NEURAL-SYMBOLIC RECURSIVE MACHINE FOR SYSTEMATIC GENERALIZATION

Qing Li[1], **Yixin Zhu**[3], **Yitao Liang**[1,3], **Ying Nian Wu**[2], **Song-Chun Zhu**[1,3], **Siyuan Huang**[1]
[1]National Key Laboratory of General Artificial Intelligence, BIGAI
[2]UCLA [3]Institute for Artificial Intelligence, Peking University

ABSTRACT

Current learning models often struggle with human-like systematic generalization; learning compositional rules from *limited* data and extrapolating them to unseen combinations. To address this, we introduce Neural-Symbolic Recursive Machine (NSR), a model whose core representation is a Grounded Symbol System (GSS), with its combinatorial syntax and semantics emerging entirely from the training data. The NSR adopts a modular approach, incorporating neural perception, syntactic parsing, and semantic reasoning, which are jointly learned through a deduction-abduction algorithm. We establish that NSR possesses sufficient expressiveness to handle a variety of sequence-to-sequence tasks and attains superior systematic generalization, thanks to the inductive biases of *equivariance* and *compositionality* inherent in each module. We assess NSR's performance against four rigorous benchmarks designed to test systematic generalization: SCAN for semantic parsing, PCFG for string manipulation, HINT for arithmetic reasoning, and a task involving compositional machine translation. Our results indicate that NSR outperforms existing neural or hybrid models in terms of generalization and transferability.

1 INTRODUCTION

A defining characteristic of human intelligence is *systematic compositionality*, the algebraic ability to create infinite interpretations from finite known components (Chomsky, 1957; Montague, 1970; Marcus, 2018)—termed "infinite use of finite means". This compositionality is crucial for generalizing from *limited* data to unseen combinations (Lake et al., 2017). Several datasets like SCAN (Lake and Baroni, 2018), PCFG (Hupkes et al., 2020), CFQ (Keysers et al., 2020), and HINT (Li et al., 2023) have been introduced to assess the systematic generalization of machine learning models. Conventional neural networks exhibit significant shortcomings on these datasets, prompting exploration into inductive biases to enhance systematic generalization. Csordás et al. (2021) and Ontanón et al. (2022) enhance Transformers' generalization by employing relative positional encoding and layer weight sharing. Chen et al. (2020) present a neural-symbolic stack machine, achieving near-perfect accuracy on SCAN-like datasets. Drozdov et al. (2023) prompts large language models to achieve compositional semantic parsing on CFQ. However, these approaches require domain-specific knowledge for designing intricate symbolic components and pose challenges in domain transfer.

To achieve human-like systematic generalization across diverse domains, we introduce Neural-Symbolic Recursive Machine (NSR), a principled framework that enables the *joint* learning of perception, syntax, and semantics. The core representation of NSR is a Grounded Symbol System (GSS), illustrated in Fig. 1. Importantly, NSR's GSS emerges entirely from training data, negating the need for domain-specific knowledge. NSR employs a modular design, incorporating neural perception, syntactic parsing, and semantic reasoning. Initially, a neural network serves as the perception module, grounding symbols in raw inputs. Subsequently, these symbols are structured into a syntax tree of the GSS by a transition-based neural dependency parser (Chen and Manning, 2014). Lastly, functional programs are utilized to interpret the semantic meaning of symbols (Ellis et al., 2021). Theoretically, we demonstrate that NSR is versatile enough to represent various sequence-to-sequence tasks. The embedded inductive biases of equivariance and compositionality allow NSR to decompose long inputs, process components progressively, and compose results, fostering the learning of meaningful symbols and compositional rules. These inductive biases are pivotal for NSR's exceptional systematic generalization.

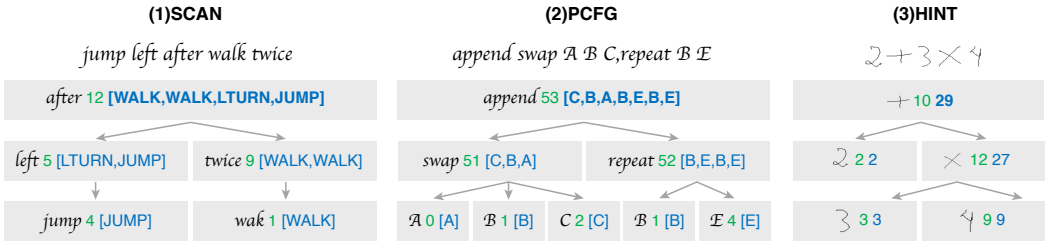


Figure 1: **Illustrative examples of GSSs demonstrating a human-like reasoning process.** (a) SCAN: each node represents a triplet of (*word*, symbol, action sequence). (b) PCFG: each node is composed of a triplet (*word*, symbol, letter list). (c) HINT: each node encapsulates a triplet (*image*, symbol, value). Symbols are denoted by their respective indices.

Optimizing NSR end-to-end is formidable due to the unavailability of annotations for the internal GSS and the model's non-differentiable nature. To address this, we introduce a probabilistic learning framework and formulate a novel *deduction-abduction* algorithm to coordinate the joint learning of various modules. During learning, the model initially employs greedy deduction across modules to propose an initial GSS, which may be incorrect. Subsequently, a search-based abduction is executed top-down to explore the initial GSS's neighborhood for potential solutions, refining the GSS until the accurate result is obtained. The refined GSS serves as *pseudo* supervision, facilitating the independent training of each NSR module.

We assess NSR against three benchmarks designed to test systematic generalization:

1. SCAN (Lake and Baroni, 2018), which maps natural language commands to action sequences;
2. PCFG (Hupkes et al., 2020), tasked with predicting the output sequences of string manipulation commands;
3. HINT (Li et al., 2023), focused on predicting the results of handwritten arithmetic expressions.

Each dataset includes multiple splits to evaluate various facets of systematic generalization. NSR sets a new standard on all benchmarks, achieving 100% generalization accuracy on SCAN and PCFG and improving the state-of-the-art accuracy on HINT by approximately 23%. Analyses indicate that NSR's modular design and inherent inductive biases enable stronger generalization compared to conventional neural networks and superior transferability compared to existing neural-symbolic models, requiring less domain-specific knowledge. To probe NSR's applicability to real-world scenarios, we test it on a compositional machine translation task by Lake and Baroni (2018). NSR attains 100% generalization accuracy, underscoring its viability in real-world applications with diverse and ambiguous rules.

2 Related Work

The exploration of systematic generalization in deep neural networks has garnered considerable attention in recent times within the machine learning community. Initiated by the introduction of the SCAN dataset (Lake and Baroni, 2018), numerous benchmarks spanning diverse domains have been established, encompassing semantic parsing (Keysers et al., 2020; Kim and Linzen, 2020), string manipulation (Hupkes et al., 2020), visual question answering (Bahdanau et al., 2019; Xie et al., 2021), grounded language understanding (Ruis et al., 2020), mathematical reasoning (Saxton et al., 2018; Li et al., 2023). These datasets act as platforms for assessing various facets of generalization, such as systematicity and productivity. Subsequent research on semantic parsing (Chen et al., 2020; Herzig and Berant, 2021; Drozdov et al., 2023) has explored diverse techniques, infusing various inductive biases into deep neural networks to optimize performance on these datasets. We organize the preceding approaches into three categories, based on their method of incorporating inductive bias.

1. **Architectural Prior:** This approach refines neural network architectures to enhance compositional generalization. Dessì and Baroni (2019) demonstrate the superiority of convolutional networks over RNNs in the "jump" split of SCAN. Russin et al. (2019) enhance standard RNNs by developing distinct modules for syntax and semantics. Gordon et al. (2019) introduce an equivariant seq2seq model, incorporating convolution operations into RNNs to attain local equivariance, provided a priori. Csordás et al. (2021) and Ontanón et al. (2022) note improvements in Transformers'

systematic generalization through relative position encoding and layer weight sharing. Gontier et al. (2022) propose to incorporate entity type abstractions into pretrained Transformers to enhance the logical reasoning capability.

2. **Data Augmentation:** This strategy devises schemes to create auxiliary training data to foster compositional generalization. Andreas (2020) augment data by interchanging fragments of training samples, and Akyürek et al. (2020) employ a generative model to recombine and resample training data. The meta sequence-to-sequence model (Lake, 2019) and the rule synthesizer (Nye et al., 2020) utilize samples from a meta-grammar resembling the SCAN grammar.
3. **Symbolic Scaffolding:** This method integrates symbolic elements into neural architectures to bolster compositional generalization. Liu et al. (2020) link a memory-augmented model to analytical expressions, emulating reasoning processes. Chen et al. (2020) embed a symbolic stack machine within a seq2seq framework, learning a neural controller for operation. Kim (2021) derive latent neural grammars for both encoder and decoder in a seq2seq framework. While these techniques achieve notable generalization by incorporating symbolic scaffolding, the need for domain-specific knowledge and intricate training procedures, like hierarchical reinforcement learning in Liu et al. (2020) and exhaustive search processes in Chen et al. (2020), limit their practical applicability.

Beyond technical implementations, existing works suggest two fundamental inductive biases crucial for compositional generalization: *equivariance* and *compositionality*, both pivotal for ensuring models' systematicity and productivity. The introduced NSR leverages a generalized Grounded Symbol System as symbolic scaffolding and instills equivariance and compositionality within its modules to attain robust compositional generalization. Unlike preceding neural-symbolic approaches, NSR necessitates minimal domain-specific knowledge and eschews the need for a specialized learning curriculum, resulting in improved transferability and streamlined optimization across diverse domains, as validated by our experiments; we refer the readers to Sec. 4 for details.

Our work is also related to neural-symbolic methods for logical reasoning. Rocktäschel and Riedel (2017) introduce Neural Theorem Provers (NTPs) for end-to-end differentiable proving of queries to knowledge bases by operating on dense vector representations of symbols. Minervini et al. (2020) propose Greedy NTPs to overcome the computational limitations of NTPs and extend them to real-world datasets. Mao et al. (2018) propose a Neuro-Symbolic Concept Learner, relying on a pre-established domain-specific language, to learn visual concepts from question-answering pairs.

3 Neural-Symbolic Recursive Machine

3.1 Representation: Grounded Symbol System (GSS)

The longstanding debate between connectionism and symbolism revolves around the optimal representation of the human mind (Fodor et al., 1988; Fodor and Lepore, 2002; Marcus, 2018). Connectionism emphasizes *distributed representations* (Hinton, 1984), positing that concepts are represented by activity patterns across numerous neurons. Conversely, symbolism advocates for a *physical symbol system* (Newell, 1980), where each symbol represents an atomic concept, and complex concepts are formed by syntactically combining symbols (Chomsky, 1965; Hauser et al., 2002; Evans and Levinson, 2009). Symbol systems, being more interpretable, offer superior abstraction and generalization compared to distributed representations (Launchbury, 2017). However, creating a symbol system for a domain is knowledge-intensive and can result in a brittle system plagued by the symbol grounding problem (Harnad, 1990).

In this work, we propose a *Grounded Symbol System (GSS)* as the internal representation for systematic generalization, offering a principled amalgamation of perception, syntax, and semantics, illustrated by Fig. 1. Formally, a GSS is a directed tree $T =< (x, s, v), e >$, where each node is a triplet of the grounded input x, the abstract symbol s, and the semantic meaning v. Edges represent semantic dependencies between parent and child nodes, with an edge $i \rightarrow j$ indicating dependency of node i's meaning on node j's.

Given the inherent fragility and labor-intensity of handcrafted symbol systems, grounding them on raw inputs and learning their syntax and semantics from training examples become crucial, a topic we delve into in the following sections.

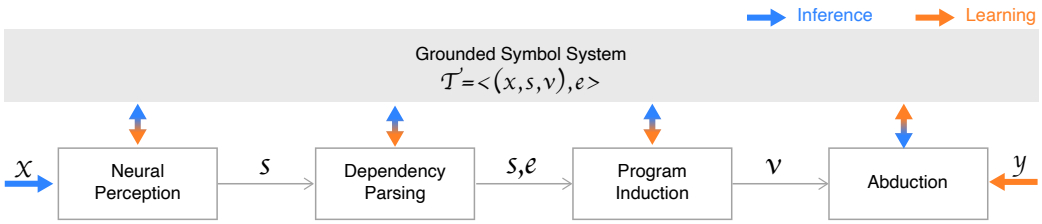


Figure 2: **Illustration of inference and learning pipeline in NSR.**

3.2 Model: Neural-Symbolic Recursive Machine (NSR)

We delineate the structure of the proposed NSR, designed to induce a GSS from the training data. As depicted in Fig. 2, NSR is composed of three trainable modules: a neural perception module to ground symbols in raw input, a dependency parser to infer dependencies between symbols, and a program synthesizer to deduce semantic meanings. Given the absence of ground-truth GSS during training, these modules must be trained end-to-end without intermediate supervision. Subsequently, we detail the three modules of NSR and discuss the end-to-end learning of NSR utilizing our introduced deduction-abduction algorithm.

Neural Perception The perception module's objective is to convert raw input x (*e.g.*, a handwritten expression) into a symbolic sequence s, represented by a list of indices. This module manages the perceptual variance inherent in raw input signals, ensuring each predicted token $w_i \in s$ corresponds to a specific segment of the input $x_i \in x$. Formally, this relationship is expressed as:

$$p(s|x;\theta_p) = \prod_i p(w_i|x_i;\theta_p) = \prod_i \text{softmax}(\phi(w_i, x_i;\theta_p)), \tag{1}$$

where $\phi(w_i, x_i;\theta_p)$ denotes a scoring function, parameterized by a neural network with parameters θ_p. The architecture of this neural network is contingent upon the nature of the raw input and can be pre-trained; for instance, a pre-trained convolutional neural network is employed for image inputs.

Dependency Parsing To infer dependencies between symbols, a transition-based neural dependency parser is utilized (Chen and Manning, 2014), a method prevalent in parsing natural language sentences. This parser, operating through a state machine, delineates possible transitions to convert the input sequence into a dependency tree, constructing it by iteratively applying predicted transitions until the parsing concludes; refer to Fig. A1 for illustration. At each step, a transition is predicted based on the state representation, derived from a local window encompassing the top elements in the stack and buffer, and their immediate children. A two-layer feed-forward neural network, given the state representation, predicts the transition. Formally, for an input sentence s, the relationship is:

$$p(e|s;\theta_s) = p(\mathcal{T}|s;\theta_s) = \prod_{t_i \in \mathcal{T}} p(t_i|c_i;\theta_s), \tag{2}$$

where θ_s represents the parser's parameters, $\mathcal{T} = \{t_1, t_2, ..., t_l\} \vDash e$ denotes the transition sequence generating the dependencies e, and c_i is the state representation at step i. A greedy decoding strategy is employed in practice to predict the transition sequence for the input sentence.

Program Induction Drawing inspiration from advancements in program induction (Ellis et al., 2021; Balog et al., 2017; Devlin et al., 2017), we employ *functional programs* to depict the semantics of symbols, conceptualizing learning as program induction. Symbolic programs, compared to solely statistical methods, offer improved generalizability and interpretability and are typically more sample-efficient. Formally, given input symbols s and their dependencies e, the relationship is defined as:

$$p(v|e, s;\theta_l) = \prod_i p(v_i|s_i, \text{children}(s_i);\theta_l) \tag{3}$$

where θ_l represents the set of programs induced for each symbol. Symbolic programs are utilized in practice, ensuring a deterministic reasoning process.

Learning symbol semantics is tantamount to searching for a program consistent with provided examples, with candidate programs composed of pre-defined primitives. Relying on Peano axioms (Peano, 1889), a universal set of primitives is identified, including `0`, `inc`, `dec`, `==`, and `if`, proven to be adequate for representing any symbolic function; refer to Sec. 3.4 for details. To expedite

the search and enhance generalization, we incorporate a minimal subset of Lisp primitives and the recursion primitive (Y-combinator (Peyton Jones, 1987)). We leverage DreamCoder (Ellis et al., 2021) for program induction, modifying it to accommodate noise in examples during the search.

Model Inference As illustrated in Fig. 2, the neural perception module first maps the input x, e.g., a handwritten expression in Fig. 1 (3) HINT, to a symbol sequence, $2 + 3 \times 9$. The dependency parsing module then parses the symbol sequence into a tree in the form of dependencies, e.g., $+ \rightarrow 2\times$, $\times \rightarrow 3\ 9$. Finally, the program induction module uses the learned programs for each symbol to calculate the values of the nodes in the tree in a bottom-up manner, e.g., $3 \times 9 \Rightarrow 27$, $2 + 27 \Rightarrow 29$.

3.3 Learning

Given the latent and non-differentiable nature of the intermediate GSS, backpropagation is impractical for learning NSR. Prior methods often use policy gradient algorithms like REINFORCE (Williams, 1992), but these have been noted for their slow or non-convergence (Liang et al., 2018; Li et al., 2020). Given the extensive space of GSS, an alternative, efficient learning algorithm is imperative.

Formally, with x as the input, $T =< (x, s, v), e >$ as the intermediate GSS, and y as the output, the likelihood of observing (x, y), marginalized over T, is:

$$p(y|x;\Theta) = \sum_{T} p(T, y|x;\Theta) = \sum_{s,e,v} p(s|x;\theta_p)p(e|s;\theta_s)p(v|s,e;\theta_l)p(y|v), \tag{4}$$

The learning objective from a maximum likelihood estimation perspective is to maximize the observed-data log-likelihood $L(x, y) = \log p(y|x)$. The derivatives of L w.r.t. $\theta_p, \theta_s, \theta_l$ are:

$$\begin{aligned}
\nabla_{\theta_p} L(x, y) &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_p} \log p(s|x;\theta_p)], \\
\nabla_{\theta_s} L(x, y) &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_s} \log p(e|s;\theta_s)], \\
\nabla_{\theta_l} L(x, y) &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_l} \log p(v|s,e;\theta_l)],
\end{aligned} \tag{5}$$

where $p(T|x, y)$ is the posterior distribution of T given (x, y), and can be rewritten as:

$$p(T|x, y) = \frac{p(T, y|x;\Theta)}{\sum_{T'} p(T', y|x;\Theta)} = \begin{cases} 0, & \text{if } T \notin Q \\ \frac{p(T|x;\Theta)}{\sum_{T'\in Q} p(T'|x;\Theta)}, & \text{if } T \in Q \end{cases} \tag{6}$$

with Q being the set of T that match y.

Given the intractability of taking expectation w.r.t. this posterior distribution, Monte Carlo sampling is used for approximation. The optimization procedure involves sampling a solution from the posterior distribution and updating each module using supervised training, addressing the challenges of sampling from a sparse distribution in large space.

Deduction-Abduction The crux of the learning procedure lies in efficient sampling from the posterior distribution $p(T|x, y)$. To address this, we introduce a *deduction-abduction* algorithm, detailed in Alg. A1. Specifically, for a given example (x, y), a greedy deduction is initially performed from x to obtain an initial GSS $T =< (x, \hat{s}, \hat{v}), \hat{e} >$. Subsequently, to align T^* with the correct result y during training, a top-down search is conducted around the neighbors of T, executing abduction across perception, syntax, and semantics; refer to Figs. 3 and A2. The search stops when the found T^* can generate the given ground truth y, or the search steps exceed the pre-defined maximum steps. Theoretically, this deduction-abduction mechanism serves as a Metropolis-Hastings sampler for $p(T|x, y)$ (Li et al., 2020).

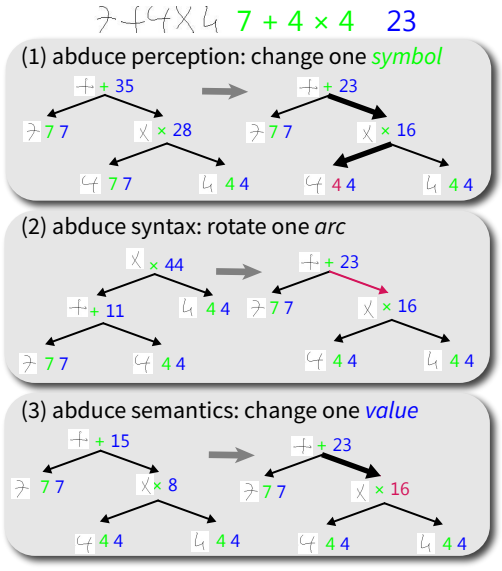


Figure 3: **Illustration of abduction in HINT over perception, syntax, and semantics.** Elements modified during abduction are emphasized in red.

3.4 EXPRESSIVENESS AND GENERALIZATION OF NSR

In this section, we delve into the properties of NSR, focusing on its expressiveness and its capability for systematic generalization, attributed to the embedded inductive biases.

Expressiveness NSR is proven to possess the expressive power necessary to model a diverse range of seq2seq tasks. Below are the theorem that support this assertion.

Theorem 3.1. Given any finite dataset $D = \{(x^i, y^i) : i = 0, ..., N\}$, an NSR exists that can aptly express D utilizing four primitives: $\{\texttt{0}, \texttt{inc}, ==, \texttt{if}\}$.

The proof of this theorem involves constructing a specialized NSR that effectively "memorizes" all examples in D through a comprehensive program:

$$\text{NSR}(x) = \texttt{if}(f_p(x)==0, y^0, \texttt{if}(f_p(x)==1, y^1, ...\texttt{if}(f_p(x)==N, y^N, \varnothing)...) \tag{7}$$

This program serves as a sophisticated lookup table, constructed using the primitives $\{\texttt{if}, ==\}$ and the index space created by $\{\texttt{0}, \texttt{inc}\}$ (proved by Lemmas C.1 and C.2). Given the universality of these four primitives across domains, NSR's ability to model a variety of seq2seq tasks is assured, offering improved transferability compared to preceding neural-symbolic approaches.

Generalization While the degenerate program outlined in Eq. (7) ensures impeccable accuracy on the training set, its generalization capabilities are limited. To foster robust generalization, it is imperative to integrate certain inductive biases. These biases should be minimal and universally applicable across domains. Drawing inspiration from compositional generalization studies (Gordon et al., 2019; Chen et al., 2020; Xie et al., 2021; Zhang et al., 2022), we postulate two essential inductive biases: *equivariance* and *compositionality*. Equivariance contributes to the model's systematicity, allowing generalizations like ``jump twice'' from {``run'', ``run twice'', ``jump''}, while compositionality enhances the model's productivity, enabling generalizations to elongated sequences such as ``run and jump twice''. The formal definitions of equivariance and compositionality are as follows:

Definition 3.1 (*Equivariance*)**.** For sets $\mathcal{X}$ and $\mathcal{Y}$, and a *permutation* group $\mathcal{P}$ with operations $T_p : \mathcal{X} \to \mathcal{X}$ and $T'_p : \mathcal{Y} \to \mathcal{Y}$, a mapping $\Phi : \mathcal{X} \to \mathcal{Y}$ is *equivariant* iff $\forall x \in \mathcal{X}, p \in \mathcal{P} : \Phi(T_p x) = T'_p \Phi(x)$.

Definition 3.2 (*Compositionality*)**.** For sets $\mathcal{X}$ and $\mathcal{Y}$, with composition operations $T_c : (\mathcal{X}, \mathcal{X}) \to \mathcal{X}$ and $T'_c : (\mathcal{Y}, \mathcal{Y}) \to \mathcal{Y}$, a mapping $\Phi : \mathcal{X} \to \mathcal{Y}$ is *compositional* iff $\exists T_c, T'_c, \forall x_1 \in \mathcal{X}, x_2 \in \mathcal{X} : \Phi(T_c(x_1, x_2)) = T'_c(\Phi(x_1), \Phi(x_2))$.

The three modules of NSR—neural perception (Eq. (1)), dependency parsing (Eq. (2)), and program induction (Eq. (3))—exhibit equivariance and compositionality, functioning as pointwise transformations based on their formulations. Models demonstrating exceptional compositional generalization, such as NeSS (Chen et al., 2020), LANE (Liu et al., 2020), and NSR, inherently possess these properties. This leads to our hypothesis regarding compositional generalization:

Hypothesis 3.1. A model achieving compositional generalization instantiates a mapping $\Phi : \mathcal{X} \to \mathcal{Y}$ that is inherently *equivariant* and *compositional*.

4 EXPERIMENTS

We assess the NSR across three benchmarks to scrutinize its capability for systematic generalization: (i) SCAN (Lake and Baroni, 2018), which translates natural language commands into sequences of actions; (ii) PCFG (Hupkes et al., 2020), which predicts the output sequences of string manipulation commands; and (iii) HINT (Li et al., 2023), which anticipates the results of handwritten arithmetic expressions. Additionally, we extend our evaluation to a compositional machine translation task (Lake and Baroni, 2018) to corroborate NSR's efficacy in addressing real-world challenges.

4.1 SCAN

The SCAN dataset (Lake and Baroni, 2018) serves as a prevalent benchmark for examining the systematic generalization of machine learning models. It is designed to translate natural language commands into action sequences, mimicking the navigation of an agent in a grid world.

Evaluation Protocols In alignment with preceding studies (Lake, 2019; Gordon et al., 2019; Chen et al., 2020), we appraise NSR under the subsequent four splits: (i) SIMPLE: the dataset is randomly partitioned into training and testing samples. (ii) LENGTH: the training set encompasses output sequences with a maximum of 22 actions, while the test set includes sequences with lengths ranging between 24 and 48. (iii) JUMP: the primitive "jump" is isolated in the training set, and the test set amalgamates "jump" with other primitives. (iv) AROUND RIGHT: the phrase "around right" is excluded from the training set, but the constituents "around" and "right" are present separately, exemplified by "around left" and "opposite right."

Baselines We juxtapose NSR against several models: (i) seq2seq (Lake and Baroni, 2018), (ii) CNN (Dessì and Baroni, 2019), (iii) Transformer (Csordás et al., 2021; Ontanón et al., 2022), (iv) equivariant seq2seq (Gordon et al., 2019)—a model that amalgamates convolutional operations with RNNs to attain local equivariance, and (v) NeSS (Chen et al., 2020)—a model that embeds a symbolic stack machine within a seq2seq framework.

Results The summarized results are presented in Tab. 1. Remarkably, both NSR and NeSS realize 100% accuracy on the splits necessitating systematic generalization. In contrast, the peak performance of other models on the LENGTH split barely reaches 20%. This stark discrepancy underscores the pivotal role and efficacy of symbolic components—specifically, the symbolic stack machine in NeSS and the GSS in NSR—in fostering systematic generalization.

Table 1: **Test accuracy across various splits of SCAN and PCFG.** The results of NeSS on PCFG are reported by modifying the source code from Chen et al. (2020) for PCFG.

models	SCAN				PCFG		
	SIMPLE	JUMP	AROUND RIGHT	LENGTH	i.i.d.	systematicity	productivity
Seq2Seq (Lake and Baroni, 2018)	99.7	1.2	2.5	13.8	79	53	30
CNN (Dessì and Baroni, 2019)	100.0	69.2	56.7	0.0	85	56	31
Transformer (Csordás et al., 2021)	-	-	-	20.0	-	96	85
Transformer (Ontanón et al., 2022)	-	0.0	-	19.6	-	83	63
equivariant Seq2seq (Gordon et al., 2019)	100.0	99.1	92.0	15.9	-	-	-
NeSS (Chen et al., 2020)	100.0	100.0	100.0	100.0	≈0	≈0	≈0
NSR (ours)	**100.0**	**100.0**	**100.0**	**100.0**	**100**	**100**	**100**

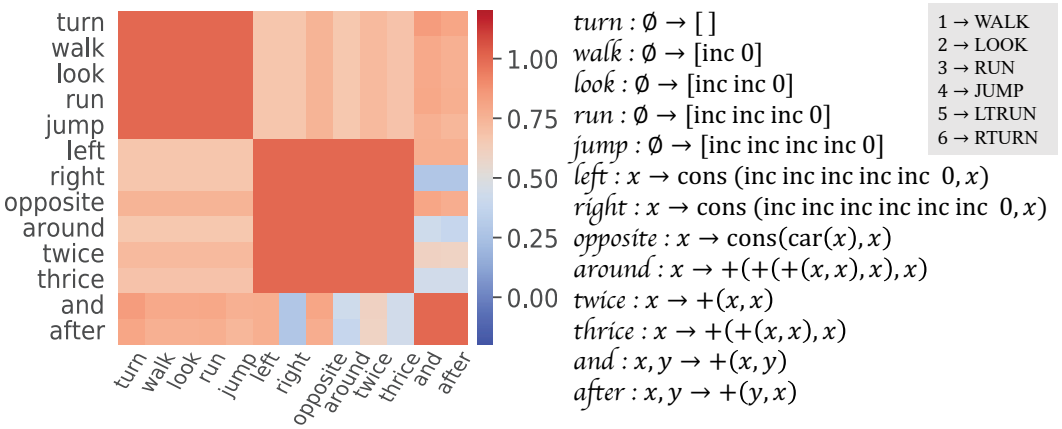
While NeSS and NSR both manifest impeccable generalization on SCAN, their foundational principles are distinctly divergent.

1. NeSS necessitates an extensive reservoir of domain-specific knowledge to meticulously craft the components of the stack machine, encompassing stack operations and category predictors. Without the incorporation of category predictors, the efficacy of NeSS plummets to approximately 20% in 3 out of 5 runs. Contrarily, NSR adopts a modular architecture, minimizing reliance on domain-specific knowledge.
2. The training regimen for NeSS is contingent on a manually curated curriculum, coupled with specialized training protocols for latent category predictors. Conversely, NSR is devoid of any prerequisite for a specialized training paradigm.

Fig. 4 elucidates the syntax and semantics assimilated by NSR from the LENGTH split in SCAN. The dependency parser of NSR, exhibiting equivariance as elucidated in Sec. 3.4, delineates distinct permutation equivalent groups syntactically among the input words: {`turn, walk, look, run, jump`}, {`left, right, opposite, around, twice, thrice`}, and {`and, after`}. It is pivotal to note that no prior information regarding these groups is imparted—they are entirely a manifestation of the learning from the training data. This is in stark contrast to the provision of pre-defined equivariant groups (Gordon et al., 2019) or the explicit integration of a category induction procedure from execution traces (Chen et al., 2020). Within the realm of the learned programs, the program synthesizer of NSR formulates an index space for the target language and discerns the accurate programs to encapsulate the semantics of each source word.

4.2 PCFG

We further assess NSR on the PCFG dataset (Hupkes et al., 2020), a task where the model is trained to predict the output of a string manipulation command. The input sequences in PCFG are synthesized using a probabilistic context-free grammar, and the corresponding output sequences are formed by



(a) Syntactic similarity amongst input words in NSR. (b) Programs induced using NSR.

Figure 4: (a) **Syntactic similarity amongst input words in NSR trained on the LENGTH split in SCAN.** The similarity between word i and word j is quantified by the percentage of test samples where substituting i with j, or vice versa, retains the dependencies as predicted by the dependency parser. (b) **Induced programs for input words using NSR.** Here, x and y represent the inputs, $\varnothing$ signifies empty inputs, `cons` appends an item to the beginning of a list, `car` retrieves the first item of a list, and + amalgamates two lists.

recursively executing the string edit operations delineated in the input sequences. The selection of input samples is designed to mirror the statistical attributes of a natural language corpus, including sentence lengths and parse tree depths.

Evaluation Protocols and Baselines The evaluation is conducted across the following splits: (i) i.i.d: where samples are randomly allocated for training and testing, (ii) systematicity: this split is designed to specifically assess the model's capability to interpret combinations of functions unseen during training, and (iii) productivity: this split tests the model's generalization to longer sequences, with training samples containing up to 8 functions and test samples having at least 9 functions. NSR is compared against (i) seq2seq (Lake and Baroni, 2018), (ii) CNN (Dessì and Baroni, 2019), (iii) Transformer (Csordás et al., 2021; Ontanón et al., 2022), and (iv) NeSS (Chen et al., 2020).

Results The results are consolidated in Tab. 1. NSR demonstrates exemplary performance, achieving 100% accuracy across all PCFG splits and surpassing the prior best-performing model (Transformer) by 4% on the "systematicity" split and by 15% on the "productivity" split. Notably, while NeSS exhibits flawless accuracy on SCAN, it encounters total failure on PCFG. A closer examination of NeSS's training on PCFG reveals that its stack operations cannot represent PCFG's binary functions, and the trace search process is hindered by PCFG's extensive vocabulary and elongated sequences. Adapting NeSS to this context would necessitate substantial domain-specific modifications and extensive refinements to both the stack machine and the training methodology.

4.3 HINT

We also evaluate NSR on HINT (Li et al., 2023), a task where the model predicts the integer result of a handwritten arithmetic expression, such as (3 + 2) × 8 → 40, without any intermediate supervision. HINT is challenging due to the high variance and ambiguity in real handwritten images, the complexity of syntax due to parentheses, and the involvement of recursive functions in semantics. The dataset includes one training set and five test subsets, each assessing different aspects of generalization across perception, syntax, and semantics.

Table 2: **Test accuracy on HINT.** Results for GRU, LSTM, and Transformer are directly cited from Li et al. (2023). NeSS results are obtained by adapting its source code to HINT.

Model	Symbol Input						Image Input					
	I	SS	LS	SL	LL	Avg.	I	SS	LS	SL	LL	Avg.
GRU	76.2	69.5	42.8	10.5	15.1	42.5	66.7	58.7	33.1	9.4	12.8	35.9
LSTM	92.9	90.9	74.9	12.1	24.3	58.9	83.9	79.7	62.0	11.2	21.0	51.5
Transformer	98.0	96.8	78.2	11.7	22.4	61.5	88.4	86.0	62.5	10.9	19.0	53.1
NeSS	≈0	≈0	≈0	≈0	≈0	≈0	-	-	-	-	-	-
NSR (ours)	**98.0**	**97.3**	**83.7**	**95.9**	**77.6**	**90.1**	**88.5**	**86.2**	**67.1**	**83.2**	**58.2**	**76.0**

Evaluation Protocols and Baselines Adhering to the protocols of Li et al. (2023), we train models on a single training set and evaluate them on five test subsets: (i) "I": expressions seen in training but composed of unseen handwritten images. (ii) "SS": unseen expressions, but their lengths and values are within the training range. (iii) "LS": expressions are longer than those in training, but their values

are within the same range. (iv) "SL": expressions have larger values, and their lengths are the same as training. (v) "LL": expressions are longer, and their values are bigger than those in training. A prediction is deemed correct only if it exactly matches the ground truth. We compare NSR against several baselines including seq2seq models like GRU, LSTM, and Transformer as reported by Li et al. (2023), and NeSS (Chen et al., 2020), with each model utilizing a ResNet-18 as the image encoder.

Results The results are summarized in Tab. 2. NSR surpasses the state-of-the-art model, Transformer, by approximately 23%. The detailed results reveal that this improvement is primarily due to better extrapolation in syntax and semantics, with NSR elevating the accuracy on the "LL" subset from 19.0% to 58.2%. The gains on the "I" and "SS" subsets are more modest, around 2%. For a more detailed insight into NSR's predictions on HINT, refer to Fig. A3. Similar to its performance on PCFG, NeSS fails on HINT, underscoring the challenges discussed in Sec. 4.2.

4.4 COMPOSITIONAL MACHINE TRANSLATION

In order to assess the applicability of NSR to real-world scenarios, we conduct a proof-of-concept experiment on a compositional machine translation task, specifically the English-French translation task, as described by Lake and Baroni (2018). This task has been a benchmark for several studies (Li et al., 2019; Chen et al., 2020; Kim, 2021) to validate the efficacy of their proposed methods in more realistic and complex domains compared to synthetic tasks like SCAN and PCFG. The complexity and ambiguity of the rules in this translation task are notably higher.

We utilize the publicly available data splits provided by Li et al. (2019). The training set comprises 10,000 English-French sentence pairs, with English sentences primarily initiating with phrases like "I am", "you are", and their respective contractions. Uniquely, the training set includes 1,000 repetitions of the sentence pair ("I am daxy", "je suis daxiste"), introducing the pseudoword "daxy". The test set, however, explores 8 different combinations of "daxy" with other phrases, such as "you are not daxy," which are absent from the training set.

Results Tab. 3 presents the results of the compositional machine translation task. We compare NSR with Seq2Seq (Lake and Baroni, 2018) and NeSS (Chen et al., 2020). It is noteworthy that two distinct French translations for "you are" are prevalent in the training set; hence, both are deemed correct in the test set. NSR, akin to NeSS, attains 100% generalization accuracy on this task, demonstrating its potential applicability to real-world tasks characterized by diverse and ambiguous rules.

Table 3: **Accuracy on compositional machine translation.**

Model	Accuracy
Seq2Seq	12.5
Transformer	14.4
NeSS	100.0
NSR (ours)	100.0

5 CONCLUSION AND DISCUSSION

We introduced NSR, a model capable of learning Grounded Symbol System from data to facilitate systematic generalization. The Grounded Symbol System offers a generalizable and interpretable representation, allowing a principled amalgamation of perception, syntax, and semantics. NSR employs a modular design, incorporating the essential inductive biases of equivariance and compositionality in each module to realize compositional generalization. We developed a probabilistic learning framework and introduced a novel deduction-abduction algorithm to enable the efficient training of NSR without GSS supervision. NSR has demonstrated superior performance across diverse domains, including semantic parsing, string manipulation, arithmetic reasoning, and compositional machine translation.

Limitations While NSR has shown impeccable accuracy on a conceptual machine translation task, we foresee challenges in its deployment for real-world tasks due to (i) the presence of noisy and abundant concepts, which may enlarge the space of the grounded symbol system and potentially decelerate the training of NSR, and (ii) the deterministic nature of the functional programs in NSR, limiting its ability to represent probabilistic semantics inherent in real-world tasks, such as the existence of multiple translations for a single sentence. Addressing these challenges remains a subject for future research.

Acknowledgment The authors would like to thank NVIDIA for their generous support of GPUs and hardware. This work is supported in part by the National Science and Technology Major Project (2022ZD0114900) and the Beijing Nova Program.

REFERENCES

Akyürek, E., Akyürek, A. F., and Andreas, J. (2020). Learning to recombine and resample data for compositional generalization. In *International Conference on Learning Representations (ICLR)*. 3

Andreas, J. (2020). Good-enough compositional data augmentation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. 3

Bahdanau, D., de Vries, H., O'Donnell, T. J., Murty, S., Beaudoin, P., Bengio, Y., and Courville, A. (2019). Closure: Assessing systematic generalization of clevr models. *arXiv preprint arXiv:1912.05783*. 2

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., and Tarlow, D. (2017). Deepcoder: Learning to write programs. In *International Conference on Learning Representations (ICLR)*. 4, A1

Carpenter, T. P., Fennema, E., Franke, M. L., Levi, L., and Empson, S. B. (1999). Children's mathematics. *Cognitively Guided*. A3

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1, 4

Chen, X., Liang, C., Yu, A. W., Song, D., and Zhou, D. (2020). Compositional generalization via neural-symbolic stack machines. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1, 2, 3, 6, 7, 8, 9, A7

Chomsky, N. (1957). Syntactic structures. In *Syntactic Structures*. De Gruyter Mouton. 1

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT press. 3

Csordás, R., Irie, K., and Schmidhuber, J. (2021). The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1, 2, 7, 8, A7

Dessì, R. and Baroni, M. (2019). Cnns found to jump around more skillfully than rnns: Compositional generalization in seq2seq convolutional networks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2, 7, 8, A7

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.-r., and Kohli, P. (2017). Robustfill: Neural program learning under noisy i/o. In *International Conference on Machine Learning (ICML)*. 4, A1

Drozdov, A., Schärli, N., Akyürek, E., Scales, N., Song, X., Chen, X., Bousquet, O., and Zhou, D. (2023). Compositional semantic parsing with large language models. *ICLR*. 1, 2

Ellis, K., Ritchie, D., Solar-Lezama, A., and Tenenbaum, J. B. (2018a). Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems (NeurIPS)*. A1

Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., and Tenenbaum, J. B. (2021). Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Programming Language Design and Implementation*. 1, 4, 5, A1, A3

Ellis, K. M., Morales, L. E., Sablé-Meyer, M., Solar Lezama, A., and Tenenbaum, J. B. (2018b). Library learning for neurally-guided bayesian program induction. In *Advances in Neural Information Processing Systems (NeurIPS)*. A1

Evans, N. and Levinson, S. C. (2009). With diversity in mind: Freeing the language sciences from universal grammar. *Behavioral and Brain Sciences*, 32(5):472–492. 3

Fodor, J. A. and Lepore, E. (2002). *The Compositionality Papers*. Oxford University Press. 3

Fodor, J. A., Pylyshyn, Z. W., et al. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71. 3

Gontier, N., Reddy, S., and Pal, C. (2022). Does entity abstraction help generative transformers reason? *TMLR*. 3

Gordon, J., Lopez-Paz, D., Baroni, M., and Bouchacourt, D. (2019). Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations (ICLR)*. 2, 6, 7, A2, A7

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346. 3

Hauser, M. D., Chomsky, N., and Fitch, W. T. (2002). The faculty of language: what is it, who has it, and how did it evolve? *Science*, 298(5598):1569–1579. 3

Herzig, J. and Berant, J. (2021). Span-based semantic parsing for compositional generalization. *ACL*. 2

Hinton, G. E. (1984). Distributed representations. Technical Report CMU-CS-84-157, Carnegie Mellon University. 3

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366. A2

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. (2020). Compositionality decomposed: how do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795. 1, 2, 6, 7

Keysers, D., Schärli, N., Scales, N., Buisman, H., Furrer, D., Kashubin, S., Momchev, N., Sinopalnikov, D., Stafiniak, L., Tihon, T., et al. (2020). Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations (ICLR)*. 1, 2

Kim, N. and Linzen, T. (2020). Cogs: A compositional generalization challenge based on semantic interpretation. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2

Kim, Y. (2021). Sequence-to-sequence learning with latent neural grammars. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3, 9

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. A3

Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., and Mansinghka, V. (2015). Picture: A probabilistic programming language for scene perception. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. A1

Lake, B. M. (2019). Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3, 7

Lake, B. M. and Baroni, M. (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning (ICML)*. 1, 2, 6, 7, 8, 9, A7

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338. A1

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40. 1

Launchbury, J. (2017). A darpa perspective on artificial intelligence. *Retrieved November*, 11:2019. 3

Li, Q., Huang, S., Hong, Y., Chen, Y., Wu, Y. N., and Zhu, S.-C. (2020). Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In *International Conference on Machine Learning (ICML)*. 5

Li, Q., Huang, S., Hong, Y., Zhu, Y., Wu, Y. N., and Zhu, S.-C. (2023). A minimalist dataset for systematic generalization of perception, syntax, and semantics. In *ICLR*. 1, 2, 6, 8, 9, A7

Li, Y., Zhao, L., Wang, J., and Hestness, J. (2019). Compositional generalization for primitive substitutions. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 9

Liang, C., Norouzi, M., Berant, J., Le, Q. V., and Lao, N. (2018). Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems (NeurIPS)*. 5

Liu, Q., An, S., Lou, J.-G., Chen, B., Lin, Z., Gao, Y., Zhou, B., Zheng, N., and Zhang, D. (2020). Compositional generalization by learning analytical expressions. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3, 6

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30. A2

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. (2018). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations (ICLR)*. 3

Marcus, G. F. (2018). *The algebraic mind: Integrating connectionism and cognitive science*. MIT press. 1, 3

Minervini, P., Bošnjak, M., Rocktäschel, T., Riedel, S., and Grefenstette, E. (2020). Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5182–5190. 3

Montague, R. (1970). Universal grammar. *Theoria*, 36(3):373–398. 1

Newell, A. (1980). Physical symbol systems. *Cognitive science*, 4(2):135–183. 3

Nye, M., Solar-Lezama, A., Tenenbaum, J., and Lake, B. M. (2020). Learning compositional rules via neural program synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3

Ontanón, S., Ainslie, J., Cvicek, V., and Fisher, Z. (2022). Making transformers solve compositional tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*. 1, 2, 7, 8, A7

Peano, G. (1889). *Arithmetices principia: Nova methodo exposita*. Fratres Bocca. 4

Peyton Jones, S. L. (1987). *The implementation of functional programming languages*. Prentice-Hall, Inc. 5

Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. *Advances in neural information processing systems*, 30. 3

Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D., and Lake, B. M. (2020). A benchmark for systematic generalization in grounded language understanding. *Advances in Neural Information Processing Systems (NeurIPS)*. 2

Russin, J., Jo, J., O'Reilly, R. C., and Bengio, Y. (2019). Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*. 2

Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. (2018). Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*. 2

Solomonoff, R. J. (1964). A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22. A1

Van Gansbeke, W., Vandenhende, S., Georgoulis, S., Proesmans, M., and Van Gool, L. (2020). Scan: Learning to classify images without labels. In *European Conference on Computer Vision (ECCV)*. A2

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256. 5

Xie, S., Ma, X., Yu, P., Zhu, Y., Wu, Y. N., and Zhu, S.-C. (2021). Halma: Humanlike abstraction learning meets affordance in rapid problem solving. *arXiv preprint arXiv:2102.11344*. 2, 6

Zhang, C., Xie, S., Jia, B., Wu, Y. N., Zhu, S.-C., and Zhu, Y. (2022). Learning algebraic representation for systematic generalization in abstract reasoning. In *European Conference on Computer Vision (ECCV)*. 6, A2

A MODEL DETAILS

Dependency Parsing Fig. A1 illustrate the process of parsing an arithmetic expression via the dependency parser. Formally, a *state* $c = (\alpha, \beta, A)$ in the dependency parser consists of a *stack* α, a *buffer* β, and a set of *dependency arcs* A. The initial state for a sequence $s = w_0 w_1 ... w_n$ is $\alpha = [\texttt{Root}], \beta = [w_0 w_1 ... w_n], A = \varnothing$. A state is regarded as terminal if the buffer is empty and the stack only contains the node `Root`. The parse tree can be derived from the dependency arcs A. Let α_i denote the i-th top element on the stack, and β_i the i-th element on the buffer. The parser defines three types of transitions between states:

- LEFT-ARC: add an arc $\alpha_1 \rightarrow \alpha_2$ to A and remove α_2 from the stack α. Precondition: $|\alpha| \geqslant 2$.
- RIGHT-ARC: add an arc $\alpha_2 \rightarrow \alpha_1$ to A and remove α_1 from the stack α. Precondition: $|\alpha| \geqslant 2$.
- SHIFT: move β_1 from the buffer β to the stack α. Precondition: $|\beta| \geqslant 1$.

The goal of the parser is to predict a transition sequence from an initial state to a terminal state. The parser predicts one transition from $\mathcal{T} = \{\text{LEFT-ARC}, \text{RIGHT-ARC}, \text{SHIFT}\}$ at a time, based on the current state $c = (\alpha, \beta, A)$. The state representation is constructed from a local window and contains following three elements: (i) The top three words on the stack and buffer: $\alpha_i, \beta_i, i = 1, 2, 3$; (ii) The first and second leftmost/rightmost children of the top two words on the stack: $lc_1(\alpha_i), rc_1(\alpha_i), lc_2(\alpha_i), rc_2(\alpha_i), i = 1, 2$; (iii) The leftmost of leftmost/rightmost of rightmost children of the top two words on the stack: $lc_1(lc_1(\alpha_i)), rc_1(rc_1(\alpha_i)), i = 1, 2$. We use a special `Null` token for non-existent elements. Each element in the state representation is embedded to a d-dimensional vector $e \in R^d$, and the full embedding matrix is denoted as $E \in R^{|\Sigma| \times d}$, where Σ is the concept space. The embedding vectors for all elements in the state are concatenated as its representation: $c = [e_1\ e_2 ... e_n] \in R^{nd}$. Given the state representation, we adopt a two-layer feed-forward neural network to predict the transition.

Program Induction Program induction, *i.e.*, synthesizing programs from input-output examples, was one of the oldest theoretical frameworks for concept learning within artificial intelligence (Solomonoff, 1964). Recent advances in program induction focus on training neural networks to guide the program search (Kulkarni et al., 2015; Lake et al., 2015; Balog et al., 2017; Devlin et al., 2017; Ellis et al., 2018a;b). For example, Balog et al. (2017) train a neural network to predict properties of the program that generated the outputs from the given inputs and then use the neural network's predictions to augment search techniques from the programming languages community. Ellis et al. (2021) released a neural-guided program induction system, *DreamCoder*, which can efficiently discover interpretable, reusable, and generalizable programs across a wide range of domains, including both classic inductive programming tasks and creative tasks such as drawing pictures and building scenes. DreamCoder adopts a "wake-sleep" Bayesian learning algorithm to extend program space with new symbolic abstractions and train the neural network on imagined and replayed problems.

To learn the semantics of a symbol c from a set of examples D_c is to find a program ρ_c composed from a set of primitives L, which maximizes the following objective:

$$\max_{\rho} p(\rho | D_c, L) \propto p(D_c | \rho)\ p(\rho | L), \tag{A1}$$

where $p(D_c|\rho)$ is the likelihood of the program ρ matching D_c, and $p(\rho|L)$ is the prior of ρ under the program space defined by the primitives L. Since finding a globally optimal program is usually intractable, the maximization in Eq. (A1) is approximated by a stochastic search process guided by a neural network, which is trained to approximate the posterior distribution $p(\rho|D_c, L)$. We refer the readers to DreamCoder (Ellis et al., 2021)[1] for more technical details.

[1] https://github.com/ellisk42/ec

B LEARNING

Derivation of Eq. (5) Take the derivative of L w.r.t. θ_p,

$$\begin{aligned}\nabla_{\theta_p} L(x,y) &= \nabla_{\theta_p} \log p(y|x) = \frac{1}{p(y|x)} \nabla_{\theta_p} p(y|x) \\ &= \sum_T \frac{p(T,y|x;\Theta)}{\sum_{T'} p(T',y|x;\Theta)} \nabla_{\theta_p} \log p(s|x;\theta_p) \\ &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_p} \log p(s|x;\theta_p)].\end{aligned} \tag{A2}$$

Similarly, for θ_s, θ_l, we have

$$\begin{aligned}\nabla_{\theta_s} L(x,y) &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_s} \log p(e|s;\theta_s)], \\ \nabla_{\theta_l} L(x,y) &= \mathbb{E}_{T\sim p(T|x,y)}[\nabla_{\theta_l} \log p(v|s,e;\theta_l)],\end{aligned} \tag{A3}$$

Deduction-Abduction Alg. A1 describes the procedure for learning NSR by the proposed deduction-abduction algorithm. Fig. 3 illustrates the one-step abduction over perception, syntax, and semantics in HINT and Fig. A2 visualizes a concrete example to illustrate the deduction-abduction process. It is similar for SCAN and PCFG.

C EXPRESSIVENESS AND GENERALIZATION OF NSR

Expressiveness

Lemma C.1. Given a finite unique set of $\{x^i : i = 0, ..., N\}$, there exists a sufficiently capable neural network f_p such that: $\forall x^i, f_p(x^i) = i$.

This lemma asserts the existence of a neural network capable of mapping every element in a finite set to a unique index, *i.e.*, $x^i \to i$, as supported by (Hornik et al., 1989; Lu et al., 2017). The parsing process in this scenario is straightforward, given that every input is mapped to a singular token.

Lemma C.2. Any index space can be constructed from the primitives $\{0, \texttt{inc}\}$.

This lemma is grounded in the fact that all indices are natural numbers, which can be recursively defined by $\{0, \texttt{inc}\}$, allowing the creation of indices for both inputs and outputs.

Generalization Equivariance and compositionality are formalized utilizing group theory, following the approaches of Gordon et al. (2019) and Zhang et al. (2022). A discrete group $\mathcal{G}$ comprises elements $\{g_1, ..., g_{|\mathcal{G}|}\}$ and a binary group operation "$\cdot$", adhering to group axioms (closure, associativity, identity, and invertibility). Equivariance is associated with a *permutation* group $\mathcal{P}$, representing permutations of a set $\mathcal{X}$. For compositionality, a *composition* operation $\mathcal{C}$ is considered, defining $T_c : (\mathcal{X}, \mathcal{X}) \to \mathcal{X}$.

The three modules of NSR—neural perception (Eq. (1)), dependency parsing (Eq. (2)), and program induction (Eq. (3))—exhibit equivariance and compositionality, functioning as pointwise transformations based on their formulations. Eqs. (1) to (3) demonstrate that in all three modules of the NSR system, the joint distribution is factorized into a product of several independent terms. This factorization process makes the modules naturally adhere to the principles of equivariance and recursiveness, as outlined in Definitions 3.1 and 3.2.

D EXPERIMENTS

D.1 EXPERIMENTAL SETUP

For tasks taking symbols as input (*i.e.*, SCAN and PCFG), the perception module is not required in NSR; For the task taking images as input, we adopt ResNet-18 as the perception module, which is pre-trained unsupervisedly (Van Gansbeke et al., 2020) on handwritten images from the training set. In the dependency parser, the token embeddings have a dimension of 50, the hidden dimension of the transition classifier is 200, and we use a dropout of 0.5. For the program induction, we adopt the

default setting in DreamCoder (Ellis et al., 2021). For learning NSR, both the ResNet-18 and the dependency parser are trained by the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 10^{-4}. NSR are trained for 100 epochs for all datasets.

Compute All training can be done using a single NVIDIA GeForce RTX 3090Ti under 24 hours.

D.2 Ablation Study

To explore how well the individual modules of NSR are learned, we perform an ablation study on HINT to analyze the performance of each module of NSR. Specifically, along with the final results, the HINT dataset also provides the symbolic sequences and parse trees for evaluation. For Neural Perception, we report the accuracy of classifying each symbol. For Dependency parsing, we report the accuracy of attaching each symbol to its correct parent, given the ground-truth symbol sequence as the input. For Program Induction, we report the accuracy of the final results, given the ground-truth symbol sequence and parse tree.

Overall, each module achieves high accuracy, as shown in Tab. A1. For Neural Perception, most errors come from the two parentheses, "(" and ")", because they are visually similar. For Dependency Parsing, we analyze the parsing accuracies for different concept groups: digits (100%), operators (95.85%), and parentheses (64.28%). The parsing accuracy of parentheses is much lower than those of digits and operators. We think this is because, as long as digits and operators are correctly parsed in the parsing tree, where to attach the parentheses does not influence the final results because parentheses have no semantic meaning. For Program Induction, we can manually verify that the induced programs (Fig. 4) have correct semantics. The errors are caused by exceeding the recursion limit when calling the program for multiplication. The above analysis is also verified by the qualitative examples in Fig. A3.

D.3 Qualitative Examples

Figs. A3 and A4 show several examples of the NSR predictions on SCAN and HINT.

Fig. A5 illustrates the evolution of semantics along the training of NSR in HINT. This pattern is highly in accordance with how children learn arithmetic in developmental psychology (Carpenter et al., 1999): The model first masters the semantics of digits as *counting*, then learns $+$ and $-$ as recursive counting, and finally figures out how to define $\times$ and $\div$ based on $+$ and $-$. Crucially, $\times$ and $\div$ are impossible to be correctly learned before mastering $+$ and $-$. The model is endowed with such an incremental learning capability since the program induction module allows the semantics of concepts to be built compositionally from those learned earlier (Ellis et al., 2021).

ID	Stack Buffer	Transition	Dependency
0	3 + 4 × 2	Shift	
1	3 + 4 × 2	Shift	
2	3 + 4 × 2	Left-Arc	3 ← +
3	+ 4 × 2	Shift	
4	+ 4 × 2	Shift	
5	+ 4 × 2	Left-Arc	4 ←×
6	+ × 2	Shift	
7	+ × 2	Right-Arc	×→ 2
8	+ ×	Right-Arc	+→×

Start: σ = [ROOT], β = w_1, ..., w_n, A = ∅
1. Shift σ, w_i|β, A ➔ σ|w_i, β, A
2. Left-Arc$_r$ σ|w_i|w_j, β, A ➔ σ|w_j, β, AU{$r(w_j,w_i)$}
3. Right-Arc$_r$ σ|w_i|w_j, β, A ➔ σ|w_i, β, AU{$r(w_i,w_j)$}
Finish: σ = [w], β = ∅

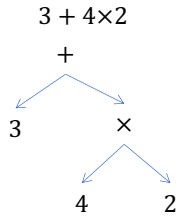


Figure A1: **Applying the transition-based dependency parser to an example of HINT.** It is similar for SCAN and PCFG.

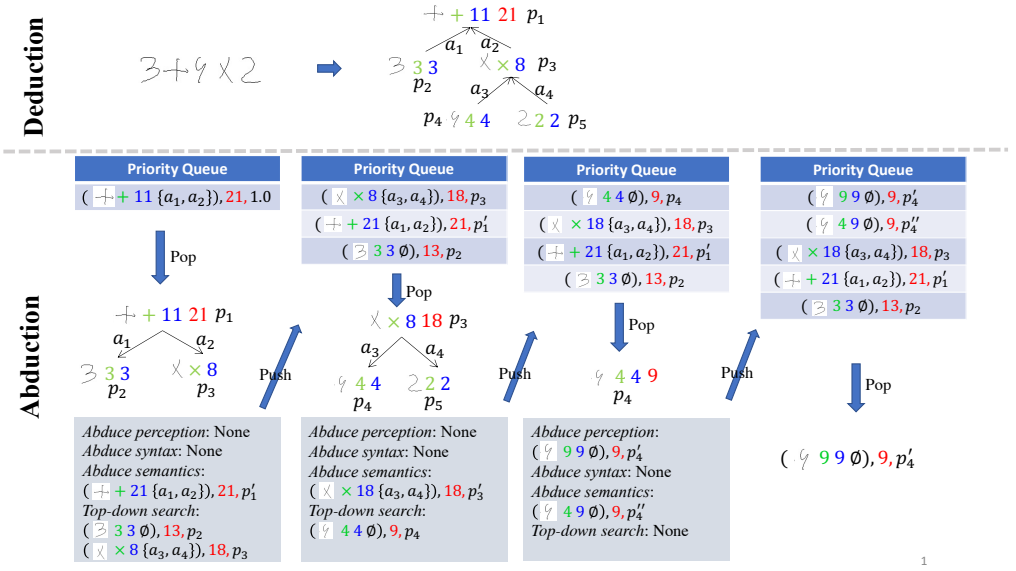
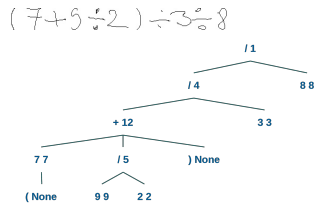


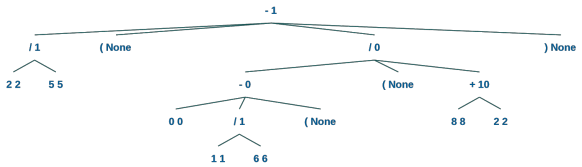
Figure A2: **An illustration of the deduction-abduction process for an example of HINT.** Given a handwritten expression, the system performs a greedy deduction to propose an initial solution, generating a wrong result. In abduction, the root node, paired with the ground-truth result, is first pushed to the priority queue. The abduction over perception, syntax, and semantics is performed on the popped node to generate possible revisions. A top-down search is also applied to propagate the expected value to its children. All possible revisions are then pushed into the priority queue. This process is repeated until we find the most likely revision for the initial solution.

Test subset I

GT: (7+9/2)/3/8 = 1 PD: (7+9/2)/3/8 = 1

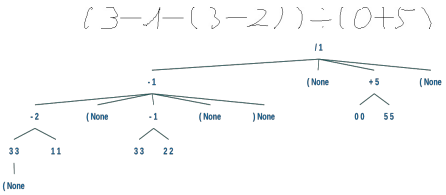


GT: 2/5-(0-1/6)/(8+2) = 1 PD: 2/5-(0-1/6(/(8+2) = 1

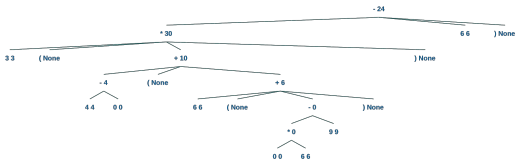


Test subset SS

GT: (3-1-(3-2))/(0+5) = 1 PD: (3-1-(3-2()/(0+5(= 1

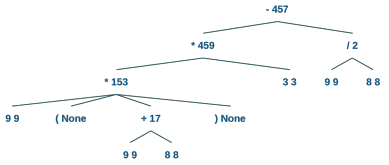


GT: 3*(4-0+(6+(0*6-9))-6) = 12 PD: 3*(4-0+(6+(0*6-9))-6) = 24

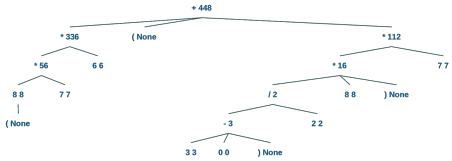


Test subset SL

GT: 9*(9+8)*3-9/8 = 457 PD: 9*(9+8)*3-9/8 = 457

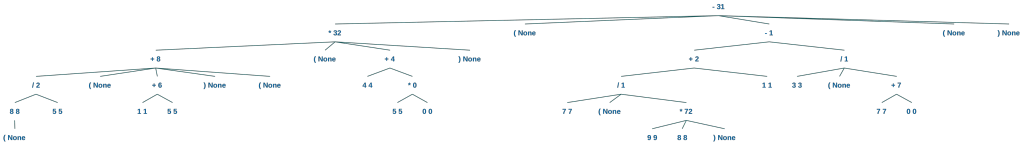


GT: (8*7*6+(3-0)/2*8)*7 = 2464 PD: (8*7*6+(3-0)/2*8)*7 = 448



Test subset LS

GT: (8/5+(1+5))*(4+5*0)-(7/(9*8)+1-3/(7+0)) = 31 PD: (8/5+(1+5)(*(4+5*0)-(7/(9*8)+1-3/(7+0() = 31



Test subset LL

GT: (8*7-5/5)*(3-(2-1)+1)/(9*1*(8+1)+(9+3)-0) = 2 PD: (8*7-5/5)*(3-(2-1)+1)/(9*1*(8+1)/(9+3)-0) = 24

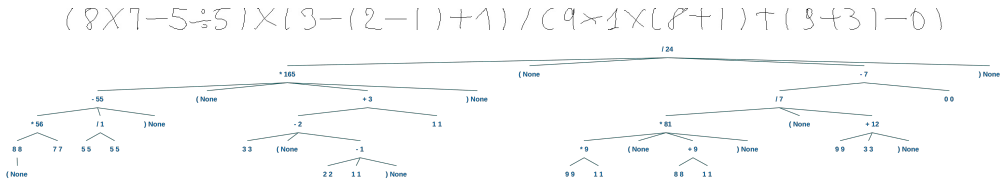


Figure A3: **Examples of NSR predictions on the test set of HINT.** "GT" and "PD" denote "ground-truth" and "prediction," respectively. Each node in the tree is a tuple of (symbol, value).

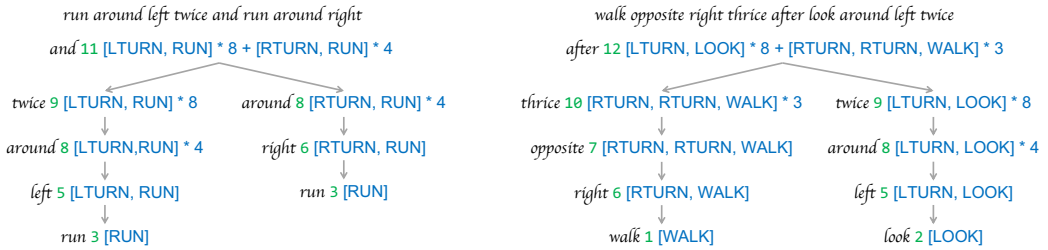


Figure A4: **Examples of NSR predictions on the test set of the SCAN LENGTH split.** We use * (repeating the list) and + (concatenating two lists) to shorten the outputs for easier interpretation.

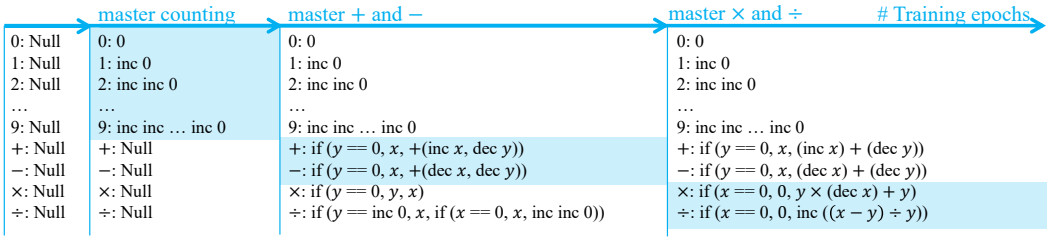


Figure A5: **The evolution of learned programs in NSR for HINT.** The recursive programs in DreamCoder are represented by lambda calculus (a.k.a. λ-calculus) with Y-combinator. Here, we translate the induced programs into pseudo code for easier interpretation. Note that there might be different yet functionally-equivalent programs to represent the semantics of a symbol; we only visualize a plausible one here.

Table A1: **Accuracy of the individual modules of NSR on the HINT dataset.**

Module	Neural Perception	Dependency Parsing	Program Induction
Accuracy	93.51	88.10	98.47

Table A2: **The test accuracy on different splits of SCAN and PCFG.** The results of NeSS on PCFG are reported by adapting the source code from Chen et al. (2020) on PCFG. Reported accuracy (%) is the average of 5 runs with standard deviation if available.

models	**SCAN**				**PCFG**		
	SIMPLE	**JUMP**	**AROUND RIGHT**	**LENGTH**	**i.i.d.**	**systematicity**	**productivity**
Seq2Seq (Lake and Baroni, 2018)	99.7	1.2	2.5	13.8	79	53	30
CNN (Dessì and Baroni, 2019)	100.0±0.0	69.2±8.2	56.7±10.2	0.0±0.0	85	56	31
Transformer (Csordás et al., 2021)	-	-	-	20.0	-	96±1	85±1
Transformer (Ontanón et al., 2022)	-	0.0	-	19.6	-	83	63
equivariant Seq2seq (Gordon et al., 2019)	100.0	99.1±0.04	92.0±0.24	15.9±3.2	-	-	-
NeSS (Chen et al., 2020)	100.0	100.0	100.0	100.0	≈0	≈0	≈0
NSR (ours)	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100±0**	**100±0**	**100±0**

Table A3: **The test accuracy on HINT.** We directly cite the results of GRU, LSTM, and Transformer from Li et al. (2023). The results of NeSS are reported by adapting its source code on HINT. Reported accuracy (%) is the median and standard deviation of 5 runs.

Model	**Symbol Input**						**Image Input**					
	I	**SS**	**LS**	**SL**	**LL**	**Avg.**	**I**	**SS**	**LS**	**SL**	**LL**	**Avg.**
GRU	76.2±0.6	69.5±0.6	42.8±1.5	10.5±0.2	15.1±1.2	42.5±0.7	66.7±2.0	58.7±2.2	33.1±2.7	9.4±0.3	12.8±1.0	35.9±1.6
LSTM	92.9±1.4	90.9±1.1	74.9±1.5	12.1±0.2	24.3±0.3	58.9±0.7	83.9±0.9	79.7±0.8	62.0±2.5	11.2±0.1	21.0±0.8	51.5±1.0
Transformer	98.0±0.3	96.8±0.6	78.2±2.9	11.7±0.3	22.4±1.1	61.5±0.9	88.4±1.3	86.0±1.3	62.5±4.1	10.9±0.2	19.0±1.0	53.1±1.6
NeSS	≈0	≈0	≈0	≈0	≈0	≈0	-	-	-	-	-	-
NSR (ours)	**98.0±0.2**	**97.3±0.5**	**83.7±1.2**	**95.9±4.6**	**77.6±3.1**	**90.1±2.7**	**88.5±1.0**	**86.2±0.9**	**67.1±2.4**	**83.2±3.9**	**58.2±3.3**	**76.0±2.6**

Algorithm A1: Learning by Deduction-Abduction

Input : Training set $D = (x_i, y_i) : i = 1, 2, ..., N$

Output : $\theta_p^{(T)}, \theta_s^{(T)}, \theta_l^{(T)}$

Initial Module: perception $\theta_p^{(0)}$, syntax $\theta_s^{(0)}$, semantics $\theta_l^{(0)}$
for $t \leftarrow 0$ *to* T **do**
  Buffer $\mathcal{B} \leftarrow \varnothing$
  foreach $(x, y) \in D$ **do**
    $T \leftarrow \textsc{Deduce}(x, \theta_p^{(t)}, \theta_s^{(t)}, \theta_l^{(t)})$
    $T^* \leftarrow \textsc{Abduce}(T, y)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup T^*$
  $\theta_p^{(t+1)}, \theta_s^{(t+1)}, \theta_l^{(t+1)} \leftarrow \text{learn}(\mathcal{B}, \theta_p^{(t)}, \theta_s^{(t)}, \theta_l^{(t)})$
return $\theta_p^{(T)}, \theta_s^{(T)}, \theta_l^{(T)}$

Function $\textsc{Deduce}(x, \theta_p, \theta_s, \theta_l)$ **:**
  Sample $\hat{s} \sim p(s|x; \theta_p), \hat{e} \sim p(e|\hat{s}; \theta_s), \hat{v} = f(\hat{s}, \hat{e}; \theta_l)$
  return $T =< (x, \hat{s}, \hat{v}), \hat{e} >$

Function $\textsc{Abduce}(T, y)$ **:**
  $Q \leftarrow$ PriorityQueue()
  $Q.push(\text{root}(T), y, 1.0)$
  while Q *is not empty* **do**
    $A, y_A, p \leftarrow Q.pop()$
    $A \leftarrow (x, w, v, arcs)$
    if $A.v == y_A$ **then**
      return $T(A)$
    `// Abduce perception`
    foreach $w' \in \Sigma$ **do**
      $A' \leftarrow A(w \rightarrow w')$
      if $A'.v == y_A$ **then**
        $Q.push(A', y_A, p(A'))$
    `// Abduce syntax`
    foreach $arc \in arcs$ **do**
      $A' \leftarrow \text{rotate}(A, arc)$
      if $A'.v == y_A$ **then**
        $Q.push(A', y_A, p(A'))$
    `// Abduce semantics`
    $A' \leftarrow A(v \rightarrow y_A)$
    $Q.push(A', y_A, p(A'))$
    `// Top-down search`
    foreach $B \in$ *children*(A) **do**
      $y_B \leftarrow \text{Solve}(B, A, y_A | \theta_l(A.w))$
      $Q.push(B, y_B, p(B))$