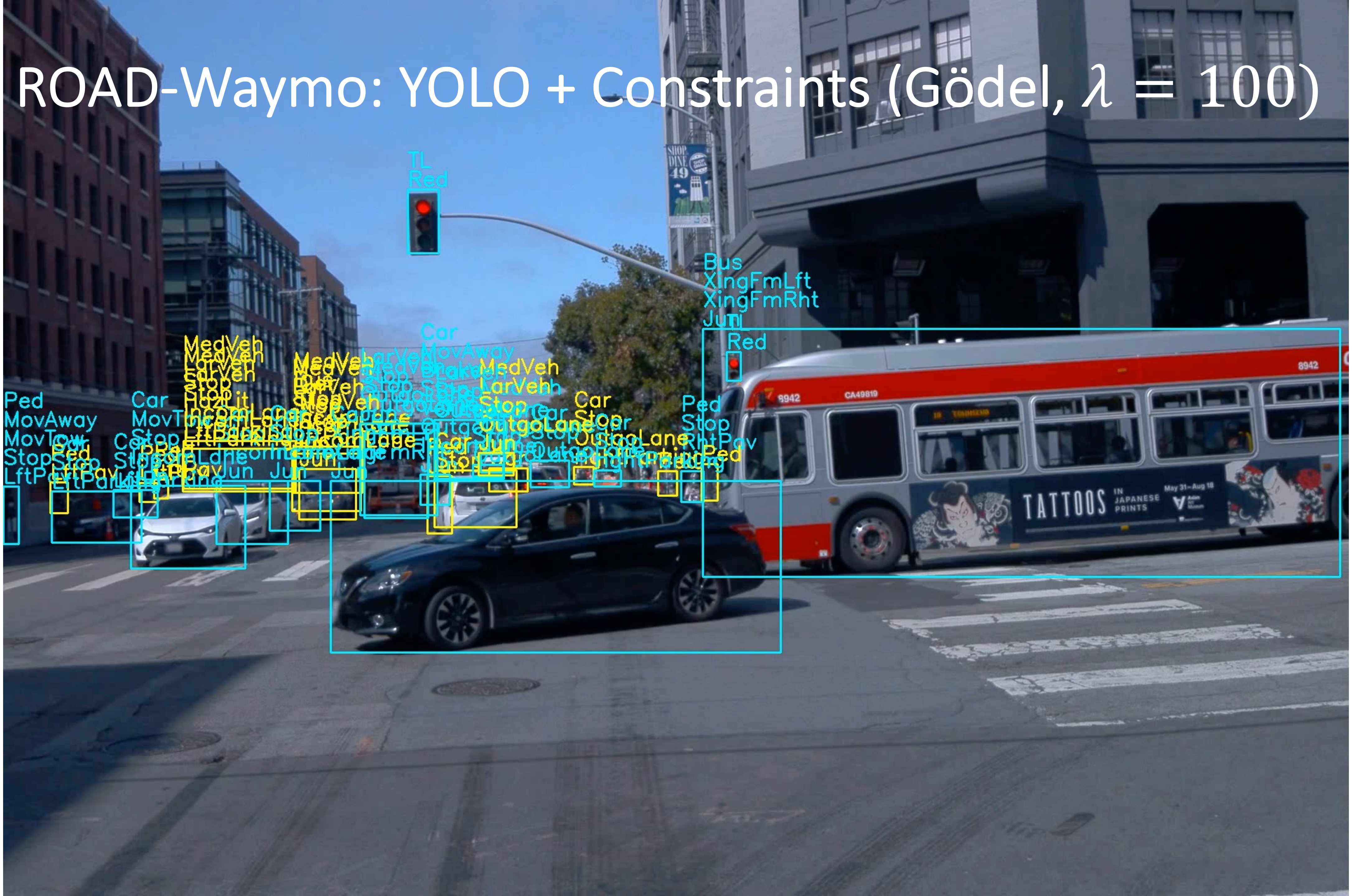
$$L = L_{neural} + \lambda \cdot L_{constrained}$$

- extends MOD-CL, the winning model of ROAD-R Challenge for NeurIPS 2023
- seamless integration of the constrained loss into object detection models
- adaptive selection of 12 t-norms of fuzzy logic in evaluating constrained loss
- dynamic change of λ (constraint satisfaction degree) by regularization scheduling

ROAD-Waymo: YOLO (vanilla, $\lambda = 0$)



ROAD-Waymo: YOLO + Constraints (Gödel, $\lambda = 100$)



Outline

1. Introduction: Towards Trustworthy AI
2. Algebraic Approaches to Logic Programming
3. A Framework for Differentiable ASP
4. Tools for Differentiable ASP (unpublished)

Algebraic approach to logic programming

- Linear algebraic approaches to logic programming contribute to a step toward realizing robust and scalable logical inference.
  1. Matrix-vector product methods are used for exact computation, which can be scalable, and are the basis for the differentiable method.
  2. Differentiable methods are used for approximate computation, which can be robust to noise, and are connected to machine learning.

- Machine learning of logic programs can be realized by computing matrix/tensor representation of programs from input-output pairs.

Logical inference in vector spaces, I

—*Linear-algebraic* methods (Sakama, Inoue & Sato, 2021)

- **Common Principle:**
  - **Representation (encoding):** formulate logical formulas as vectors/matrices/tensors
  - **Computation:** apply linear algebraic operations on these elements

- P : (logic) program, constraints ⥱ matrix $\boldsymbol{M}_P$
- I : assignment/interpretation ⥱ vector $\boldsymbol{v}_I$
- $J = \mathrm{T}_P(I) = \{ h \mid (h \leftarrow b_1 \& \ldots \& b_m) \in P, \{b_1,\ldots,b_m\} \subseteq I \}$: immediate consequences

  ⥱ vector $\boldsymbol{v}_J = \theta(\boldsymbol{M}_P \boldsymbol{v}_I)$, where θ is a binary threshold function

- **Expected:**
  - High performance computation based on the sparsity of matrices (Nguyen, Inoue & Sakama, 2022)
  - Parallelism by GPU computation + partial evaluation (poss. exponential speedup)

➢ Chiaki Sakama, Katsumi Inoue, Taisuke Sato: "Logic programming in tensor spaces", *AMAI*, **89**:1133-1153 (2021).

Logical inference in vector spaces, II

—*Continuous/differentiable methods* (Sato & Kojima 2019)

- **Common Principle:**
  - Set a loss function *L*
  - Formulate a problem as cost minimization of *L* with parameter tensor ***x***
  - Compute a minimum ***x*** of *L* by SGD/Newton's method
  - if *L*(***x***) = 0, then ***x*** is a solution
  - Threshold ***x*** to a binary tensor representing a logical solution

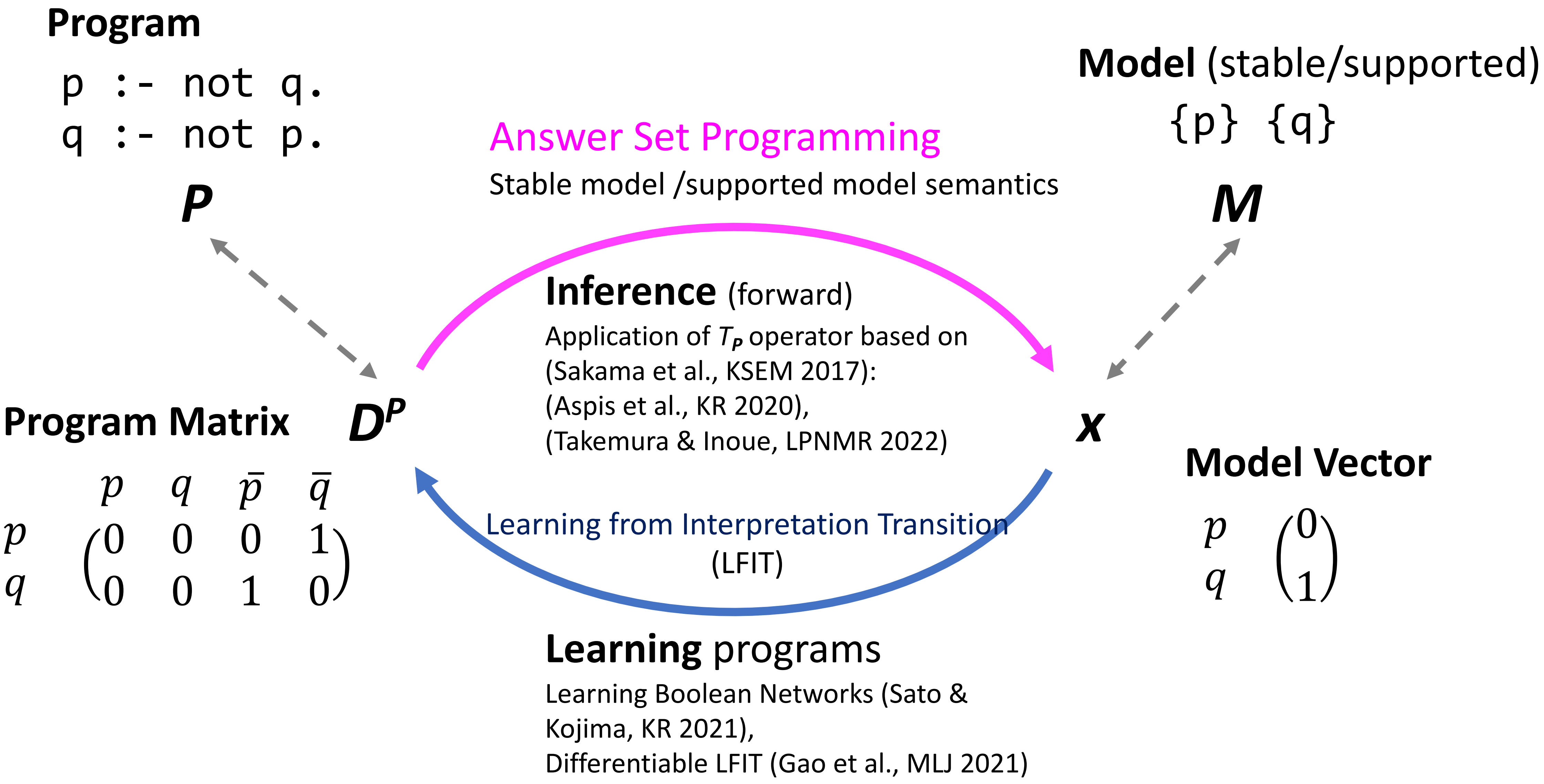
$$\boxed{\frac{\partial L(\boldsymbol{x})}{\partial \boldsymbol{x}}}$$

Gradient of *L*(***x***)

- **Expected:**
  - Robustness by continuity
  - Scalability by multi-core/GPU parallelism
  - Smoothness to combine with neural systems

➢ Sato T., Kojima R.: "Logical Inference as Cost Minimization in Vector Spaces", *IJCAI 2019 International Workshops*, LNAI **12158**, pp.239-255 (2020).

Differentiable reasoning & learning in vector spaces



Program

```
p :- not q.
q :- not p.
```

P

Model (stable/supported)

```
{p} {q}
```

M

Answer Set Programming

Stable model /supported model semantics

Inference (forward)

Application of T_P operator based on (Sakama et al., KSEM 2017): (Aspis et al., KR 2020), (Takemura & Inoue, LPNMR 2022)

Program Matrix $\boldsymbol{D^P}$

$$\begin{array}{c} \\ p \\ q \end{array} \begin{array}{cccc} p & q & \bar{p} & \bar{q} \\ \end{array}$$

$$\begin{array}{c|cccc} & p & q & \bar{p} & \bar{q} \\ \hline p & 0 & 0 & 0 & 1 \\ q & 0 & 0 & 1 & 0 \end{array}$$

$\boldsymbol{x}$

Model Vector

$$\begin{array}{cc} p & \multirow{2}{*}{} \\ q & \end{array} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Learning from Interpretation Transition (LFIT)

Learning programs

Learning Boolean Networks (Sato & Kojima, KR 2021), Differentiable LFIT (Gao et al., MLJ 2021)

Differentiable computation of supported models

1. Embed a logic program ***P*** into a Program Matrix ***D^P***

Program

P

```
p :- p.
q :- not p.
```

{p} and {q} are supported, but only {q} is stable

Program Matrix

$$\boldsymbol{D^P} \quad \begin{array}{c|cccc} & p & q & \bar{p} & \bar{q} \\ \hline p & 1 & 0 & 0 & 0 \\ q & 0 & 0 & 1 & 0 \end{array}$$

Interpretation vector

$$\boldsymbol{x} \quad \begin{array}{c|c} p & 0 \\ q & 0 \end{array}$$

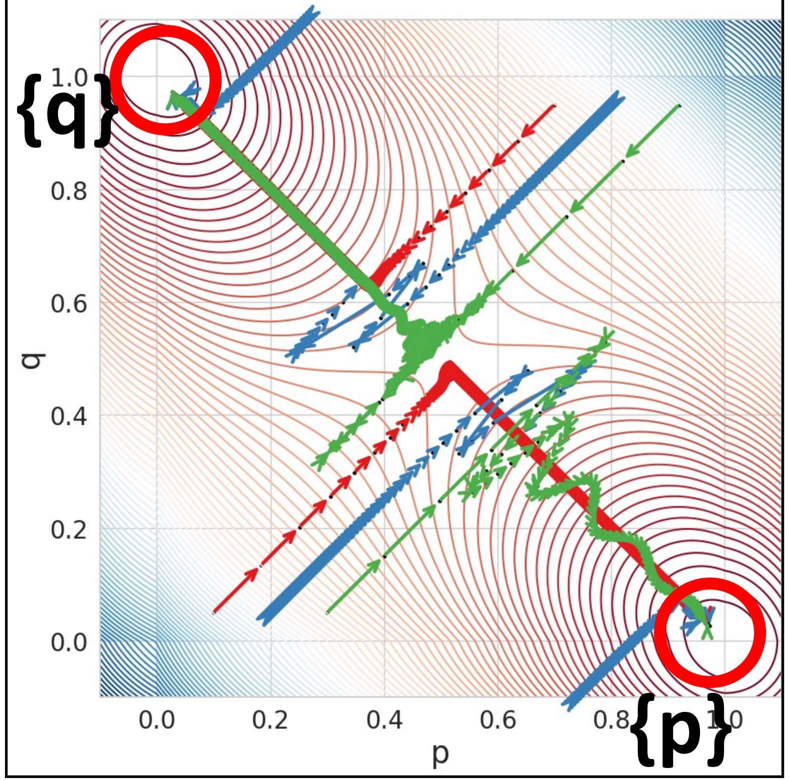
2. Define Loss function w.r.t. continuous-valued interpretation such that *Loss* = 0 corresponds to an intended model of ***P***

$$L(\boldsymbol{x}) \longleftrightarrow \frac{\partial L(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

Loss function

Gradient of $L(\boldsymbol{x})$

3. Minimize the loss with gradient descent, to reach supported models



Takemura, A. & Inoue, K. (2022). Gradient-Based Supported Model Computation in Vector Spaces. LPNMR 2022.

Differentiable computation of stable models [this talk]

1. Embed a logic program ***P*** into a Program Matrix ***D^P***

Program

P

```
p :- p.
q :- not p.
```

{p} and {q} are supported, but only {q} is stable

Program Matrix

D^P

$$\begin{array}{c|cccc} & p & q & \bar{p} & \bar{q} \\ \hline p & 1 & 0 & 0 & 0 \\ q & 0 & 0 & 1 & 0 \end{array}$$

Interpretation vector

x

$$\begin{array}{c|c} p & 0 \\ q & 0 \end{array}$$

2. Define Loss function w.r.t. continuous-valued interpretation such that *Loss* = 0 corresponds to models of ***P***

$$L(\boldsymbol{x}) \longleftrightarrow \frac{\partial L(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

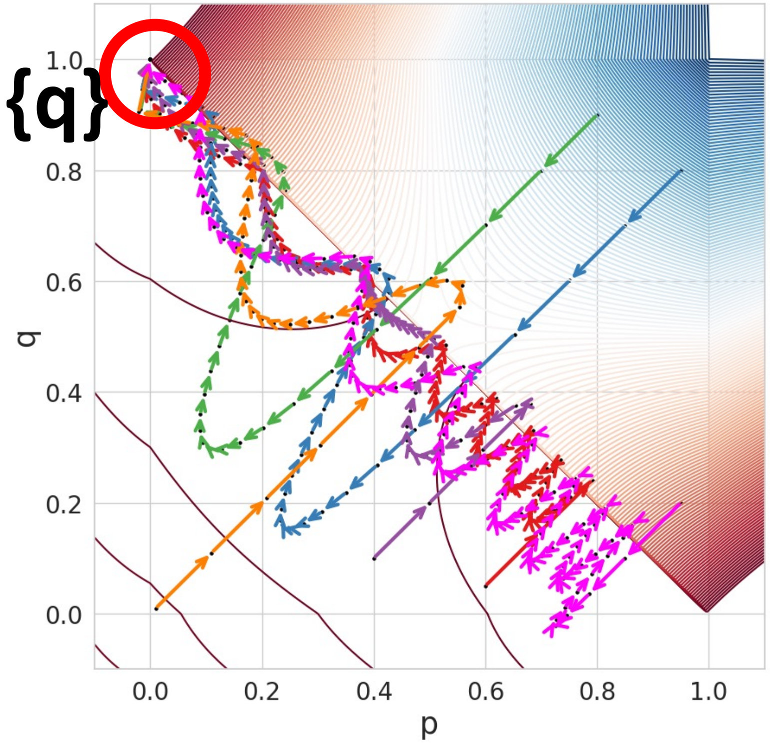
Loss function

Gradient of *L*(***x***)

Semantically inspired checks

- ✓ Supported model
- ✓ Unfounded set
- ✓ Loop formulas

3. Minimize the loss with gradient descent, to reach stable models



➢ Sato, T., Takemura, A. & Inoue, K.: Towards end-to-end ASP computation, arXiv:2306.06821, 2023.

Loss function (is exactly 0 when *x* is a supported model)

- Given
  - $\boldsymbol{D}^P$: program matrix, shape: [|**Heads**|, |$\mathbf{B_{P+}}$|] #|$\mathbf{B_{P+}}$| = 2·|Heads|
  - $\boldsymbol{x}$: candidate interpretation vector, shape: [|$\mathbf{B_{P+}}$|, 1]
  - $||\boldsymbol{x}||_F$: Frobenius norm (2-norm)
  - F_θ: thresholding function (parameterized θ-thresholding)

$$L(\boldsymbol{x}) = \frac{1}{2}\{\underbrace{\|F_\theta\,(\boldsymbol{D}^P[\boldsymbol{x};\mathbf{1}-\boldsymbol{x}]) - \boldsymbol{x}\|_F^2} + \lambda_1\underbrace{\|\boldsymbol{x}\odot(\boldsymbol{x}-1)\|_F^2} + \lambda_2\underbrace{\|\boldsymbol{f}-(x\odot\boldsymbol{f})\|_F^2\}}$$

1\.

When ***m*** is a supported model, $T_P(\boldsymbol{m})=\boldsymbol{m}$

When this term is 0, we have $F_\theta(\boldsymbol{D}^P\boldsymbol{x}) = \boldsymbol{x}$

2\.

Pruning fractional interpretations

(0 if all elements are 0 or 1)

3\.

Penalty for ‘forgetting’ facts

(0 if assignments on facts do not change)

Logical reasoning realized in vector spaces (in our group)

first-order (FO) deduction (Sato, **TPLP 2017**)

FO abduction (Sato, Inoue & Sakama, **IJCAI 2018**)

logic programming (LP) fixpoint computation (Sakama, Inoue & Sato, **KSEM 2017**; **AMAI 2021**)

Sparse method for LP (Nguyen, Inoue & Sakama, **ICLP 2021**; **NGC 2022**)

ASP (supported models) (Sato, Inoue & Sakama, **ICAART 2020**)

LP abduction (Nguyen, Inoue & Sakama, **ICTAI 2021**; **PADL 2023**; **ICTAI 2024**)

differentiable ASP (supported models) (Takemura & Inoue, **LPNMR 2022**; **ECAI 2024**)

differentiable ASP (stable models) (Sato, Takemura & Inoue, arXiv 2023; **NSAI 2025**)

SAT (MatSat) (Sato & Kojima, **PoS 2021**)

Boolean network learning (Sato & Kojima, **KR 2021**)

differentiable LFIT (transformer-based) (Phua & Inoue, **ILP 2019**; **ILP 2021**; **NeSy 2024**)

differentiable LFIT (matrix learning) (Gao, Wang, Cao & Inoue, **MLJ 2022**)

induction of FO LP (Gao, Inoue, Cao & Wang, **IJCAI 2022**; **AIJ 2024**)

DNF learning (Sato & Inoue, **MLJ 2023**)

differentiable rule learning from real-valued time-series data (Gao, Inoue, Cao, Wang & Yang, **ICLR 2025**)

Similarities between minimization tasks

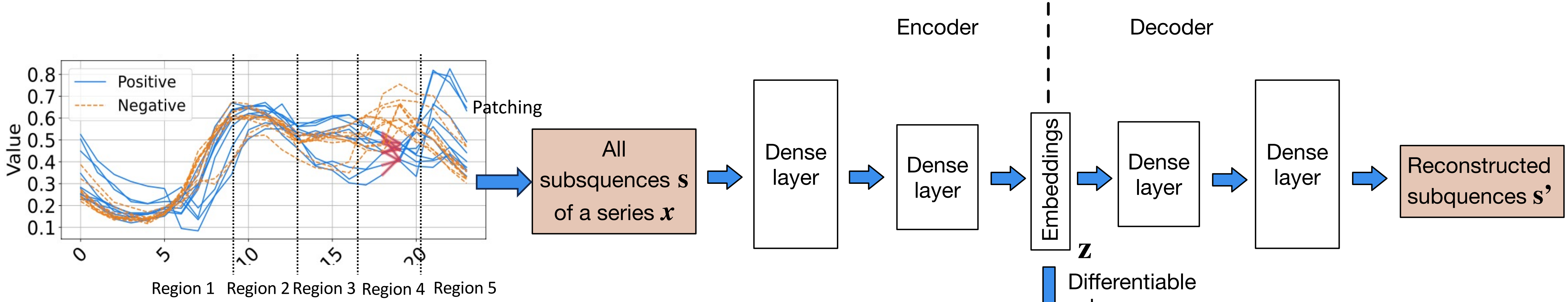
task	minimize …	*X* / *x* is ….
Matrix decomposition	$\lVert X\ - \boldsymbol{A}\ \boldsymbol{B}\ \rVert_F^2$	0-1 matrix ‘relation’
Relation abduction [Sato+ 2018]	$\lVert R_1 - R_3\boldsymbol{X}\rVert_F^2$	0-1 matrix ‘relation’
Satisfiability [Sato & Kojima 2018]	$\lVert 1 - t(Q\ [\boldsymbol{x}; 1 - \boldsymbol{x}])\rVert_F^2$	0-1 vector ‘assignment’
Supported model [Takemura & Inoue 2022]	$\lVert t(P[\boldsymbol{x}; 1 - \boldsymbol{x}]) - \boldsymbol{x}\rVert_F^2$	0-1 vector ‘interpretation’
Supported model (N.B.: This term does not check for unfounded sets)	$\lVert t(D^T t'(P[\boldsymbol{x}; 1 - \boldsymbol{x}]) - \boldsymbol{x}\rVert_F^2$	0-1 vector ‘interpretation’

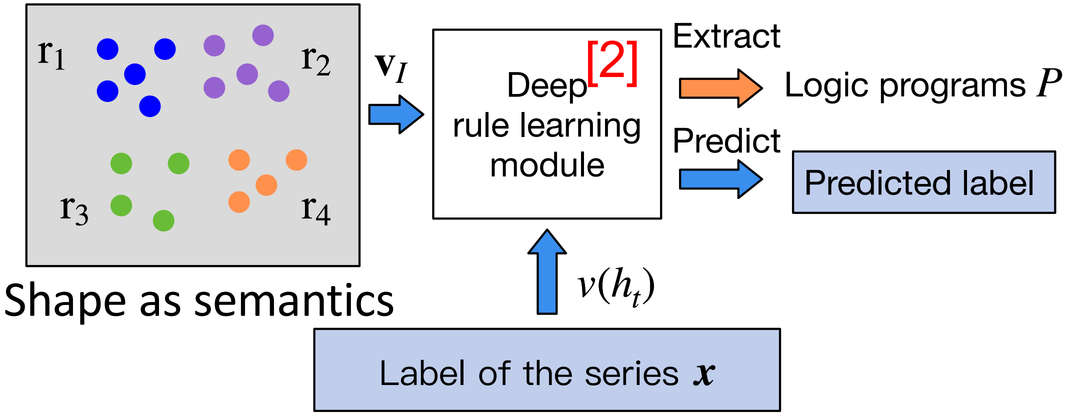
Sato, T., Inoue, K., & Sakama, C. (2018). Abducing Relations in Continuous Spaces. IJCAI 18 https://doi.org/10.24963/ijcai.2018/270

Sato, T., & Kojima, R. (2020). Logical Inference as Cost Minimization in Vector Spaces. IJCAI 19 Workshops https://doi.org/10.1007/978-3-030-56150-5_12

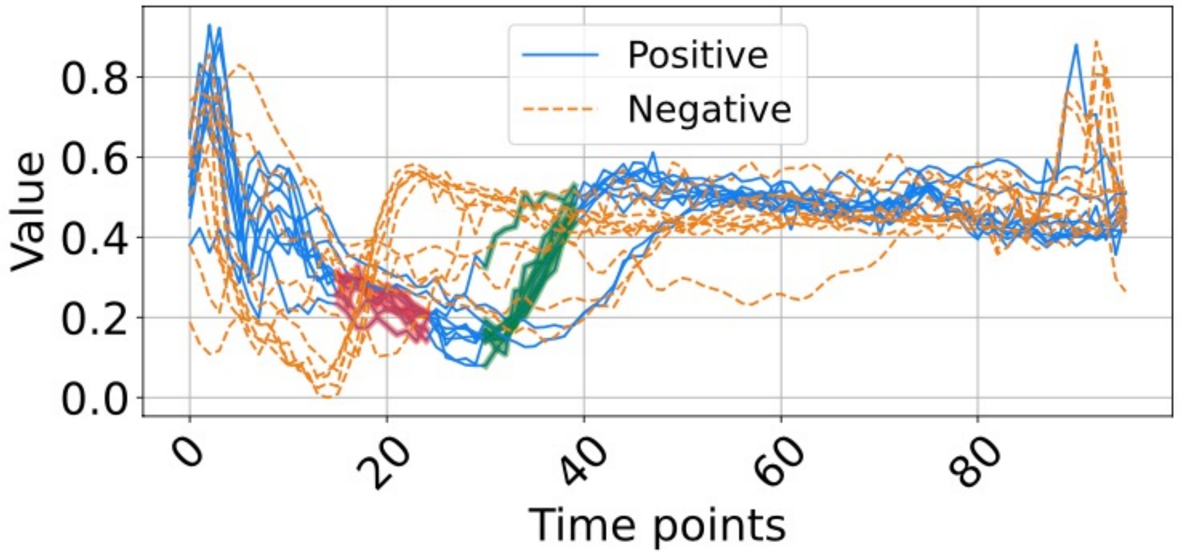
Takemura, A., & Inoue, K. (2022). Gradient-Based Supported Model Computation in Vector Spaces. LPNMR 2022.

Differentiable rule induction from raw time series data[1]





$h_p \leftarrow \text{pattern}_2(X) \wedge \text{region}_1(X) \wedge \text{pattern}_1(Y) \wedge \text{region}_2(Y)$ $(p = 0.83,\ r = 0.89)$



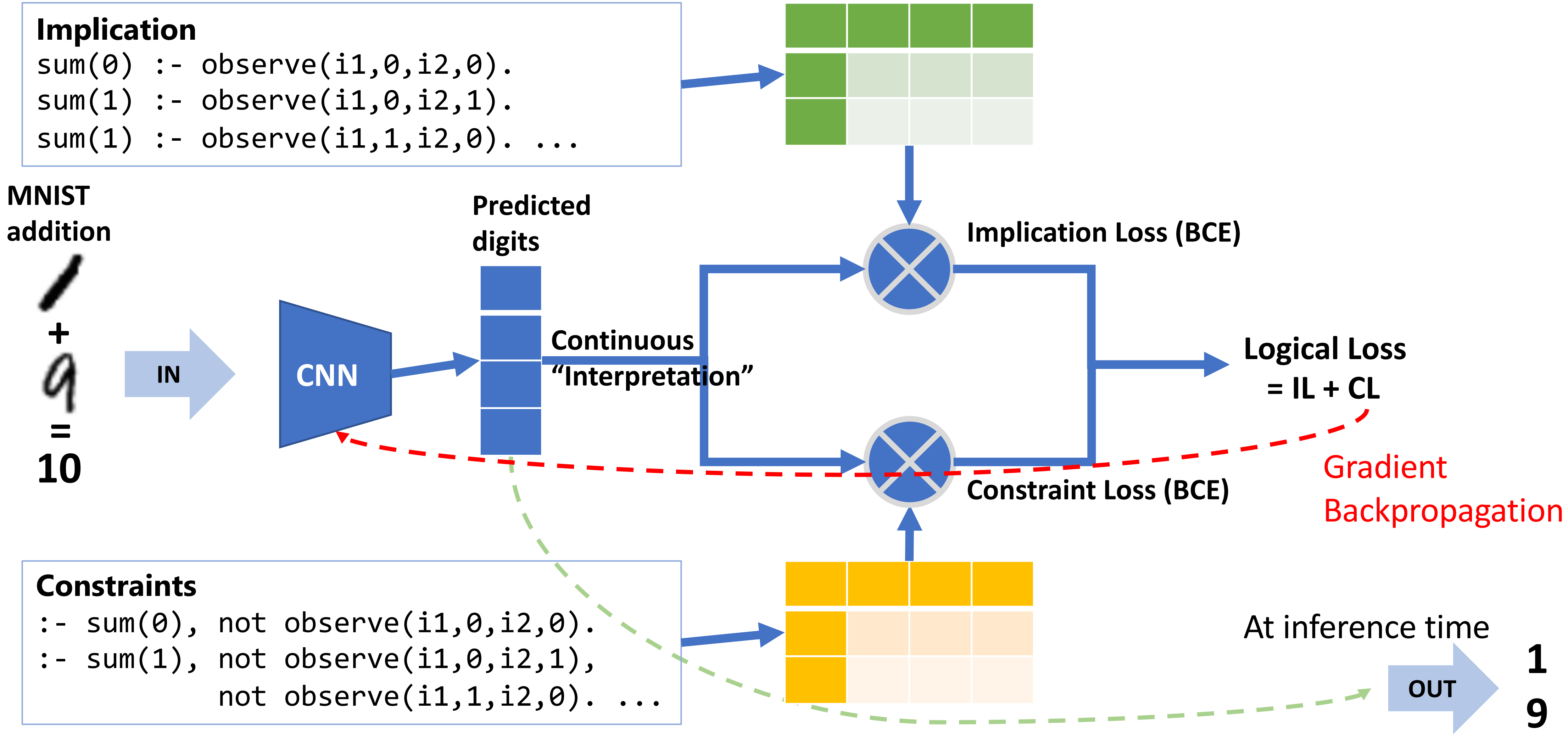
[1] K. Gao, K. Inoue, Y. Cao, H. Wang, F. Yang: *ICLR 2025*

[2] K. Gao, K. Inoue, Y. Cao, H. Wang*: IJCAI 2022, AIJ 2024*

Application viewpoints

- We have shown that logical computation can be transformed to numeric computation using algebraic representation.
- The methods have some effects on purely symbolic domains, e.g., random instances whose solving heuristics are not well-known.
- But they are more effective on in uncertain environments, in which errors often occur. Then we can construct **robust** reasoning systems.
- Other expected domains exist on such **interfaces between low-level perception and high-level reasoning** in neurosymbolic fields.

Loss functions for NeSy tasks with embedded logic programs



Implication

```
sum(0) :- observe(i1,0,i2,0).
sum(1) :- observe(i1,0,i2,1).
sum(1) :- observe(i1,1,i2,0). ...
```

Constraints

```
:- sum(0), not observe(i1,0,i2,0).
:- sum(1), not observe(i1,0,i2,1),
           not observe(i1,1,i2,0). ...
```

➢ Akihiro Takemura, Katsumi Inoue: "Differentiable Logic Programming for Distant Supervision", *ECAI 2024.*

Outline

1. Introduction: Towards Trustworthy AI
2. Background: Algebraic Approaches to Logic Programming
3. Main: A Framework for Differentiable ASP
4. Supplementary: Tools for Differentiable ASP (unpublished)

Answer Set Programming (ASP)

- A declarative approach to combinatorial problems
- A problem is specified by a logic program *P*
- A solution (answer set) is a set of ground atoms representing a stable model of *P*
- There are many applications: planning, diagnosis, robotics, NLP, KG etc
- Potasco project (https://potassco.org/) has been main driving force in developing ASP systems

- In neuro-symbolic AI, ASP has been used for symbolic representation and reasoning

Stable model computation

- Stable model semantics [Gelfond & Lifschitz 1988]
  - Smodels [Niemelä & Simons 1997]: bottom-up backtracking search
- SAT solver based
  - ASSAT [Lin & Zhao 2004]: incremental loop formula test
  - Cmodels [Lierler 2005]: disjunctive adaption of ASSAT
- CDNL (conflict driven nogood learning)
  - clasp [Gebser+ 2007]: generalization of CDCL
- Neural combined approach
  - ∂ASP/SAT [Nickles 2018]: ASP solver + decision literal by cost function
  - NeurASP [Yang+ 2020]: ASP solver + (neural atoms + soft-max) + NN
  - SLASH [Skryagin+ 2021]: similar to NeurASP + probabilistic circuit
- Supported model computation by matrix encoding
  - [Aspis+ 2020]: MD condition + cost function (quadratic polynomial, sigmoid)
  - [Takemura & Inoue 2022]: SD condition + cost function (quadratic polynomial, ReLU-like)
- No end-to-end approach to stable model computation exists

End-to-end ASP

- We reformulate stable model computation for propositional normal logic programs in vector spaces and compute stable models by minimizing a cost function
- Unlike [Aspis+ 2020] and [Takemura & Inoue 2022], which compute **supported models**, we compute **stable models** by
  - incorporating **constraints** and **loop formulas**
  - imposing no restriction on the syntactic form of programs such as the *MD condition* [Sakama, Inoue & Sato 2017] and the *SD condition* [Sakama, Inoue & Sato 2021]
- We compute a root ***u*** of a non-negative cost function L^{Su} by Newton's method
  - L^{Su} is derived from *strong disjunction* min(x+y,1) in **Łukasiewicz** (real valued) **logic**:
  - $v\,(x \oplus y) = \min(1, v\,(x) + v\,(y)) = \min_1(v\,(x) + v\,(y))$

Matricized program $\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$

- $\boldsymbol{P} = \begin{cases} \text{p :- q \& \textasciitilde r.} \\ \text{p :- \textasciitilde q \& s.} \\ \text{q.} \end{cases}$

$$\text{comp}(\boldsymbol{P}) = \begin{cases} \text{p} \Leftrightarrow (\text{q \& \textasciitilde r}) \vee (\text{\textasciitilde q \& s}) \\ \text{q} \Leftrightarrow \text{() : empty body} \\ \text{r} \Leftrightarrow \text{\{\} : empty disjunction} \\ \text{s} \Leftrightarrow \text{\{\} : empty disjunction} \end{cases}$$

- $\boldsymbol{C} = \begin{array}{cccc|cccc} \text{p} & \text{q} & \text{r} & \text{s} & \text{\textasciitilde p} & \text{\textasciitilde q} & \text{\textasciitilde r} & \text{\textasciitilde s} \\ \hline \underbrace{\begin{matrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}}_{C^{pos}} & & & & \underbrace{\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}}_{C^{neg}} & & & \end{array}$

- p's 1st rule has body q & ~r
- p's 2nd rule has body ~q & s
- q's 1st rule has empty body (unit clause)
- r has no rule
- s has no rule

- $\boldsymbol{D} = \begin{array}{ccccc} \text{p} & \text{p} & \text{q} & \text{r} & \text{s} \\ \hline 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$

- p has two rules $\boldsymbol{C}(1,:) \vee \boldsymbol{C}(2,:)$
- q has one rule $\boldsymbol{C}(3,:)$
- r has no rule
- s has no rule

Supported model

- Given a normal logic program $\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$, compute $\boldsymbol{P}$'s supported models $\boldsymbol{s}$ in vector spaces
- $\boldsymbol{s}$ is possibly a stable model of $\boldsymbol{P}$
- Put $\boldsymbol{C} = [\boldsymbol{C}^{pos}\ \boldsymbol{C}^{neg}]$, where $\boldsymbol{C}^{pos}$: positive part of $\boldsymbol{C}$, $\boldsymbol{C}^{neg}$: negative part of $\boldsymbol{C}$
- Let $\boldsymbol{s}_I$ be a binary vector as interpretation $\boldsymbol{I}$ for $\boldsymbol{P}$

$\boldsymbol{M} = \mathbf{1} - \min_1(\boldsymbol{C}^{pos}(\mathbf{1} - \boldsymbol{s}_I) + \boldsymbol{C}^{neg}\boldsymbol{s}_I)$: truth value of rule bodies by $\boldsymbol{s}_I$

$\boldsymbol{dS} = \min_1(\boldsymbol{DM})$: truth value of disjunctive rule bodies by $\boldsymbol{s}_I$

- $\boldsymbol{dS} = \boldsymbol{s}_I$ iff $\boldsymbol{s}_I$ is a model of comp($\boldsymbol{P}$)

  iff $\boldsymbol{s}_I$ is a supported model of $\boldsymbol{P}$

Cost function L^{Su} and its Jacobian J_L^{Su}

- $\boldsymbol{dS} = \min_1(\boldsymbol{DM})$, $\boldsymbol{M} = \mathbf{1} - \min_1(\boldsymbol{C}^{pos}(\mathbf{1} - \boldsymbol{s}_l) + \boldsymbol{C}^{neg}\boldsymbol{s}_l)$
- $L^{\mathrm{Su}} = (1/2)\cdot\| \boldsymbol{dS} - \boldsymbol{s}_l \|^2 + (1/2)\cdot \ell_2 \cdot \| \boldsymbol{s}_l \odot (\mathbf{1} - \boldsymbol{s}_l) \|^2 \quad (\ell_2 > 0)$
- Let $\boldsymbol{E} = \boldsymbol{dS} - \boldsymbol{s}_l$ and $\boldsymbol{F} = \boldsymbol{s}_l \odot (\mathbf{1} - \boldsymbol{s}_l)$. Then,
- $L^{\mathrm{Su}} = (1/2)\cdot\| \boldsymbol{E} \|^2 + (1/2)\cdot \ell_2 \cdot \| \boldsymbol{F} \|^2$
- $L^{\mathrm{Su}} = 0$ iff $\boldsymbol{dS} = \boldsymbol{s}_l$ and $\boldsymbol{s}_l$ is binary

  iff $\boldsymbol{s}_l$ is a supported model of $\boldsymbol{P} = \boldsymbol{(C, D)}$
- $J_L^{\mathrm{Su}} = \left(\dfrac{\partial(\boldsymbol{E}\cdot\boldsymbol{E})}{\partial \boldsymbol{s}_l}\right) + \ell_2 \cdot \left(\dfrac{\partial(\boldsymbol{F}\cdot\boldsymbol{F})}{\partial \boldsymbol{s}_l}\right)$

  $= (\boldsymbol{C}^{pos} - \boldsymbol{C}^{neg})^{\mathrm{T}}\,[\boldsymbol{N} \leq \mathbf{1}] \odot (\boldsymbol{D}^{\mathrm{T}}([(\boldsymbol{DM}) \leq \mathbf{1}] \odot \boldsymbol{E}))) - \boldsymbol{E}$

  $+ \ell_2 \cdot ((\mathbf{1} - 2\cdot \boldsymbol{s}_l) \odot \boldsymbol{F})$, where $\boldsymbol{N} = \boldsymbol{C}^{pos}(\mathbf{1} - \boldsymbol{s}_l) + \boldsymbol{C}^{neg}\boldsymbol{s}_l$

Constraints and L^{c}

- $\hat{\boldsymbol{C}} = [\hat{\boldsymbol{C}}^{pos}\ \hat{\boldsymbol{C}}^{neg}]$ represents a set of (integrity) constraints
- Example: $\boldsymbol{C} = \begin{cases} :\!-\ \ a\ \&\ {\sim}b. \\ :\!-\ \ b\ \&\ {\sim}c. \end{cases}$
- $$\hat{\boldsymbol{C}} = \begin{array}{cc} \begin{array}{ccc} a & b & c \end{array} & \begin{array}{ccc} {\sim}a & {\sim}b & {\sim}c \end{array} \\ \hline \underbrace{\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}}_{\hat{\boldsymbol{C}}^{pos}} & \underbrace{\begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}}_{\hat{\boldsymbol{C}}^{neg}} \end{array}$$

  :- a & ~b.
  :- b & ~c.

- $L^{c} = (\mathbf{1} \bullet (\mathbf{1} - \min(\boldsymbol{N}_{\hat{c}}, 1)))$ = |violated constraints|

  where $\boldsymbol{N}_{\hat{c}} = \hat{\boldsymbol{C}}^{pos} \cdot (\mathbf{1} - \boldsymbol{s}_{l}) + \hat{\boldsymbol{C}}^{neg} \cdot \boldsymbol{s}_{l}$ = |false literals in constraint evaluation|
- $L^{c} = 0$ iff every conjunct in the body is evaluated false (constraint is satisfied)
- $J_{L}^{c} = (\hat{\boldsymbol{C}}^{pos} - \hat{\boldsymbol{C}}^{neg})^{T} [\boldsymbol{N}_{\hat{c}} < \mathbf{1}]$

Computing supported models satisfying constraints

- Given a program **$\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$** and constraints $\hat{\boldsymbol{C}}$, we compute supported models by minimizing $L^{\mathrm{Su+c}} = L^{\mathrm{Su}} + \ell_3 \cdot L^{\mathrm{c}}$ to zero
  - $L^{\mathrm{Su}} = (1/2) \cdot \| \min_1(\boldsymbol{DM}) - \boldsymbol{s}_I \|^2 + (1/2) \cdot \ell_2 \cdot \| \mathbf{s}_\mathbf{I} \odot (\mathbf{1} - \boldsymbol{s}_I) \|^2 \quad (\ell_2 > 0)$
  - $L^{\mathrm{c}} = (\mathbf{1} \bullet (\mathbf{1} - \min(\boldsymbol{N}_{\hat{c}}, 1)))$
- We use Newton's method with Jacobian $\boldsymbol{J}_L^{\mathrm{Su+c}} = \boldsymbol{J}_L^{\mathrm{Su}} + \ell_3 \cdot \boldsymbol{J}_L^{\mathrm{c}}$

  $\boldsymbol{J}_L^{\mathrm{Su}} = (\mathbf{C}^{\mathrm{pos}} - \mathbf{C}^{\mathrm{neg}})^{\mathrm{T}} [\boldsymbol{N} \leq \mathbf{1}] \odot (\boldsymbol{D}^{\mathrm{T}} ([(\boldsymbol{D} \cdot \boldsymbol{M}) \leq \mathbf{1}] \odot \boldsymbol{E}))) - \boldsymbol{E} + \ell_2 \, (\mathbf{s}_\mathbf{I} \odot (\mathbf{1} - \boldsymbol{s}_I) \odot (\mathbf{1} - 2 \cdot \boldsymbol{s}_I))$

  $\boldsymbol{J}_{\mathrm{L}}^{\mathrm{c}} = (\hat{\boldsymbol{C}}^{pos} - \hat{\boldsymbol{C}}^{neg})^{\mathrm{T}} [\boldsymbol{N}_{\hat{c}} < \mathbf{1}]$

- For stable models, we compute supported models from random initialization until a stable models is found

Minimization algorithm

- Input: metricized program $\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$, constraint matrix $\hat{\boldsymbol{C}}$

  Output: binary vector $\boldsymbol{s}_l^*$ such that $L^{\mathrm{Su+c}}(\boldsymbol{s}_l^*) = 0$

```
initialize $\boldsymbol{s}_l$ randomly
for i = 1 to max_try
        for j = 1 to max_itr
            threshold optimally $\boldsymbol{s}_l$ to binary $\boldsymbol{s}_l^*$ and compute error = $J^{\mathrm{Su+c}}(\boldsymbol{s}_l^*)$;
            if (error = 0)  break ;
            compute $L^{\mathrm{Su+c}} = L^{\mathrm{Su}} + \ell_3 \cdot L^{\mathrm{c}}$ and $\boldsymbol{J}_L^{\mathrm{Su+c}} = \boldsymbol{J}_L^{\mathrm{Su}} + \ell_3 \cdot \boldsymbol{J}_L^{\mathrm{c}}$ ;
            $\boldsymbol{s}_l = \boldsymbol{s}_l - \gamma\,(L^{\mathrm{Su+c}}/\| \boldsymbol{J}_L^{\mathrm{Su+c}} \|_2^2)\, \boldsymbol{J}_L^{\mathrm{Su+c}}$ ;
        endfor
        if (error = 0)  break ;
        perturbate $\boldsymbol{s}_l$ ;
    endfor
```

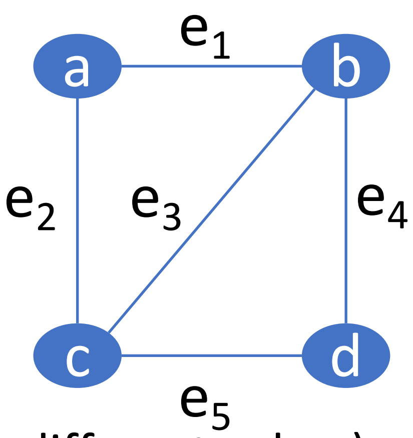
3-coloring of G0

- Consider a 3-coloring problem of graph G0:

  Nodes = {a, b, c, d}, color = one-of(red, blue, green)
- Program *P*: one-of(a1, a2, a3) .. one-of(d1, d2, d3)

```
a1 :- ~a2, ~a3.     a2 :- ~a1, ~a3.     a3 :- ~a1, ~a2.
b1 :- ~b2, ~b3.     b2 :- ~b1, ~b3.     b3 :- ~b1, ~b2.
c1 :-  ~c2, ~c3.    c2 :- ~c1, ~c3.     c3 :- ~c1, ~c2.
d1 :- ~d2, ~d3.     d2 :- ~d1, ~d3.     d3 :- ~d1, ~d2.
```

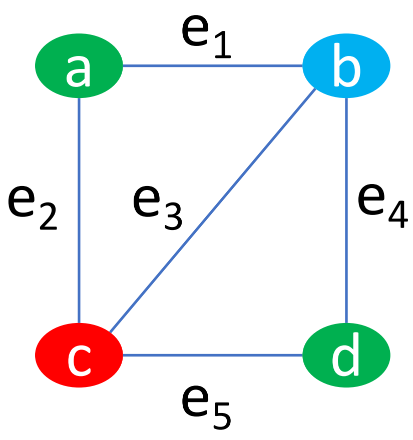
G0



- Constraints *C* (two nodes connected by an edge must have different colors)

:- a1, b1. :- a2, b2. :- a3, b3. (by e_1)
:- a1, c1. :- a2, c2. :- a3, c3. (by e_2)
:- b1, c1. :- b2, c2. :- b3, c3. (by e_3)
:- b1, d1. :- b2, d2. :- b3, d3. (by e_4)
:- d1, c1. :- d2, c2. :- d3, c3. (by e_5)

$$\mathbf{u} = [a1\ a2\ a3\ \ b1\ b2\ b3\ \ c1\ c2\ c3\ \ d1\ d2\ d3]^T$$
$$= [\,0\ \ 0\ \ 1\ \ \ 0\ \ 1\ \ 0\ \ \ 1\ \ 0\ \ 0\ \ \ 0\ \ 0\ \ 1\,]^T$$



Matrix encoding

- Program $\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$

$$\boldsymbol{D} = \begin{array}{|cccc|l} \mathbf{I} & & & & :a \\ & \mathbf{I} & & & :b \\ & & \mathbf{I} & & :c \\ & & & \mathbf{I} & :d \end{array}$$

every atom has a single rule

$$\boldsymbol{C} = \begin{array}{r|cccccccc|} & \underset{a_1\,a_2\,a_3}{a} & b & c & d & {\sim}a & {\sim}b & {\sim}c & {\sim}d \\ \hline a: & & & & & \boldsymbol{H} & & & \\ b: & & 0 & & & & \boldsymbol{H} & & \\ c: & & & & & & & \boldsymbol{H} & \\ d: & & & & & & & & \boldsymbol{H} \end{array}$$

$$\underbrace{\qquad\qquad}_{\boldsymbol{C}^{pos}}\quad\underbrace{\qquad\qquad}_{\boldsymbol{C}^{neg}}$$

$$\boldsymbol{H} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

- Constraint $\hat{\boldsymbol{C}}$

$$\hat{\boldsymbol{C}} = \left(\underbrace{\;\;0\;\;}_{\hat{\boldsymbol{C}}^{pos}}\quad \underbrace{\begin{matrix} \boldsymbol{I} & \boldsymbol{I} & & \\ \boldsymbol{I} & & \boldsymbol{I} & \\ & \boldsymbol{I} & \boldsymbol{I} & \\ & \boldsymbol{I} & & \boldsymbol{I} \\ & & \boldsymbol{I} & \boldsymbol{I} \end{matrix}}_{\hat{\boldsymbol{C}}^{neg}}\right)$$

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Computing performance

average over 10 runs	time(s)	#solution /10 trials
Su	6.7 (0.7)	5.2 (0.9)
GL reduct	8.1 (0.7)	4.7 (0.7)

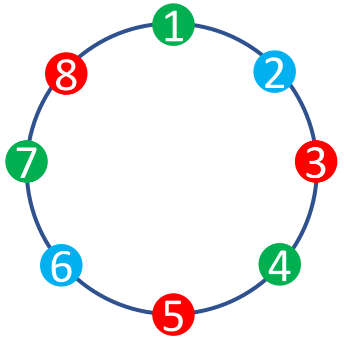
1 trial: max_try = 20, max_itr = 50, $\ell_2 = \ell_3$= 0.1

Programs are run on a PC with Intel(R) Core(TM) i7-10700@2.90GHz CPU with 26GB memory

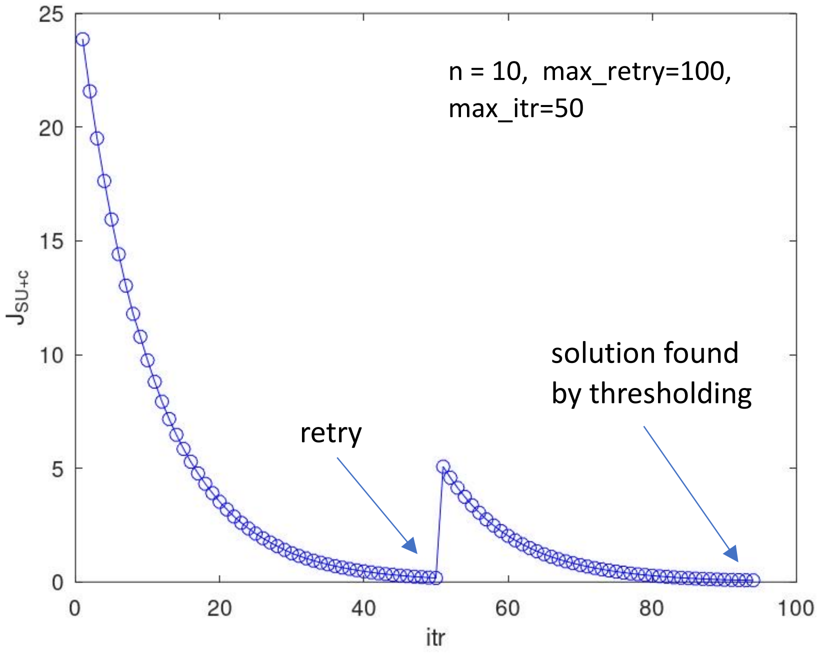
3-coloring of cycle graph

- Consider a 3-coloring problem of cycle graph:

  Nodes = {1,..,n}, color = one-of(red, blue, green)
- 3n atoms: ***D***(3n x 3n), ***C***(3n x 6n), $\hat{\boldsymbol{C}}$(3n x 6n)
- #3-coloring_of_cycle(n) = $2^n + 2\cdot(-1)^n$



Convergence curve

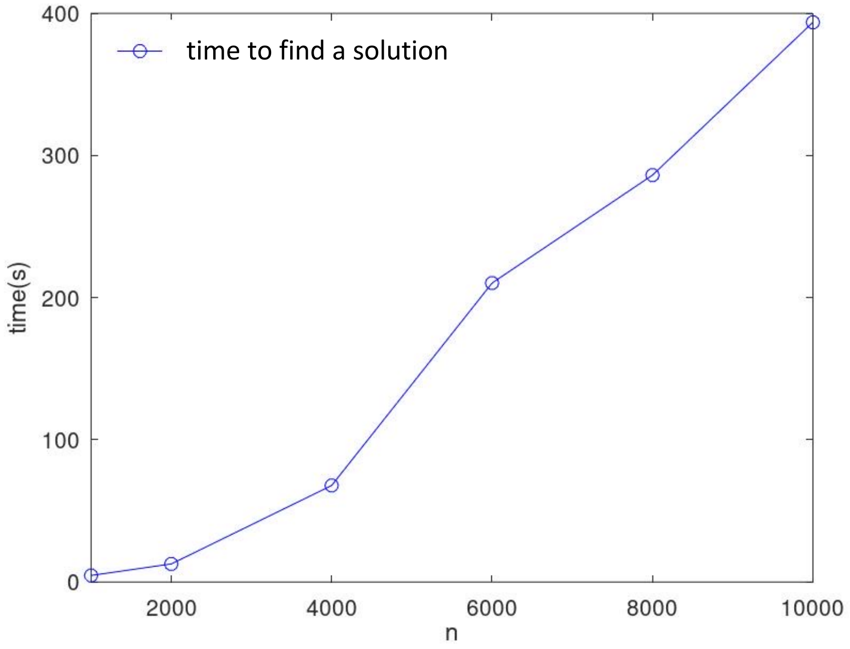


n = 10, max_retry=100,
max_itr=50

retry

solution found
by thresholding

Scalability



Hamiltonian cycle problem

- Hamiltonian cycle (HC): a round trip visiting every city once
- Two types of encoding possible
  - non-tight normal logic program + constraint [Niemela 1999], [Lin+ 2003]
  - tight normal logic program + constraint (none?)

transformation to inherently tight program

- We modify the SAT encoding of HC by [Zhou 2020]

U(j,q) = 1: node j is in HC and visited at time q (1≤i,q≤K)

H(i,j) = 1: edge i→j is in HC

(1) one-of(H(i,j_1)..H(i,j_k)) : outgoing edges are exclusive (1≤i≤K)

(2) one-of(H(i_1,j)..H(i_k,j)) : incoming edges are exclusive (1≤i≤K)

(3) H(1,j) ⇒ U(j,2) : redundant and removed

(4) H(i,1) ⇒ U(i,K) : if i→1 exists, i is visited at time K (2≤i≤ K)

(5) H(i,j) & U(i,q-1) ⇒ U(j,q) : if i→j exists and node i is visited at time q-1, node j is visited at time q (1≤i,j≤K, 2≤q≤ K)

(6) one-of(U(i,1)..U(i,K)) : node i is visited exactly once (1≤i≤K)

(7) U(1,1) : node 1 is visited at time 1 (starting node)

Hamiltonian cycle problem (cont'd)

- We solve the HC problem for G1
- We introduce 72 atoms for H(i,j) and U(j,q) and encode the HC problem as follows:

(1) one-of($H(i,j_1)..H(i,j_k)$)
(4) H(i,j) & U(i,q-1) ⇒ U(j,q)
(7) U(1,1)

tight program (D Q)
Q(197 x 144)
D(72 x 197)

(2) one-of($H(i_1,j)..H(i_k,j)$)
(5) H(i,1) ⇒ U(i,K)
(6) one-of(U(i,1)..U(i,K))

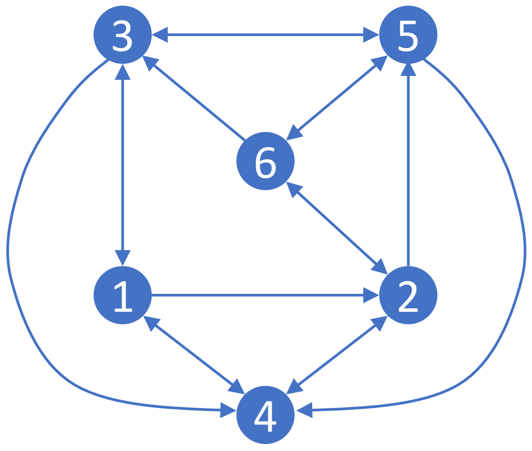
constraint ($\ell_3 \cdot J^C$)
Q_c(66 x 144)

H(1,2) :- ~H(1,3) & ~H(1,4).
U(2,4) :- (U(1,3) & H(1,2)) V ⋯ V (U(6,3) & H(6,2)).

Average time to find a HC over 10 trials (octave on PC: 2.90 GHz 32GB)

ℓ_3	0.02	0.05	0.1	0.15	0.2
time(s)	5.2(6.6)	4.5(4.4)	5.1(4.3)	8.2(8.6)	6.2(7.0)

G1



from A User's Guide to gringo, clasp, clingo, and iclingo ver.3, 2010

There are five HCs:
1 -> 2 -> 6 -> 3 -> 5 -> 4 -> 1
1 -> 2 -> 6 -> 5 -> 3 -> 4 -> 1
1 -> 3 -> 5 -> 6 -> 2 -> 4 -> 1
1 -> 4 -> 2 -> 5 -> 6 -> 3 -> 1
1 -> 4 -> 2 -> 6 -> 5 -> 3 -> 1

Loop formulas *LF*

- We can compute solely stable models s_I of a program *P* by matricizing the Lin-Zhao theorem [Lin and Zhao 2004]:

  s_I is a stable model of *P* iff $s_I \models \mathrm{comp}(P)$ and $s_I \models LF$

- Loop formulas *LF*:
  - Loop S = $\{p_1,\dots,p_k\}$: atoms such that there is a path from p_i to p_j and vice versa in the positive dependency graph of *P*; p has a self-loop when S = {p}
  - Body(p) = $G_1 \vee \cdots \vee G_j$ where rule (p :- G_i) ∈ P and

    G_i^+ (positive literals of G_i, possibly empty) $\cap\, S = \emptyset$ ($1 \le i \le j$)

    when no such G_i exists, Body(p) is false
  - $LF_{OR}(S)$: OR-type loop formula associated with S

    $= (p_1 \vee \cdots \vee p_k) \rightarrow (\mathrm{Body}(p_1) \vee \cdots \vee \mathrm{Body}(p_k))$
  - *LF* : the set of all loop formulas for *P* = $\{ LF_{OR}(S) \mid S \text{ is a loop in } P \}$

- *LF* says every loop has an exit that calls atoms outside the loop

AND-type loop formula

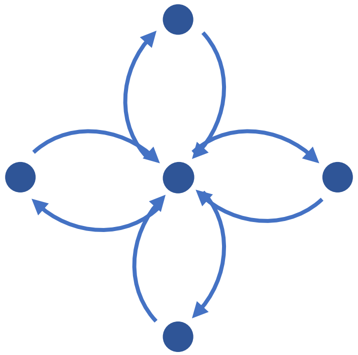
- OR-type Loop formulas in the Lin-Zhao theorem [Li and Zhao 2004] can be replaced by AND-type ones [Ferraris, Lee & Lifschitz 2006]:

$$LF_{AND}(L) = (p_1 \& \cdots \& \ p_k) \rightarrow (Body(p_1) \vee \cdots \vee Body(p_k))$$

$$= (\sim p_1 \vee \cdots \vee \sim p_k) \vee (Body(p_1) \vee \cdots \vee Body(p_k))$$

$$LF = \{ LF_{AND}(S) \mid S \text{ is a loop in } P \}$$

- To reduce the computational difficulty (complete digraph has 2^n-1 loops), we heuristically choose a subclass of loops



${}_4C_1 + \cdots + {}_4C_4 = 2^4 - 1$ loops

${}_4C_1 = 4$ minimal loops

➔ exponential reduction

Matricizing $\boldsymbol{s}_l \models LF$ by $L^{\mathrm{LF}} = 0$

- Let $S = \{p_1,\dots,p_k\}$ be a v-th loop in the positive dependency graph of $\boldsymbol{P} = (\boldsymbol{C}, \boldsymbol{D})$,
  $LF_{\mathrm{AND}}(S) = (p_1 \& \cdots \& \, p_k) \rightarrow (\mathrm{Body}(p_1) \vee \cdots \vee \mathrm{Body}(p_k))$
- Introduce a non-negative function L^{LF} of $\boldsymbol{s}_l$ by
  - $L^{LF} = \sum_{v=1}^{w} (1 - \min(\boldsymbol{A}(v), 1))$
  - $\boldsymbol{A}(v) = \boldsymbol{S}(v, :) \cdot (\boldsymbol{1} - \boldsymbol{s}_l) + \boldsymbol{S}(v, :) \cdot \boldsymbol{E}(v) \cdot \boldsymbol{M} \quad (1 \le v \le w) : s_l \models LF(\boldsymbol{S}(v, :))$

$\boldsymbol{s}_l \models \mathrm{Body}(p_1) \vee \cdots \vee \mathrm{Body}(pk)$

$\boldsymbol{s}_l \models \sim(p_1 \& \cdots \& \, pk)$

  - $\boldsymbol{S}(v, :)$: v-th loop $\{p_1,\dots,p_k\} = [0 \, .. \overset{p_1}{1} .. \overset{p_k}{1} .. \, 0\,]$
  - $\boldsymbol{E}(v)(p, :) = \{G_1,\dots,G_j\}$, where $\mathrm{Body}(p) = G_1 \vee \cdots \vee G_j = [0 \, .. \overset{G_1}{1} .. \overset{G_j}{1} .. \, 0]$
  - $\boldsymbol{M} = \boldsymbol{1} - \min_1(\boldsymbol{C}^{pos}(\boldsymbol{1} - \boldsymbol{s}_l) + \boldsymbol{C}^{neg}\boldsymbol{s}_l)$, where $\boldsymbol{C} = [\boldsymbol{C}^{pos} \; \boldsymbol{C}^{neg}]$ ($\boldsymbol{M}$ is the truth values of rule bodies $\boldsymbol{C}$)
- We can prove for a binary $\boldsymbol{s}_l$

  $L^{\mathrm{LF}} = 0$ iff $\boldsymbol{A}(v) \ge 1$ for $\forall v$ iff $\boldsymbol{s}_l \models LF_{\mathrm{AND}}(S)$ for $\forall$ loop S iff $\boldsymbol{s}_l \models LF$
- $\boldsymbol{J}_L^{\mathrm{LF}} = \partial L^{\mathrm{LF}} / \partial \boldsymbol{s}_l$

  $= \sum_{v=1}^{w} [\boldsymbol{A}(v) \le \boldsymbol{1}] \cdot ([\boldsymbol{N}(v) \le \boldsymbol{1}] \, (\boldsymbol{S}(v, :)^{\mathrm{T}}) + ((\boldsymbol{S}(v, :) \, \boldsymbol{E}(v))^{\mathrm{T}} \odot [\boldsymbol{N} \le \boldsymbol{1}])^{\mathrm{T}} (\boldsymbol{C}^{neg} - \boldsymbol{C}^{pos})))^{\mathrm{T}} \,)$

  where $\boldsymbol{N}(v) = \boldsymbol{S}(v, :) \cdot \boldsymbol{s}_l$ and $\boldsymbol{N} = \boldsymbol{C}^{pos} \, (\boldsymbol{1} - s_l) + \boldsymbol{C}^{neg} \, \boldsymbol{s}_l$

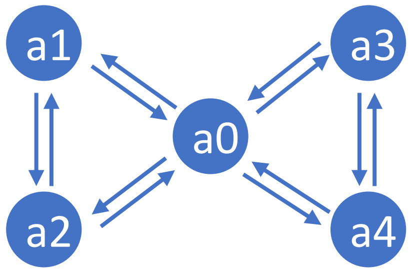
Three LF heuristics

- There are exponentially many loop formulas *LF*
  - elementary loops [Gebser+ 05], proper loops [Ji+ 14] introduced
- To guide minimization, we use a subset of *LF* associated with
  - maximal loops: *LF_max* (= SCCs, self-loop must for singleton SCC {a})
  - minimal loops: *LF_min* (= cycles, elementary loops)
  - *LF_min* but with external supports for *LF_max*: *LF_min_max*
- comp(*P*) + *LF_max* or comp(*P*) + *LF_min* may exclude some supported models but never stable ones
- $\boldsymbol{u} \models$ *LF_min_max* implies $\boldsymbol{u} \models$ *LF*, so $\boldsymbol{u} \models$ comp(*P*) + *LF_min_max*

  is a sufficient condition for stable model $\boldsymbol{u}$

Loopy program *P*1

- See differences between three heuristics



Program *P*1:

a0 :- a1 & a2 & a3 & a4.
a1 :- a0 V a2.
a2 :- a0 V a1.
a3 :- a0 V a4.
a4 :- a0 V a3.

supported models = { {}, {a1,a2}, {a3,a4}, {a0,a1,a2,a3,a4} } $(2^{4/2}-1)+1$ models
stable models = { Ø }

- Loop formulas exclude some supported models

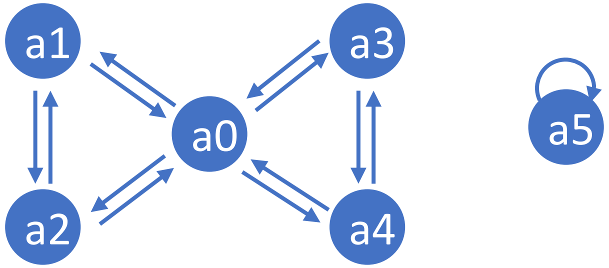
LF_max = { a0 & a1 & a2 & a3 & a4→ ⊥ } ⇏ {a0, a1, a2, a3, a4}

LF_min = { a0 & a1→a2, a0 & a2→a1, a0 & a3→a4,
a0 & a4→a3, a1 & a2→a0, a3 & a4→a0 } ⇏ {a1, a2}, {a3, a4}

LF_min_max = { a0 & a1→⊥, a0 & a2→⊥, a0 & a3→⊥,
a0 & a4→⊥, a1 & a2→⊥, a3 & a4→⊥ } ⇏ {a1,a2}, {a3,a4}, {a0,a1,a2,a3,a4}

Loopy program *P2*

- See differences between three heuristics



Program *P2*:

a0 :- a1 & a2 & a3 & a4.
a1 :- a0 V a2.
a2 :- a0 V a1.
a3 :- a0 V a4.
a4 :- a0 V a3.
a5 :- a5.
a0 :- ~a5.

$2^{4/2}+1$ = 5 supported models, 1 stable model = {a0,a1,a2,a3,a4}

- Loop formulas exclude some supported models

LF_max = { a0 & a1 & a2 & a3 & a4 → ⊥, a5 → ⊥ } ⇏ all except {a0..a4}

LF_min = { a0 & a1 → a2, a0 & a2 → a1, a0 & a3 → a4,
a0 & a4 → a3, a1 & a2 → a0, a3 & a4 →a0, a5 → ⊥ } ⇏ all except {a0..a4}

LF_min_max = { a0 & a1 → ⊥, a0 & a2 → ⊥, a0 & a3 → ⊥,
a0 & a4 → ⊥, a1 & a2 → ⊥, a3 & a4 → ⊥, a5 → ⊥ } ⇏ all supported models

Loopy program *P*2 (cont'd)

- The effect of loop formula heuristics

Average time and trials to find a stable model over 10 runs

LF	time(s)	trials	#supported model	#stable model
no *LF*	0.16	3.1	3.3	0.8
LF_max	4.1	1	1	1
LF_min	2.2	1	1	1
LF_min_max	tiemout	5	0	0

← stable model excluded

- max_retry 20, max_itr = 50
- 1 trial = (max_retry × max_itr) computation
- 1 run = 5 trials
- time = time for 10 runs
- timeout = 240s

Loopy program *P*2 (cont'd 2)

- Another solution constraint:

  when a model {a,b} is found, add (:- a&b.) to constrain for next solution

Average time and trials to find a stable model

another solution constraint	time(s)	trials
not used	11.46	10,000
used	0.09	3.5

no stable model found due to learning bias

- no_LF used (purely supported model computation)
- max_retry = 20, max_itr = 50
- 1 trial = (max_retry × max_itr) updates
- 1 run = 10,000 trials
- time = average of 10 runs

- Useful and necessary for multiple solutions

Loopy program *P2_n*

- Generalizing *P2* to *P2_n* (n: even)

```
a(0) :- a(1) & ⋯ & a(n).
  ⋮
a(2i-1) :- a(0) ∨ a(2i).   for  i=1..n/2
a(2i) :- a(0) ∨ a(2i-1).   for  i=1..n/2
  ⋮
a(n+1) :- a(n+1).
a(0) :- ~a(n+1).
```

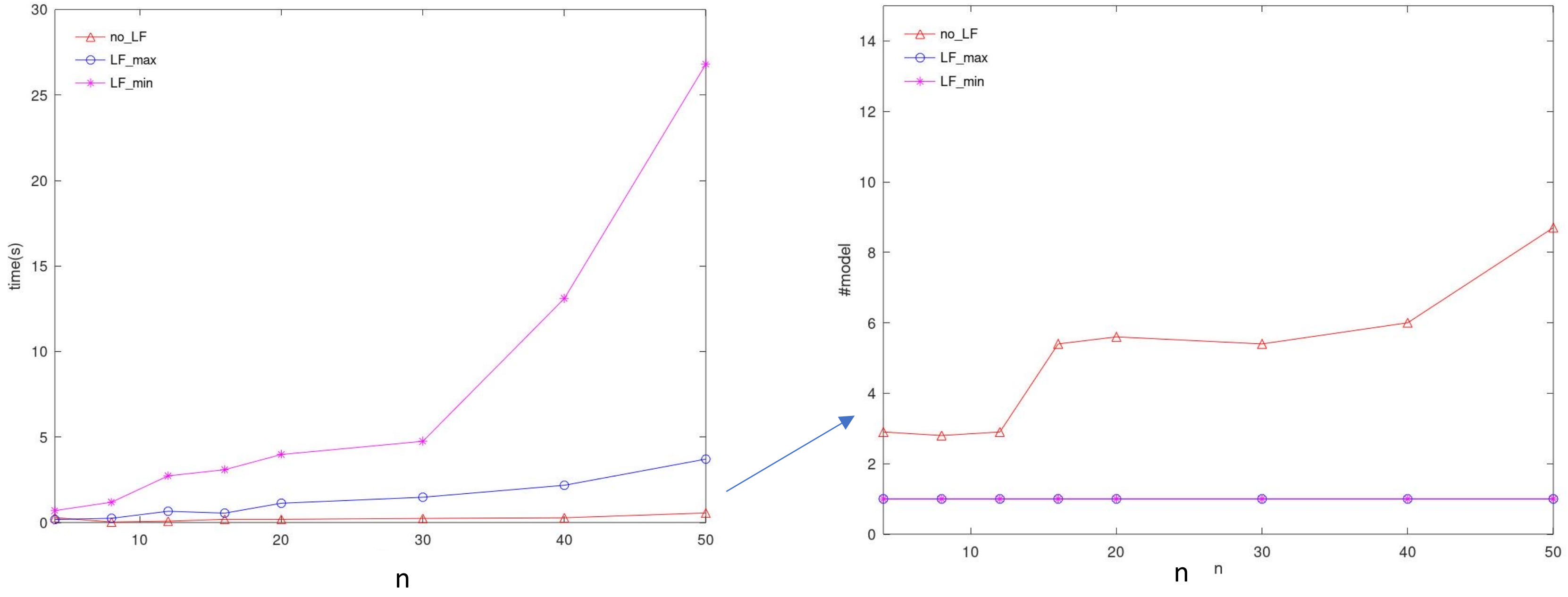
- Loop formulas
  - $2^{n/2}+1$ supported models, one stable model M_0 = {a(0),…,a(n)}
  - *LF_max* = { a(0) & a(1) &⋯& a(n) → ~a(n+1), a(n+1) → ⊥ } allows M_0
  - *LF_min* = { a(1) & a(2) → a(0), a(3) & a(4) → a(0),…, a(n+1) → ⊥ } allows M_0

Loopy program *P*2_n (cont'd)

- Scalability wrt n: time to find one stable model (left) and the total number of supported models found (right)

scalability

|computed supported models|



max_try = 10, max_itr = 100, max_fp = $2^{n/2}$+1

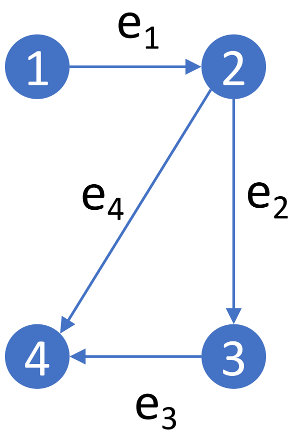
- no_*LF* is much faster than *LF_max*, *LF_min* (left)
- no_*LF* computes non-stable models, but *LF*_{*max*, *min*} don't (right)

More natural program: transitive closure

- Compute the lfp of comp(P_{tr})

$$P_{\text{tr}} = \begin{cases} \text{tr(X, Z) :- tr(X, Y) \& tr(Y, Z).} \\ \text{tr(1, 2). tr(2 ,3). tr(2, 4). tr(3, 4).} \end{cases}$$

Domain = {1,2,3,4}



  - grounding P_{tr} generates 64 rules in 16 atoms
  - matrix encoding gives (***C***(16x64) ***D***(64x128))
- P_{tr} has > 34 supported models
  - pruning by *LF_max* leaves just one stable model

Time to find a stable model

	time(s)
no_*LF*	2.8
LF_max	63.4

max_retry = 10, max_itr = 100
LF_min takes too long

Adjacency matrix
of transitive closure

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

➡ *LF_max* works but takes long time

Precomputation (1)

- For a normal logic program P = { a :- B & N. }, put P^+ = { a :- B. }
- Let P^u be the GL reduct of P by a stable model $\boldsymbol{s}_I$
  - $P^u \subseteq P^+$, so lfp(P^u) $\subseteq$ lfp(P^+), hence every atom outside P^+ is false in *any* stable model
- Precomputation: partial evaluation by false atoms
  - compute $F_P = \mathrm{HB} \backslash \mathrm{lfp}(P^+)$ in O(|P|), where |P| is the total number of atom occurrences in P [Dowling+ 84]
  - G' = conjunction G with $\{ \neg a \in G \mid a \in F_P \}$ removed
  - $P' = \{ (a \leftarrow G') \mid (a \leftarrow G) \in P,\ a \notin F_P,\ G^+ \cap F_P = \emptyset \}$
  - $C' = \{ (\leftarrow G') \mid (\leftarrow G) \in C,\ G^+ \cap F_P = \emptyset \}$
- $\boldsymbol{s}_I$ is a stable model of P satisfying constraints C

  iff $\boldsymbol{s}_I'$ a stable model of P' satisfying constraints C', where $\boldsymbol{s}_I = \boldsymbol{s}_I' + \{ a \in F_P$ is false in $\boldsymbol{s}_I \}$

Precomputation (2)

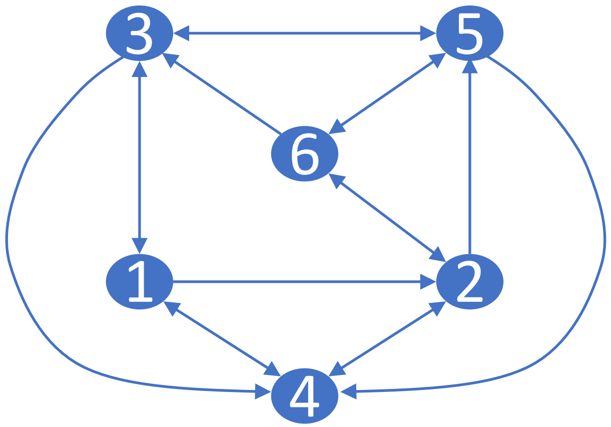
- The effect of precomputation on the HC problem example
  - |HB| = 72, $|F_P|$ = 32, so 32 atoms are detected as false, 40 atoms need to be decided

Time to find one stable model

	No precomp.	Precomp.
time(s)	2.08(2.01)	0.66(0.52)
matrix size	***D***: 72 x 197 ***C***: 194 x 144 $\hat{\boldsymbol{C}}$: 67 x 144	***D'***: 40 x 90 ***C'***: 90 x 80 $\hat{\boldsymbol{C}}$***'***: 52 x 80

max_try = 20, max_itr = 200, $l_2 = l_3 = 0.1$
average of 10 trials

G1



from “A User's Guide to gringo”, clasp, clingo, and iclingo ver.3, 2010

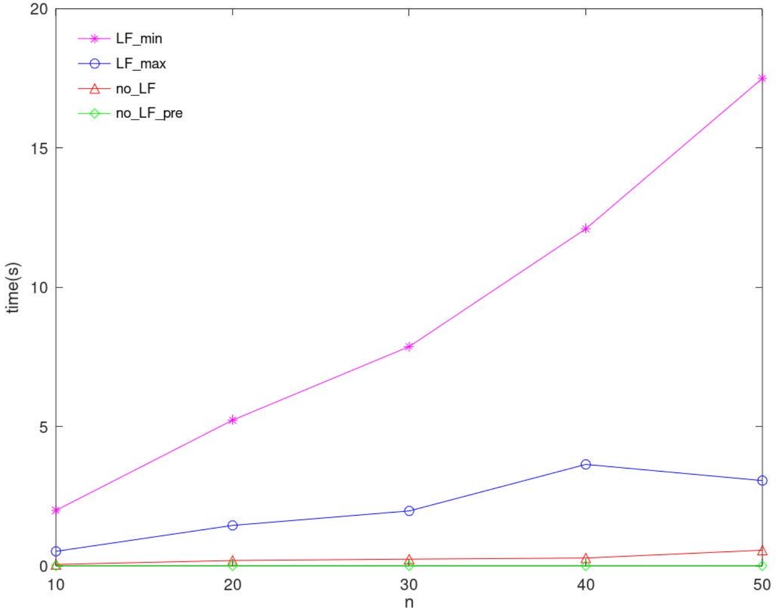
Precomputation (3)

- *P*2_n: n+2 atoms

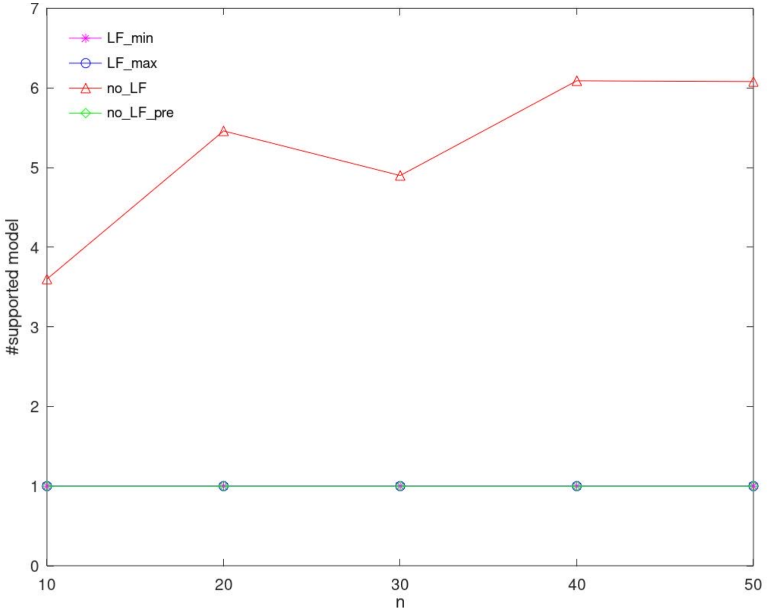
$$\left\{\begin{array}{l} a_0 \text{ :- } a_1 \,\&\, \cdots \,\&\, a_n. \\ a_1 \text{ :- } a_0 \vee a_2.\ \ a_2 \text{ :- } a_0 \vee a_1.\ \ \ldots\ \ a_{n-1} \text{ :- } a_0 \vee a_n.\ \ a_n \text{ :- } a_0 \vee a_{n-1}. \\ a_{n+1} \text{ :- } a_{n+1}. \\ a_0 \text{ :- } {\sim}a_{n+1}. \end{array}\right.$$

  - $2^{n/2}+1$ supported models, 1 stable model $\{a_0,a_1..a_n\}$ (only a_{n+1} is false)

Time to find a stable model



#computed supported models in 10 trials



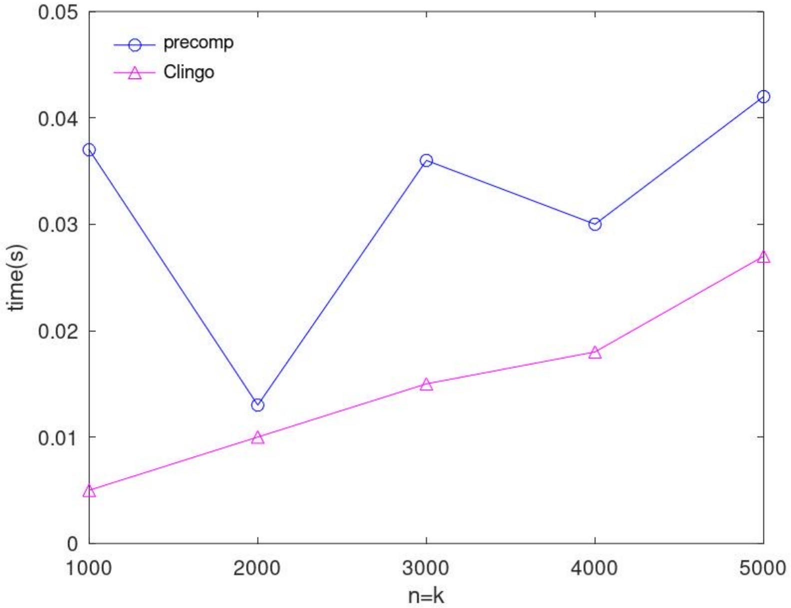
Precomputation (4)

- *P*2_n+k: n+k+2 atoms

$$\begin{cases} a_0 \text{ :- } a_1 \,\&\, \cdots \,\&\, a_n. \quad a_0 \text{ :- } {\sim}a_{n+1} \,\&\, \cdots \,\&\, {\sim}a_{n+k}. \\ a_1 \text{ :- } a_0 \vee a_2. \quad a_2 \text{ :- } a_0 \vee a_1. \ \ldots \quad a_{n-1} \text{ :- } a_0 \vee a_n. \quad a_n\text{:- } a_0 \vee a_{n-1}. \\ a_{n+1} \text{ :- } a_{n+1}. \ \ldots \ a_{n+k} \text{ :- } a_{n+k}. \end{cases}$$

  - $(2^{n/2}-1)(2^k-1)+1$ supported models, 1 stable model $\{a_0,a_1..a_n\}$ $({\sim}a_{n+1}..{\sim}a_{n+k})$

Time to find one stable model



max_try=10, max_itr=100, $l_2 = l_3 = 0.1$, average of 10 trials

$|F_P|/|HB| = 5000/10001$ when n = k = 5000
pre-computation time = 0.000005s

matrix size	***C***: 10001 x 15002 ***D***: 15002 x 20002	***C'***: 5001 x 10002 ***D'***: 10002 x 10002

In a very special case, no parameter update required and our approach comes close to clingo (even by octave implementation)

Summary

- Supported models for a propositional normal logic program *P* with constraints are computed in vector spaces for the 3-color problem and the Hamiltonian cycle problem
- Stable models of *P* are computed based on the Lin-Zhao theorem by computing supported models of *P* that satisfy AND-type loop formulas
- We proposed three heuristics for loop formulas to avoid computing non-stable models:
  - *LF_max* by maximal loops (SCCs)
  - *LF_min* by minimal loops (cycles)
  - *LF_min_max* by merging *LF_max* and *LF_min*
- We also proposed precomputation to reduce program size
- We empirically confirmed the effect of these by simple experiments
- This is an initial study of differentiable ASP using matrix encodings
- More elaboration is expected

Outline

1. Introduction: Towards Trustworthy AI
2. Background: Algebraic Approaches to Logic Programming
3. Main: A Framework for Differentiable ASP
4. Supplementary: Tools for Differentiable ASP (unpublished)

Outline of Differentiable ASP solver

- Differentiable solver for stable model semantics
- Incomplete, approximate solver

1. Parse the normal logic program ***P***
2. Append ”loop formula constraints” ***LF*** to ***P***
3. Embed ***P*** + ***LF*** into matrix
4. Using a differentiable loss function,
   update the interpretation vector with gradient information

Building blocks: Matrices and Vectors (1/2)

Program

p :- q.
p :- not r.
q :- p.
r :- r.

C: Program Matrix

	p	q	r	$\bar{p}$	$\bar{q}$	$\bar{r}$
p	0	**1**	0	0	0	0
p	0	0	0	0	0	**1**
q	**1**	0	0	0	0	0
r	0	0	**1**	0	0	0

D: Head Matrix

	p	q	r
p	**1**	0	0
p	**1**	0	0
q	0	**1**	0
r	0	0	**1**

🟩 C^P: positive part

🟥 C^N: negative part

f^T: fact vector; 1 if P has facts

p	q	r
0	0	0

x^T: Interpretation vector

a ∈ ***I*** if 1

p	q	r
1	**1**	0

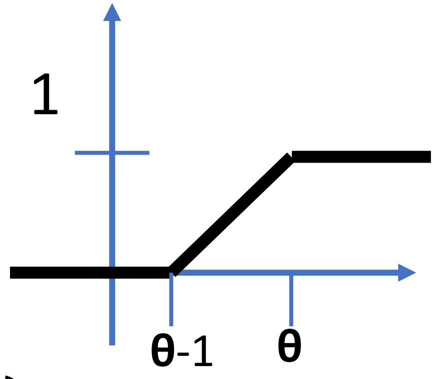
$[x; 1\text{-}x]^T$: Companion vector
(for multiplying with ***Q***)

p	q	r	$\bar{p}$	$\bar{q}$	$\bar{r}$
1	**1**	0	0	0	1

Building blocks: Differentiable Thresholding (2/2)

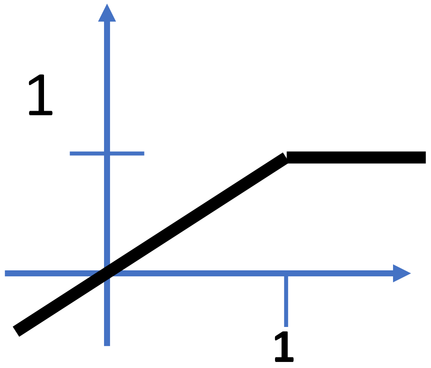
- Parameterized thresholding
  - $\boldsymbol{\theta}$: (n, 1) – vector, n=number of rules
  - θ_i : number of literals in the body
  - To check if the body of a rule is true
  - $\boldsymbol{x}_i \geq \theta_i$: body evaluates to true (then head is true)

$$ReLU_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1 - ReLU(1 - (ReLU(\boldsymbol{x} - \boldsymbol{\theta})))$$



- min1 thresholding
  - To check 'there is a rule such that…'
  - Used with "Head Matrix" (same head rules)
  - $\min(x, 1)$

$$ReLU_1(\boldsymbol{x}) = ReLU(1 - \boldsymbol{x})$$



Model Loss Function (L(x) = 0 corresponds to stable models)

- Given interpretation vector $\boldsymbol{x}$ (n_atom, 1)

$$L(x) = \frac{1}{2}\left(\begin{array}{l} \lambda_1 \| ReLU_1(\boldsymbol{D}^T ReLU_\theta(\boldsymbol{Q}[\boldsymbol{x}; \boldsymbol{1} - \boldsymbol{x}] + \boldsymbol{f_T} - \boldsymbol{f_F}) - \boldsymbol{x}\|_2^2 + \\ \lambda_2 \|\boldsymbol{x} \odot (\boldsymbol{x} - 1)\|_2^2 + \\ \lambda_3 \| ReLU_\theta(\boldsymbol{C}[\boldsymbol{x}; \boldsymbol{1} - \boldsymbol{x}])\|_2^2 \end{array}\right)$$

1. Is the model supported? (Tp(***x***) = ***x***?)
2. Is ***x*** binary?
3. Does ***x*** satisfy all constraints?

- $L(\boldsymbol{x})$ = 0 iif $\boldsymbol{x}$ is a stable model
  1. $\boldsymbol{x}$ is a supported model / ***Tp***(***M***) = ***M***
  2. $\boldsymbol{x}$ is a 0-1 binary vector
  3. $\boldsymbol{x}$ satisfies none of the constraints

Q: Program Matrix
C: Constraint Matrix
D: Head Matrix
$\boldsymbol{f}_T$: Fact vector
$\boldsymbol{f}_F$: False vector
x: Interpretation vector
ReLU$_\theta$: Parameterized thresholding
ReLU$_1$: min1 thresholding

Loss function is similar to the one in Takemura+2022.
Gradient w.r.t ***x*** was derived by hand but omitted in this presentation for brevity.

“Special” ASP rules

- Commonly used in ASP
- Choice:

  -
    ```
    {a; b; c}.
    ```
  - Choose from all possible combinations of a,b,c: {a} {b} … {a,c} … {a,b,c}
- Cardinality constraints:

  -
    ```
    { assign(N,C) : color(C) } = 1 :- node(N).
    ```
  - Assign only 1 color to a node, e.g., graph coloring
- Sum statement:

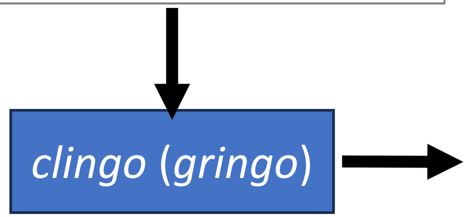
  -
    ```
    :- #sum { Price,Item : buy(Item), item(Item,Price) } > budget.
    ```
  - The sum of item price must not exceed the budget, e.g., knapsack
- Minimize statement:

  -
    ```
    # minimize { C/S,X : hotel(X), cost(X,C), star(X,S) }.
    ```
  - Minimize the cost per star rating

Encoding special ASP rules in Program Matrix

```
node(1..2).
color(1..2).
{assign(N,C) : color(C)} = 1 :- node(N).
```

Input program



clingo (gringo)

```
node(1). node(2). color(1). color(2).
#delayed(3). #delayed(4).
#delayed(3) <=>
1<=#count{0,assign(1,1):assign(1,1);0,assign(1,2):assign(1,2)}<=1
{assign(1,1);assign(1,2)}:-#delayed(3).
#aux(9)  :- 1{assign(1,1)=1,assign(1,2)=1}.
#aux(10) :- 2{assign(1,1)=1,assign(1,2)=1}.
#aux(11) :- #aux(9),not #aux(10).
:-#delayed(3),not #aux(11).
#delayed(4) <=>
1<=#count{0,assign(2,1):assign(2,1);0,assign(2,2):assign(2,2)}<=1
{assign(2,1);assign(2,2)}:-#delayed(4).
#aux(14) :- 1{assign(2,1)=1,assign(2,2)=1}.
#aux(15) :- 2{assign(2,1)=1,assign(2,2)=1}.
#aux(16) :- #aux(14),not #aux(15).
:-#delayed(4),not #aux(16).
```

Grounded by *clingo* (*gringo*)

1. #delayed – special atom
2. Cardinality turns into weighted choice rules

! Cannot directly translate into Program Matrix

Lp2mat: a translation library

INPUT: *clingo*-compatible ASP program

OUTPUT: Normal rules WITHOUT extended statements (matrix friendly)

Supported statements: #sum, #minimize (#maximize), #count

Not supported: #project, #external, #assume, #heuristic, #theory

How Lp2mat works

- 1. Grounding with *gringo*
- 2. Rule re-writing and expansion
  Translate weighted cardinality rules into normal rules

Example: #sum statement

```
#const budget=20
:- #sum { Price, Item : buy(Item), item(Item, Price) } > budget.
```

“The sum of item prices must not exceed the budget”

```
#aux(7):-21{buy(apple)=10,buy(banana)=10,buy(chocolate)=20,buy(crisps)=25,buy(soda)=30}.
```

Clingo’s version (choice begins with 21-weight)

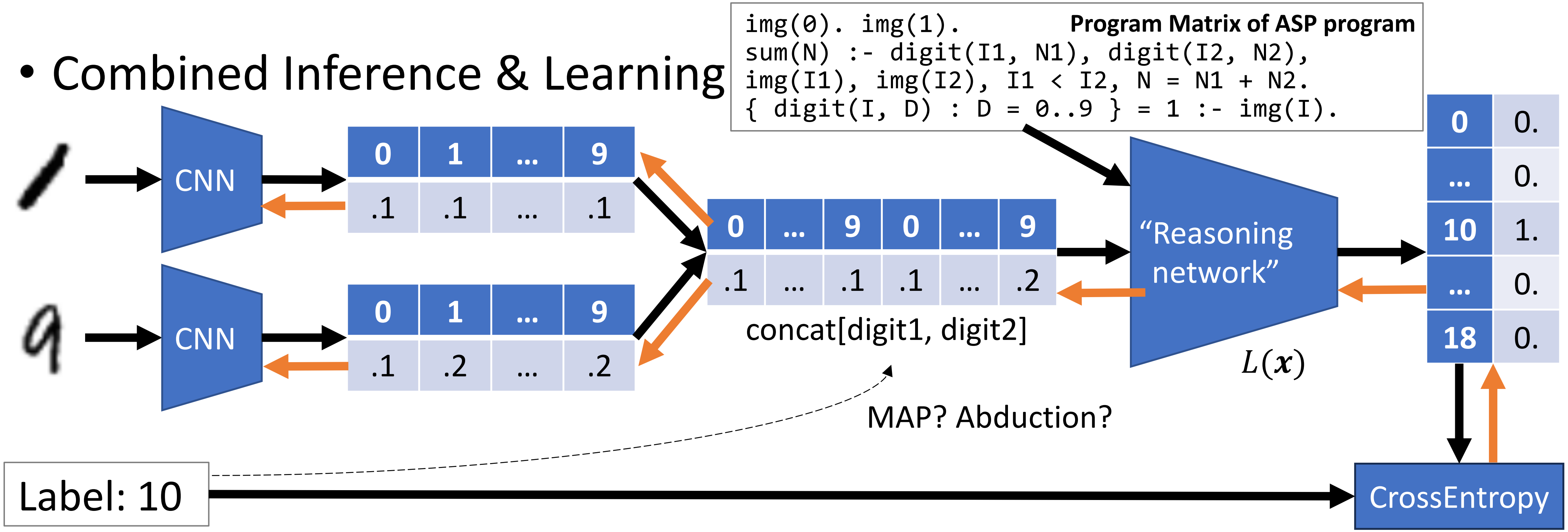
```
1 0 1 7 1 21 5 8 10 9 10 10 20 11 25 12 30  (ASP intermediate format)
```

:-#aux(7).   (#aux(7) cannot be true)

```
a_7 :- a_12.          %% buy(soda)
a_7 :- a_14_aux_1_21.
a_14_aux_1_21 :- a_11.  %% buy(crisps)
a_14_aux_1_21 :- a_15_aux_2_21.
a_15_aux_2_21 :- a_10, a_16_aux_3_1.  %% buy(chocolate)
a_16_aux_3_1 :- a_8.    %% buy(apple)
a_16_aux_3_1 :- a_9.    %% buy(banana)
:- a_7. %% NOT soda or crisps or (chocolate+apple) or chocolate(banana)
```

NeSy Applications

- Combined Inference & Learning



Program Matrix of ASP program

```
img(0). img(1).
sum(N) :- digit(I1, N1), digit(I2, N2),
img(I1), img(I2), I1 < I2, N = N1 + N2.
{ digit(I, D) : D = 0..9 } = 1 :- img(I).
```

forward

backward

MNIST addition

Inference: Given (1 , 9) ∈ Dataset, infer 10.

Learning: Given (1 , 9 , 10) ∈ Dataset, train a model that infers 10.

*learning to solve the addition task, not learning a logic program

Summary

- Differentiable loss function for computing stable models
  - Search is still a hard problem
- Lp2mat: Logic program to Program Matrix translator
- Neural-symbolic inference & learning:
  - Learning without direct supervision labels