Grammar-based Pattern Matching and Type Checking for Difference Data Structures

Naoki Yamamoto, Kazunori Ueda

Waseda University, Tokyo, Japan

APLAS2024-NIER@Kyoto / PPDP 2024@Milan (paper in ACM DL)

October 25, 2024

Historical Background: Difference Lists

- Used since early days of Prolog (for NLP etc.)
  - for constant-time list concatenation
  - modular construction of a list from building blocks

`[1,2,3|X]-X`

- a.k.a. **list segments** in Separation Logic
- Difference is ubiquitous:

time vs. duration

position vs. displacement

sequents $\vdash$

implications

contexts C[ ]

functions

continuations

Q. Can be generalized to **richer data structures**?
→ **Difference Data Structures (DDSs)**

Graph structures

- Generalizes algebraic data types
- Abstracts pointer structures
- Not just data structures (passive);
  encompasses process/control structures (autonomous)
  - unifies data, functions & HO, processes, messages, proofs, ...
- supported poorly by high-level languages
  at the “right” level of abstraction
  - cf. algebraic graph transformation formalisms
  - https://www.ueda.info.waseda.ac.jp/~ueda/pub/ICGT2024-v1.pdf
    (ICGT 2024 tutorial on GT from the PL perspective)

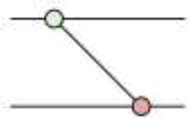
Graph transformation in different guises

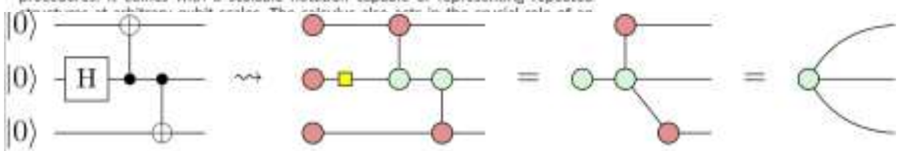
Home Tutorial Publications The ZX Seminar Accessibility Map PyZX Demo

The ZX-calculus

The ZX-calculus is a graphical language that goes beyond circuit diagrams. It 'splits the atom' of well-known quantum logic gates to reveal the compositional structure inside. The calculus works by generalising the ideas of Z and X operations, allowing us to break out of the circuit model while maintaining soundness of reasoning. In doing so we can show properties of circuits, entanglement states, and protocols, in a visually succinct but logically complete manner.

The ZX-calculus is forging the next generation of quantum software. Using the calculus gives optimisation strategies that performs state-of-the-art T-count reduction (an important metric for fault-tolerant computing) and gate compilation. The generators of the calculus correspond closely to the basic operations of lattice surgery in the surface code, giving a visual design and verification language for these codes; and ZX has also been used to discover novel error correction procedures. It comes with a scalable notation capable of representing repeated





GReTA - Graph Transformation Theory and Applications Home Seminars How to participate Team GReTA-ExACT

Graph Rewriting as a Foundation for Science and Technology (and the Universe)

Stephen Wolfram

Follow

String Diagram Rewrite Theory I: Rewriting with Frobenius Structure

FILIPPO BONCHI and FABIO GADDUCCI, University of Pisa
ALEKS KISSINGER, University of Oxford
PAWEL SOBOCINSKI, Tallinn University of Technology
FABIO ZANASI, University College London

Overview

Difference Data Structures (DDSs):

Unified framework for handling diverse concepts

- (linear) functions, continuations, evaluation contexts, ...

Problem: How to formulate **types for DDSs** ?

Contribution: **LMNtalGG** and **Difference Types**, a typing framework for DDSs based on **G**raph **G**rammars

- Implemented on a graph rewriting language **LMNtal**
- Applications:
  - (runtime) **subgraph (pattern) matching**
  - (compile-time) **type checking of rewrite rules**

Outline

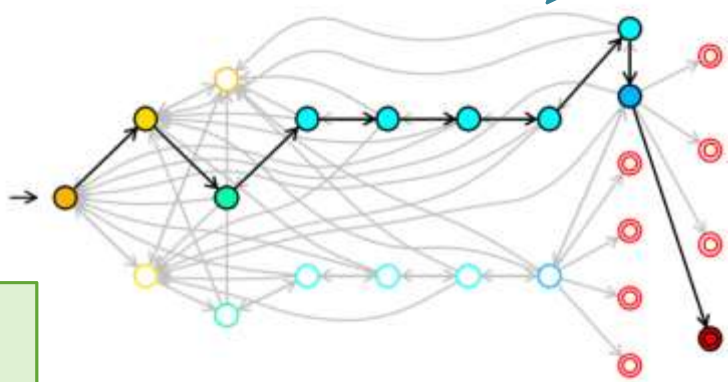
1. LMNtal: a graph rewriting language
2. LMNtalGG and Difference Types
3. Classifying LMNtalGGs: Disjoint & Indexed
4. Applications: Pattern Matching & Static Type Checking
5. Type Checking of Functional Atoms

LMNtal: a graph rewriting language

- A **Programming language** and a **modeling language**
- Full-fledged implementation **SLIM/LaViT** provides both
  - ordinary **execution** and
  - parallel **model checker** (up to ~10^9 states) **with state space visualizer**

Portal: https://bit.ly/lmntal-portal
Toolchain: https://github.com/lmntal

State space of the water jug problem

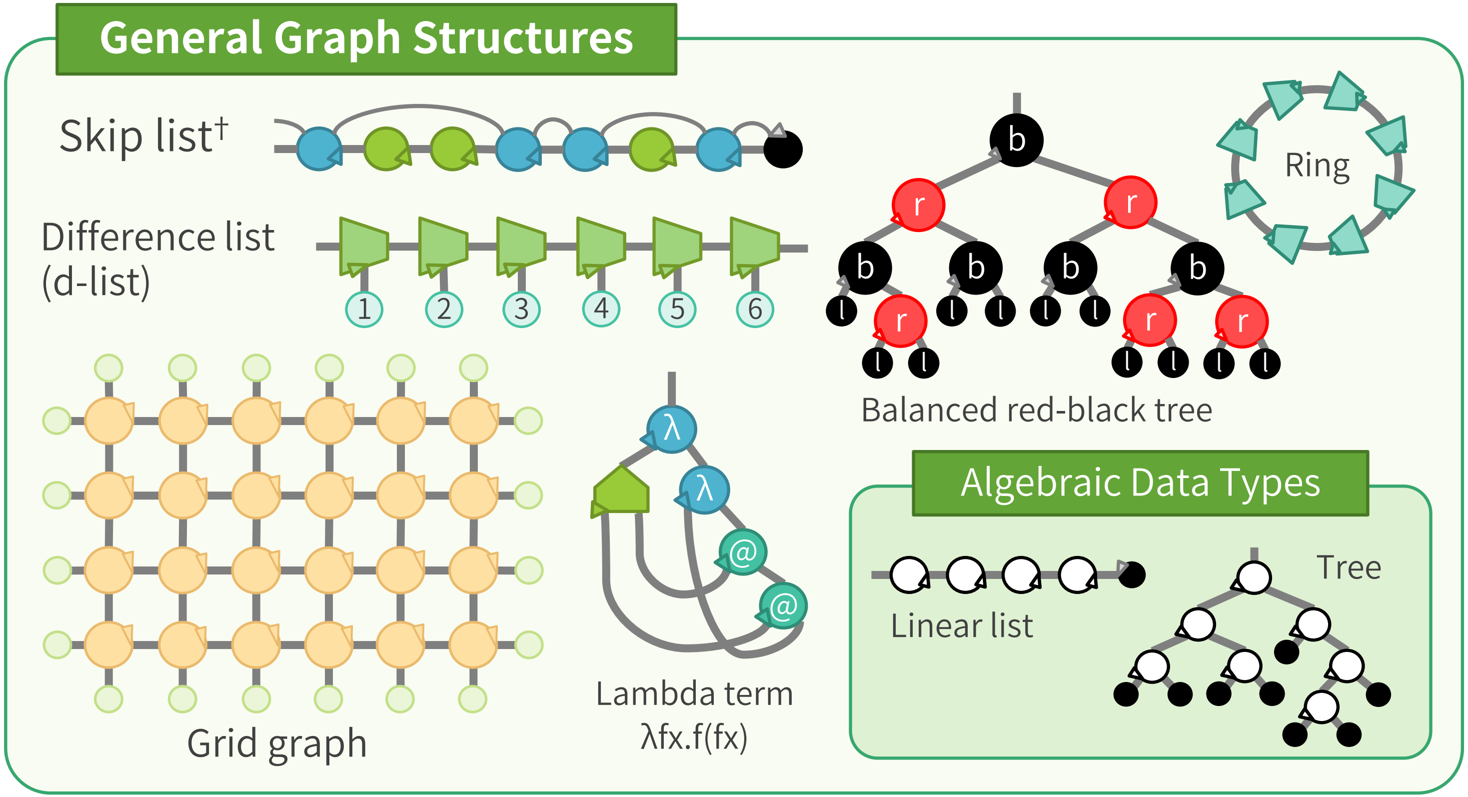


K. Ueda: LMNtal as a hierarchical logic programming language. Theoretical Computer Science 410(46), 2009.

M. Gocho et al.: Evolution of the LMNtal Runtime to a Parallel Model Checker. Computer Software 28(4), 2011.

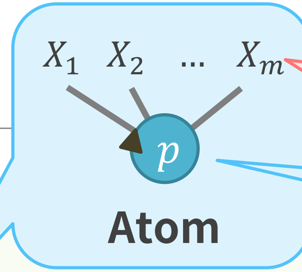
LMNtal: powerful data structures

We can handle non-algebraic data types without dangling pointers



† W. Pugh: Skip lists: A probabilistic alternative to balanced trees, C. ACM, 33(6), 1990.

LMNtal: Syntax



totally ordered links (= **port graph**)

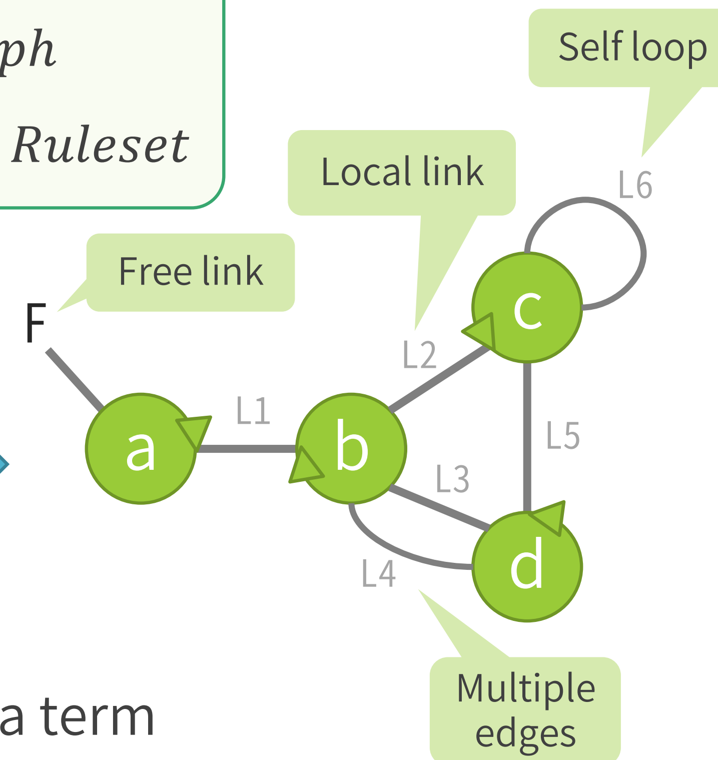
Functor p/m

$$
\begin{aligned}
\mathit{Process} &::= \mathit{Graph}\,,\mathit{Ruleset}\\
\mathit{Graph} &::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid \mathit{Graph}\,,\mathit{Graph}\\
\mathit{Ruleset} &::= \mathbf{0} \mid \mathit{Graph}\ \text{:-}\ \mathit{Graph} \mid \mathit{Ruleset}\,,\mathit{Ruleset}
\end{aligned}
$$

Null

Rewrite Rule

```
a(L1,F),b(L1,L2,L3,L4),
c(L2,L5,L6,L6),d(L5,L3,L4)
```



Link Condition:
Each link name must occur at most twice in a term

LMNtal: Structural Congruence

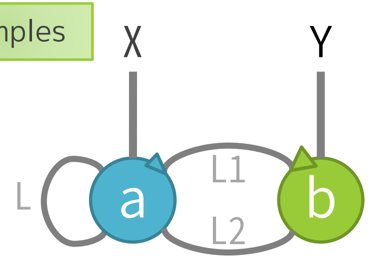
- Gives the interpretation of **LMNtal terms** as **graphs**
  - cf. standard graph theory considers graphs **up to isomorphism**

$$
\begin{array}{llcl}
\text{(E1)} & \mathbf{0}, P & \equiv & P \\
\text{(E2)} & P, Q & \equiv & Q, P \\
\text{(E3)} & P, (Q, R) & \equiv & (P, Q), R \\
\text{(E4)} & P & \equiv & P[Y/X] \\
 & & & \text{(if } X \text{ is a local link of } P\text{)} \\
\text{(E5)} & P \equiv P' & \Rightarrow & P, Q \equiv P', Q
\end{array}
$$

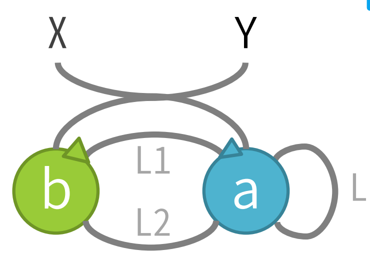
Connector: A binary infix atom $X = Y$ fuses two links

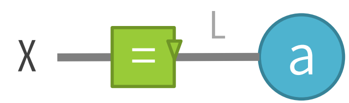
$$
\begin{array}{llcl}
\text{(E7)} & X{=}X & \equiv & \mathbf{0} \\
\text{(E8)} & X{=}Y & \equiv & Y{=}X \\
\text{(E9)} & X{=}Y, P & \equiv & P[Y/X] \\
 & \multicolumn{3}{l}{\text{(if } P \text{ is an atom and } X \text{ is a free link of } P\text{)}}
\end{array}
$$

Examples

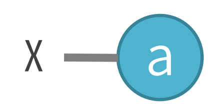


(E2)
$\equiv$





(E9)
$\equiv$



`a(L1,L2,L,L,X),b(L1,L2,Y)` `b(L1,L2,Y),a(L1,L2,L,L,X)` `L=X,a(L)` `a(X)`

(E6) & (E10) are omitted because these are rules for membranes (boxes)

LMNtal: Reduction Relation (small-step semantics)

Structural Rules

$$\text{(R1)}\quad \frac{G_1 \xrightarrow{T\,:-\,U} G_1'}{G_1\,,G_2 \xrightarrow{T\,:-\,U} G_1'\,,G_2} \qquad\qquad \text{(R3)}\quad \frac{G_2 \equiv G_1 \quad G_1 \xrightarrow{T\,:-\,U} G_1' \quad G_1' \equiv G_2'}{G_2 \xrightarrow{T\,:-\,U} G_2'}$$

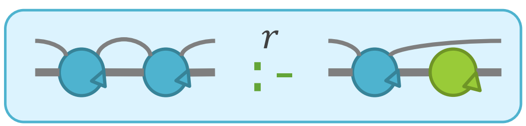
Main rule

$$\text{(R6)}\quad T \xrightarrow{T\,:-\,U} U$$

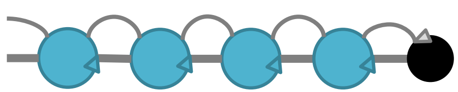
Non-determinism

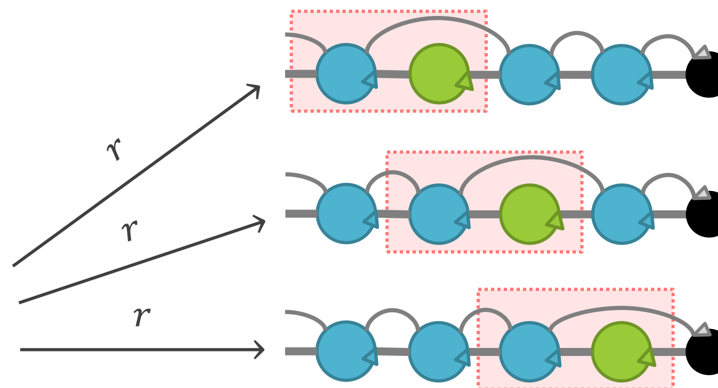
Example

Rewrite rule



Initial graph





Outline

1. LMNtal: a graph rewriting language
2. LMNtalGG and Difference Types
3. Classifying LMNtalGGs: Disjoint & Indexed
4. Applications: Pattern Matching & Static Type Checking
5. Type Checking of Functional Atoms

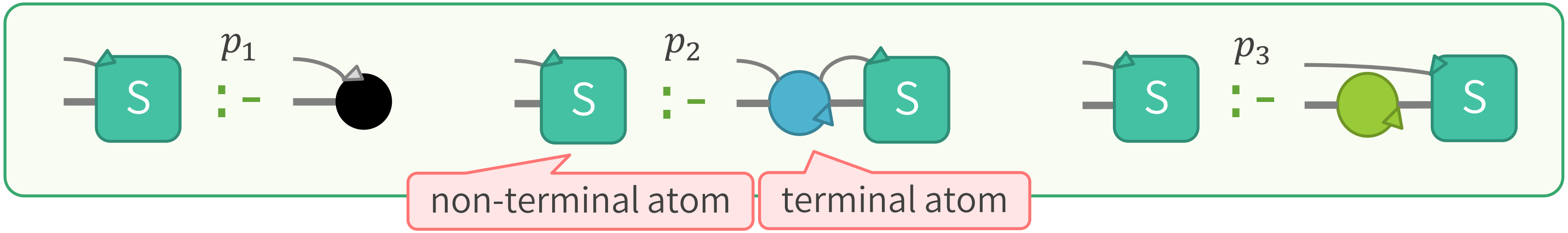
LMNtalGG: Graph Grammar on LMNtal

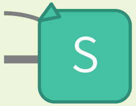
- Inductively defines a set of graphs **by a context-free graph grammar**

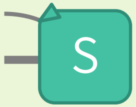
Formal Language Theory	LMNtal
Production rules	**Rewrite rules**
Symbols	Functors*

* Pairs of name and arity of atoms

Example: Production rules of skip lists



When we repeatedly apply the rules above on 

and get a graph without 

, the resulting graph is a skip list.

LMNtalGG: Context-freeness assumption

- We assume all production rules are **context-free**
  - i.e., the LHS must be a single (non-terminal) atom
- and refer to **a set of production rules** as **a grammar**
  - Every non-terminal atom can be the **start symbol**
  - The sets of **non-terminal/terminal symbols** are automatically determined by the grammar

$$\mathrm{N}(P) \triangleq \bigcup_{(\alpha \,:\!-\, \beta)\in P} \mathrm{Funct}(\alpha), \quad \mathrm{T}(P) \triangleq \mathrm{Funct}(P) \setminus \mathrm{N}(P).$$

Non-terminal symbols

Terminal symbols

LMNtalGG: Difference Types

- Types with the concept of **difference** based on LMNtalGG
  - The idea of difference lists generalized to graphs

“The graph G has the type $\alpha - \beta$ with the grammar P”

P may be omitted if clear from the context

$$G :_P \alpha - \beta \overset{\text{def}}{\Longleftrightarrow} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

where

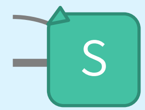
α is a single non-terminal atom
β consists only of non-terminal atoms
G doesn't include non-terminal atoms

LMNtalGG: Example

$$G :_P \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

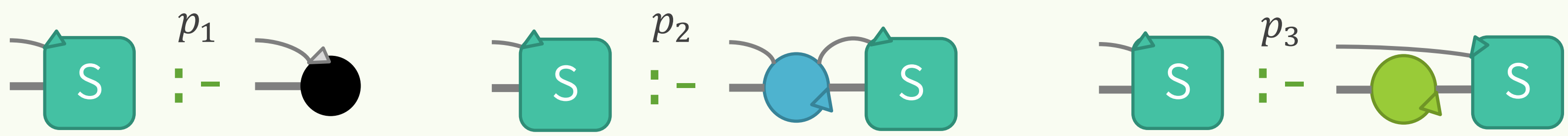
Applying the production rules to the start symbol

Start Symbol

In this example, 

is the only non-terminal symbol and all the other atoms are terminal

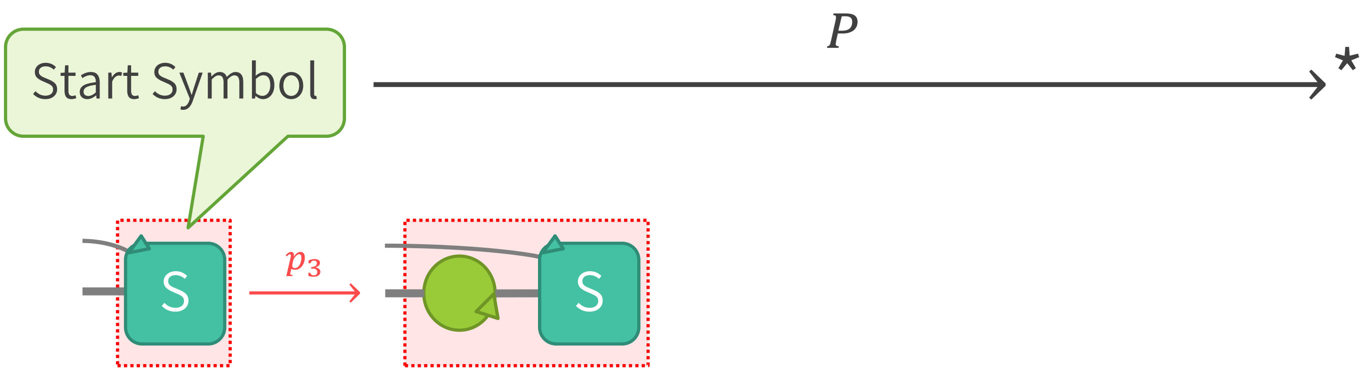
Production rules of skip lists



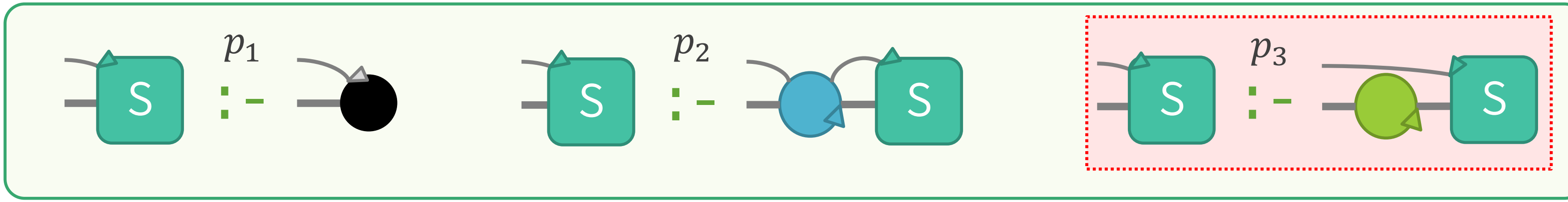
LMNtalGG: Example

$$G :_P \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

Applying the production rules to the start symbol



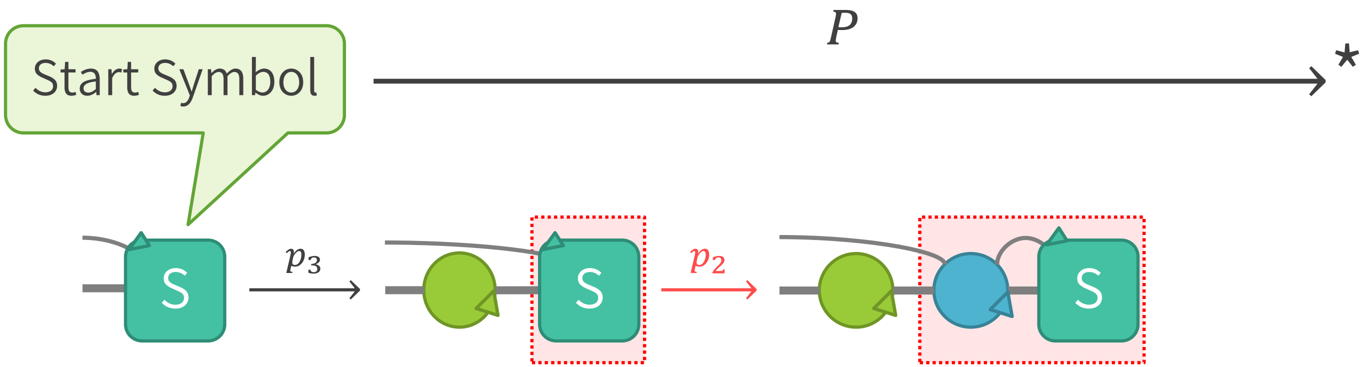
Production rules of skip lists



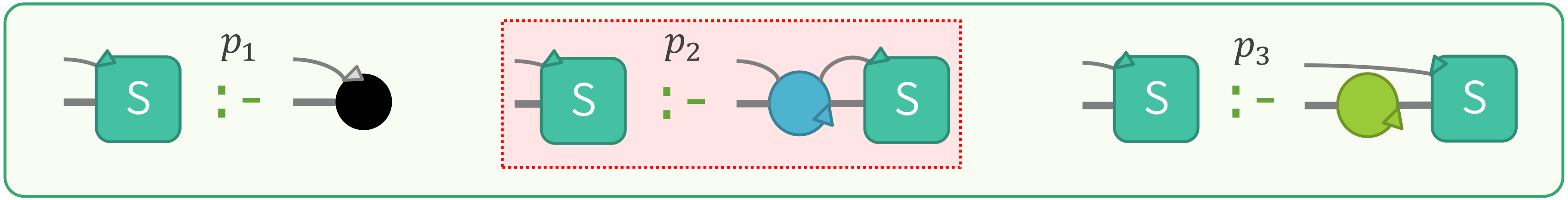
LMNtalGG: Example

$$G :_{P} \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

Applying the production rules to the start symbol



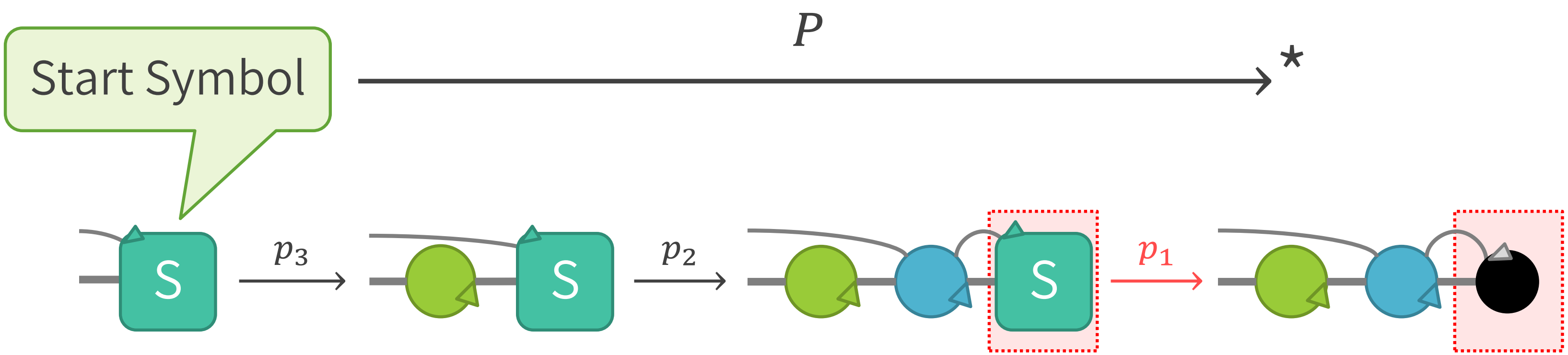
Production rules of skip lists



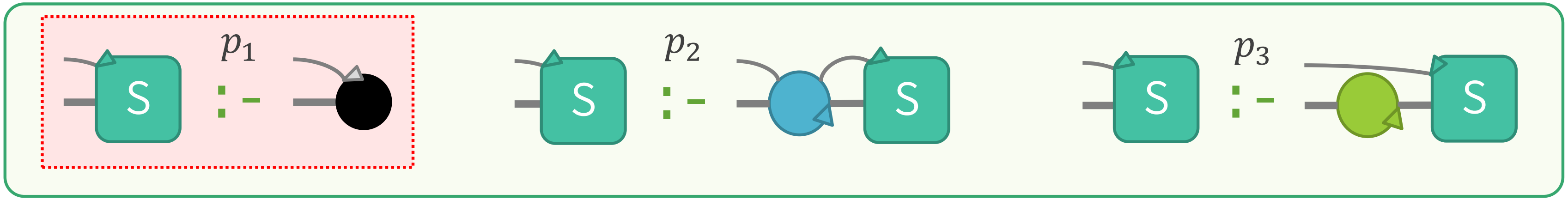
LMNtalGG: Example

$$G :_P \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

Applying the production rules to the start symbol



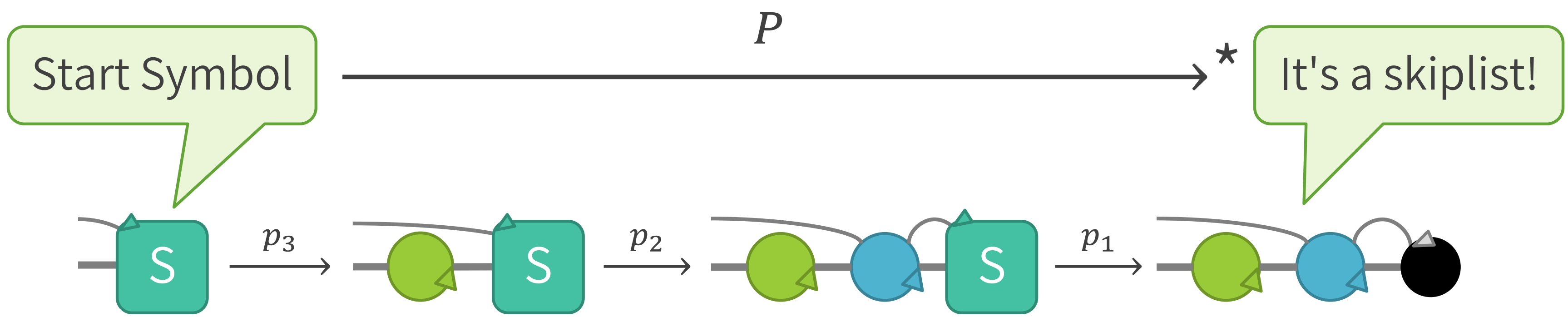
Production rules of skip lists

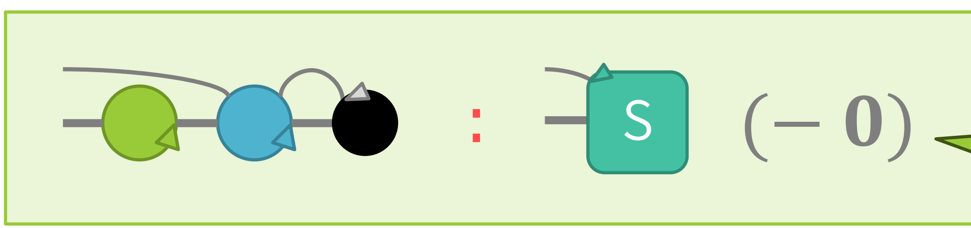


LMNtalGG: Example

$$G :_P \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

Resulting in a graph without non-terminal symbols



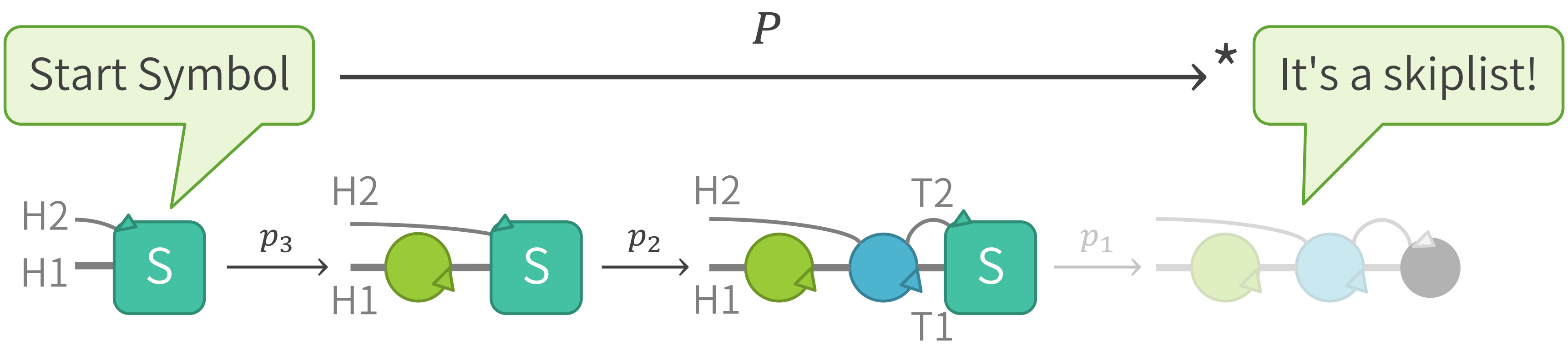


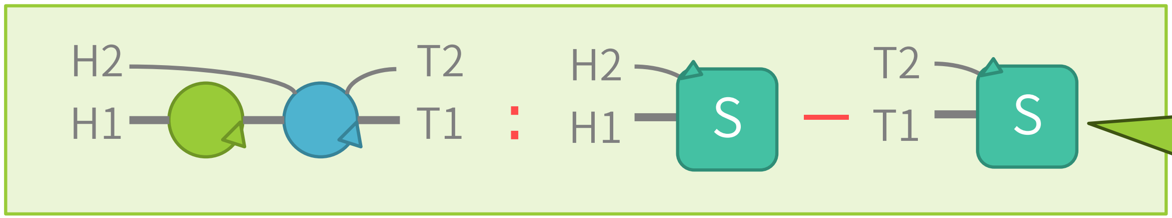
The graph on the left is a skip list

Difference Types

$$G :_P \alpha - \beta \overset{\text{def}}{\iff} \alpha \xrightarrow{P}{}^{*} (G, \beta)$$

Difference data structures can also be typed!





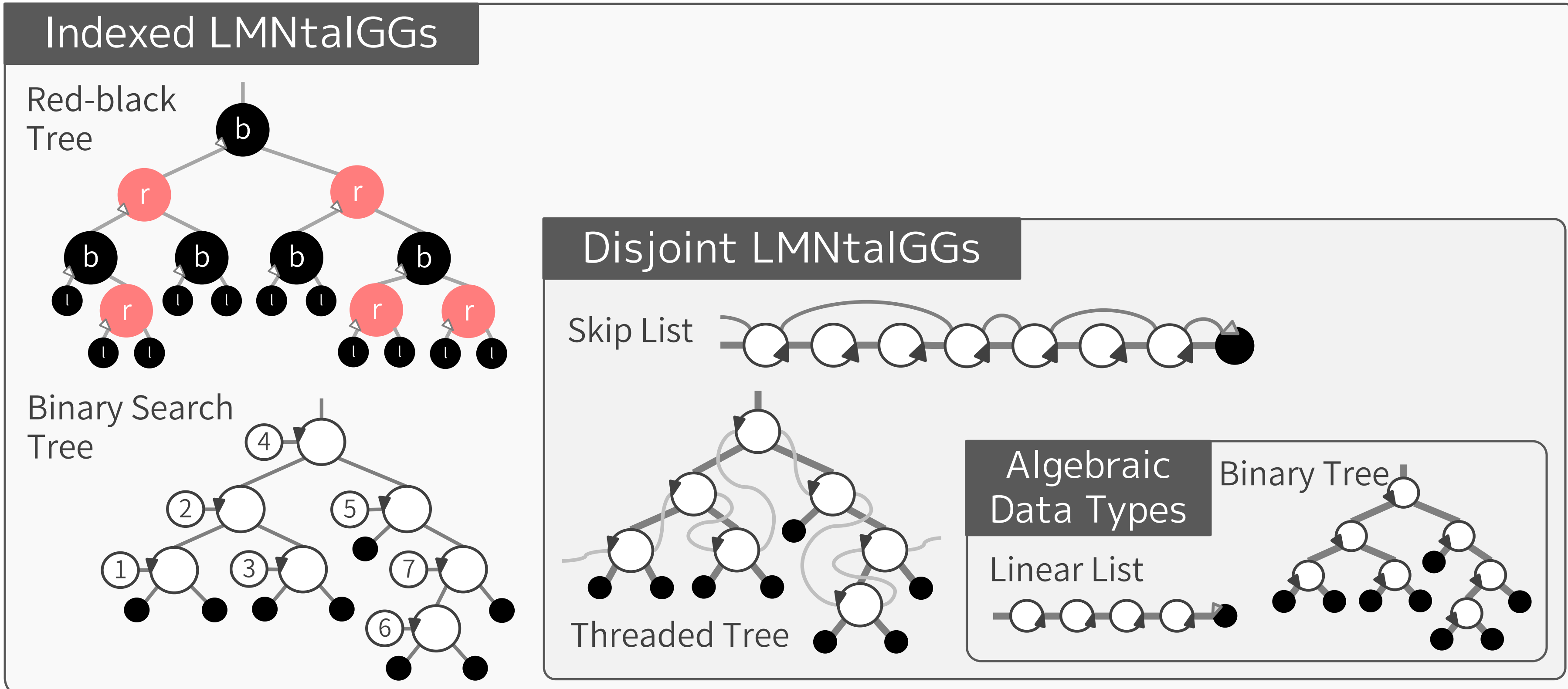
The graph on the left is a **difference** skip list

Outline

1. LMNtal: a graph rewriting language
2. LMNtalGG and Difference Types
3. Classifying LMNtalGGs: Disjoint & Indexed
4. Applications: Pattern Matching & Static Type Checking
5. Type Checking of Functional Atoms

Classifying LMNtalGGs

Two useful classes of LMNtalGGs

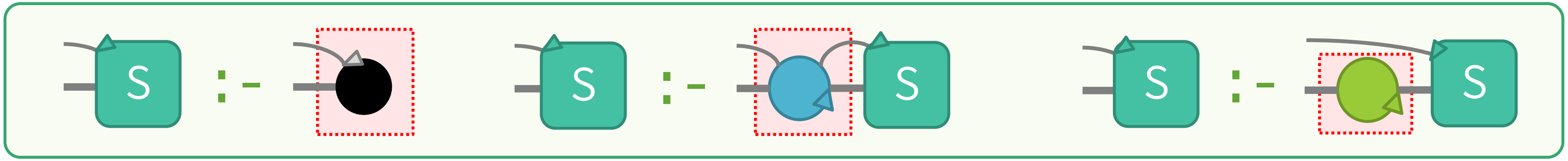


Basic Class: Disjoint LMNtalGG

A grammar P is **disjoint**

$\overset{\text{def}}{\Longleftrightarrow}$ RHS of each rule contains **exactly one** terminal symbol that never appears in the RHSs of other production rules

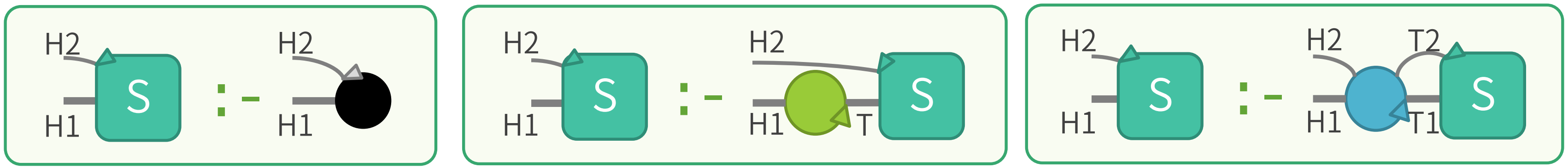
- a.k.a. **inversion property** in standard type theory
  - Types of subterms can be inferred from the top-level constructor
- Example: The grammar of skip lists is disjoint



For disjoint LMNtalGGs, we can derive types of graphs uniquely

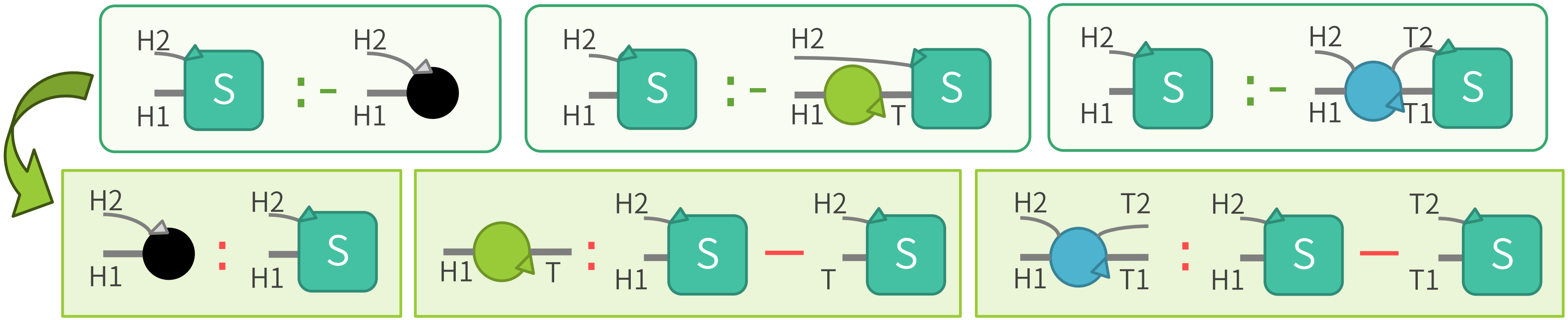
Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules



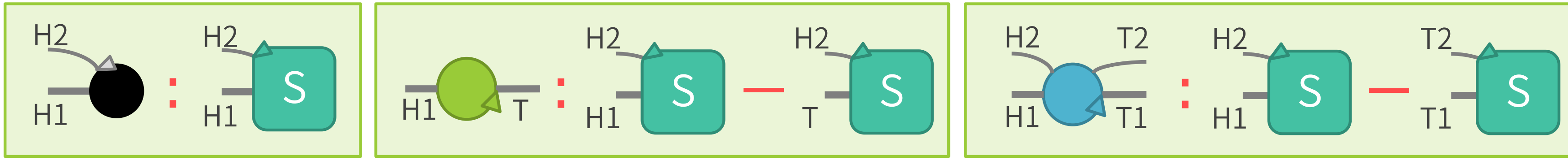
Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules



Type derivation with disjoint LMNtalGG

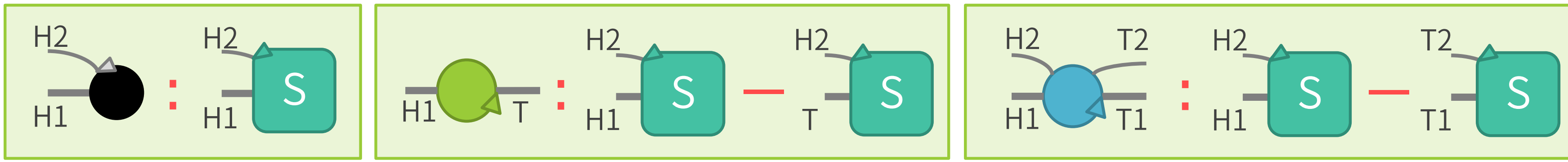
1. Obtain typings of all terminal symbols from production rules



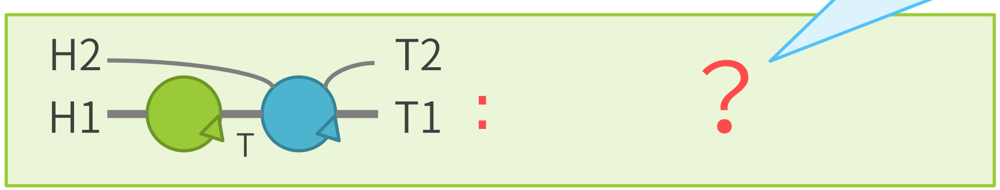
We use these typings as **axioms** of typing

Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules

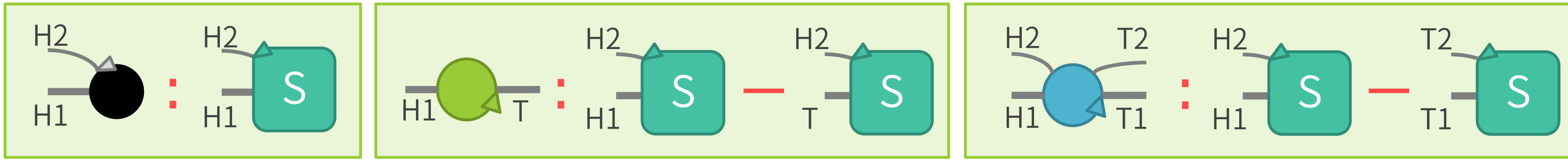


How can we get the type here?

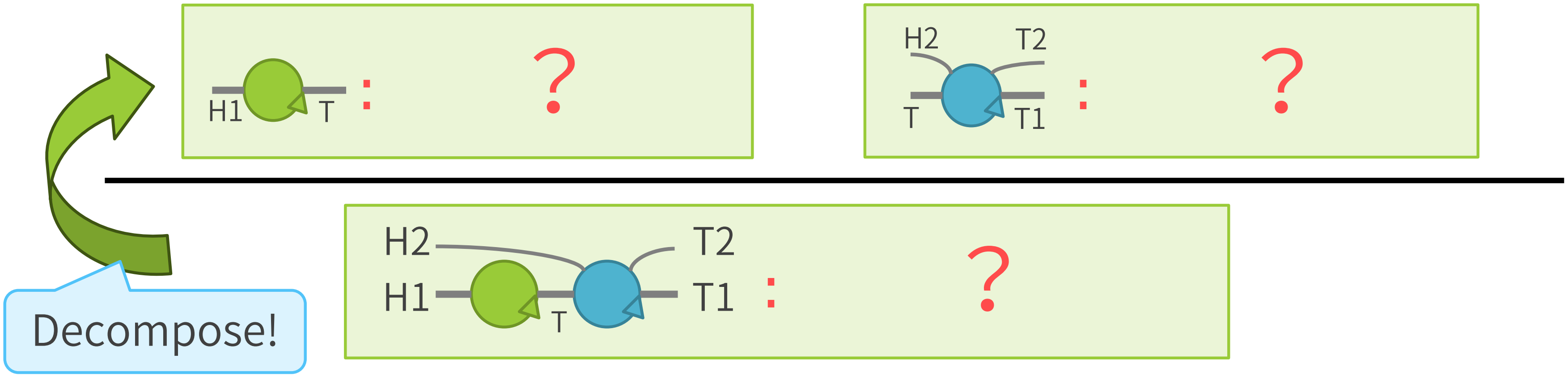


Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules

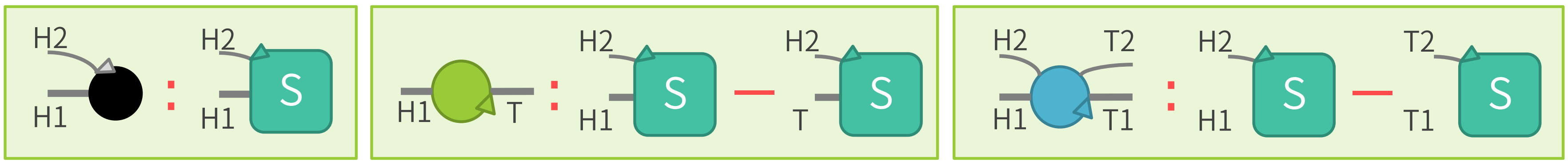


2. Construct the type of graph from subgraphs' typings

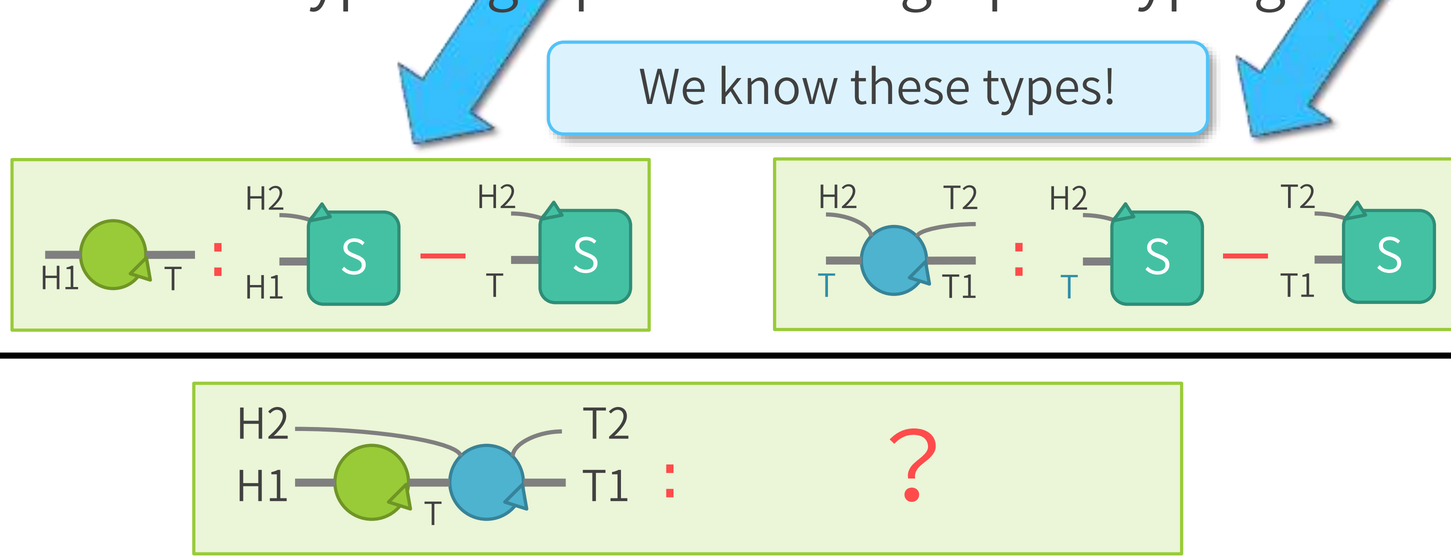


Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules



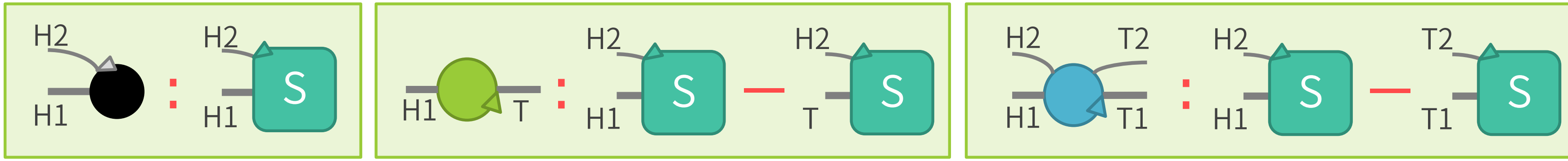
2. Construct the type of graph from subgraphs' typings



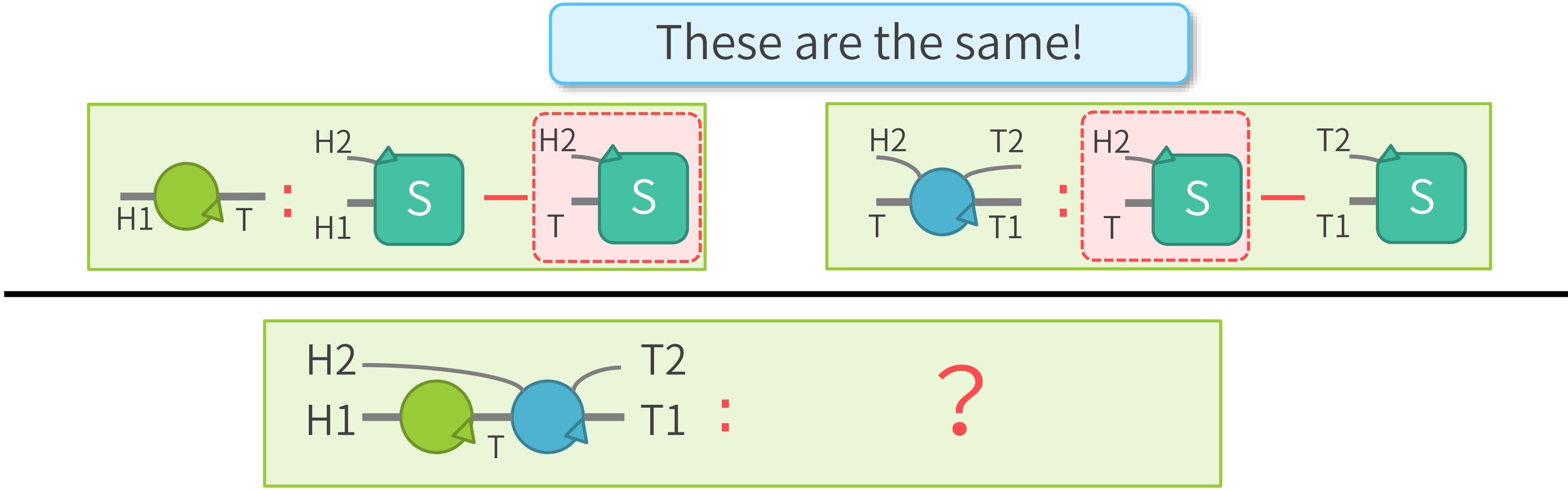
* We rename link names as needed（α-conversion, Prop. 3.5)

Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules

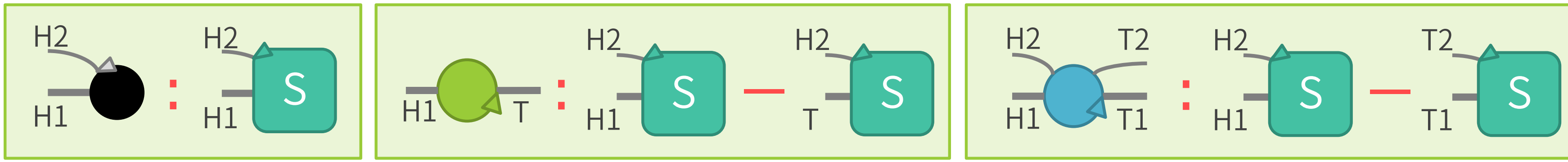


2. Construct the type of graph from subgraphs' typings

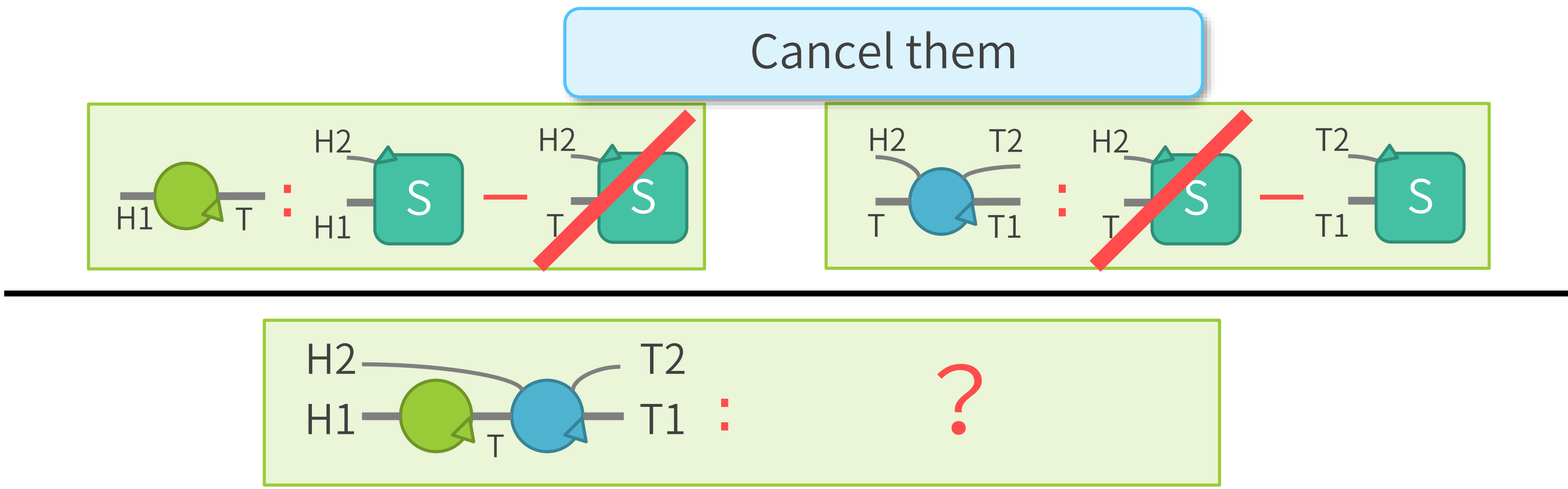


Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules

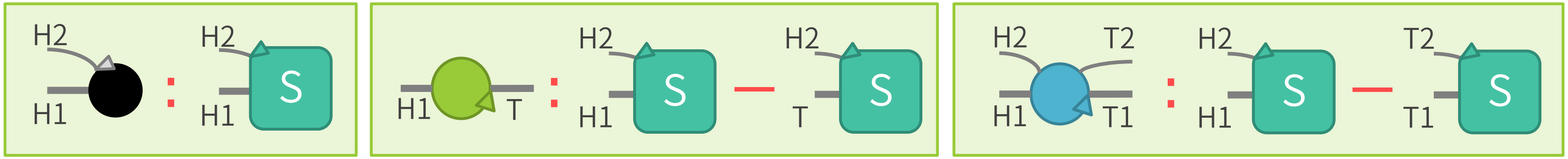


2. Construct the type of graph from subgraphs' typings

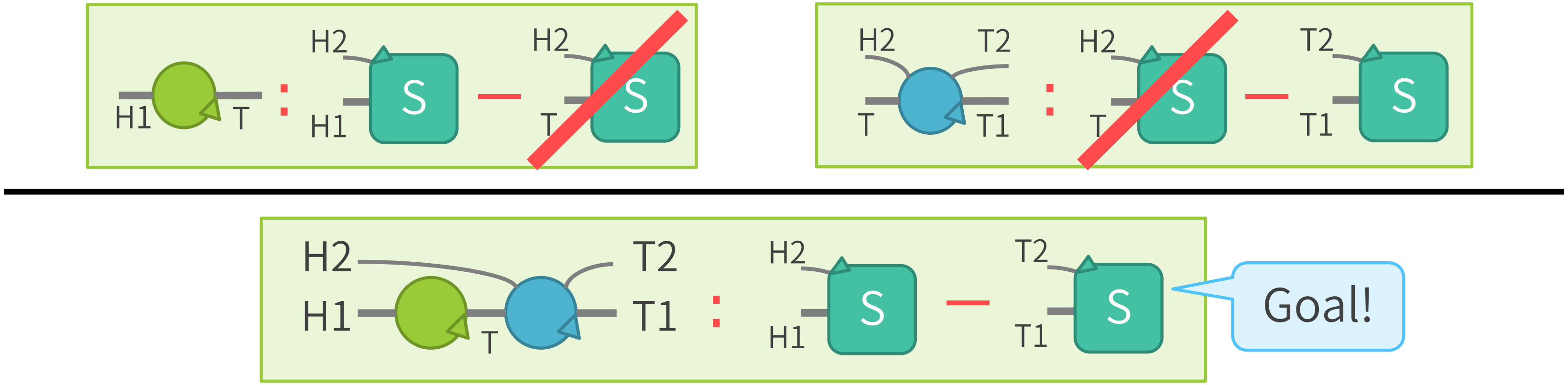


Type derivation with disjoint LMNtalGG

1. Obtain typings of all terminal symbols from production rules



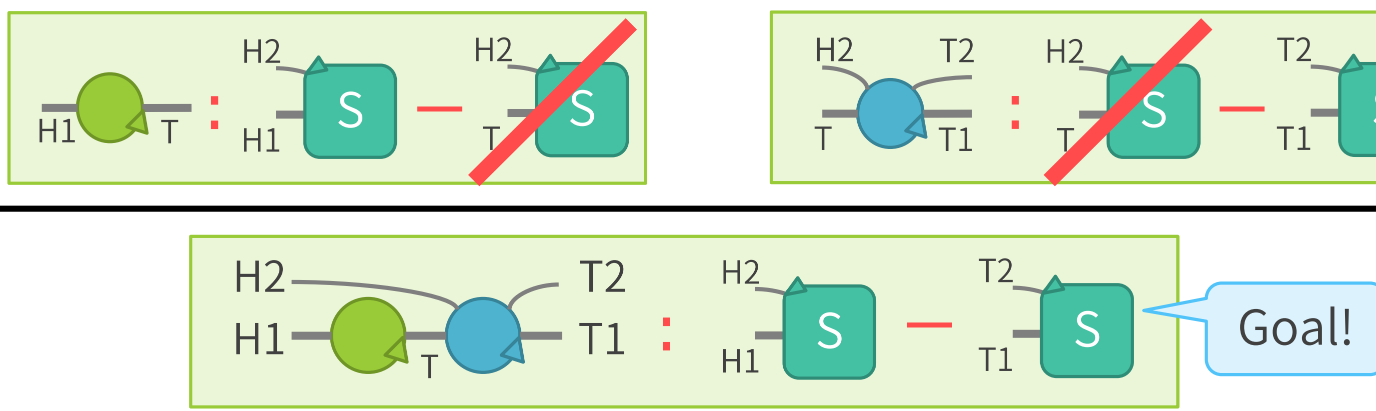
2. Construct the type of graph from subgraphs' typings



Type derivation: remarks

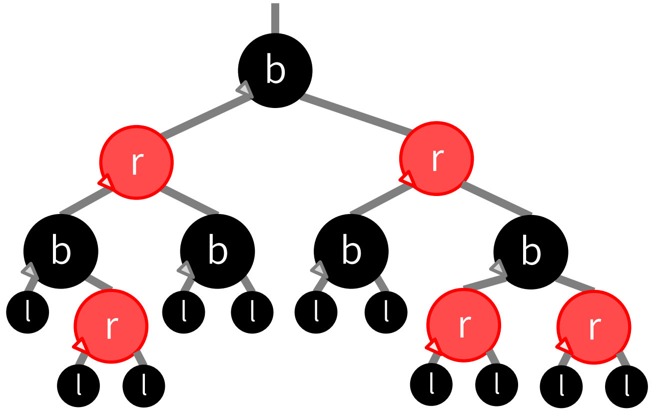
- Costs linear time w.r.t. # of atoms
  - For non-disjoint LMNtalGGs, you can still typecheck rules but at a cost
- Similar to the **cut rule** of the **Sequent Calculus**

difference types ↔ **sequents**,
type composition ↔ **cut rule**



Broader Class: Indexed LMNtalGG

- Non-terminal symbols can have integer indices
  - Inspired by indexed grammars †
  - Can handle shapes with numeric constraints (e.g., balanced red-black trees)



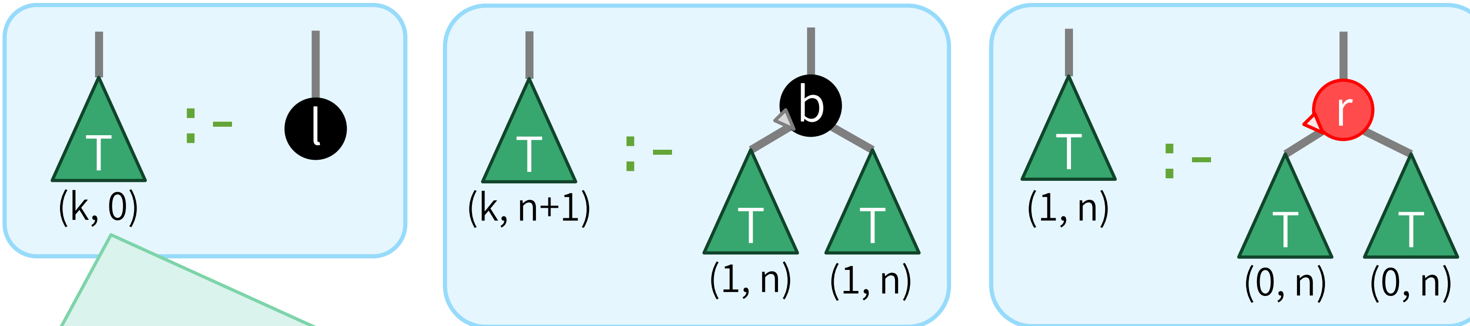
Requirements for red-black trees

1. The root and leaves are b
2. r 's children are b
3. # of b on the path from the root to a leaf (black height) is a constant

numeric constraint

† A. V. Aho: Indexed Grammars—An Extension of Context-Free Grammars, J. ACM, 15(4), 1968.

Red-black Trees with Indexed LMNtalGG



Indices: { Color of the root (0: black, 1: red or black)
Black height

Requirements for red-black trees

1. The root and leaves are b
2. r 's children are b
3. # of b on the path from the root to a leaf (black height) is a constant

This grammar can be considered disjoint (when indices are ignored)

Outline

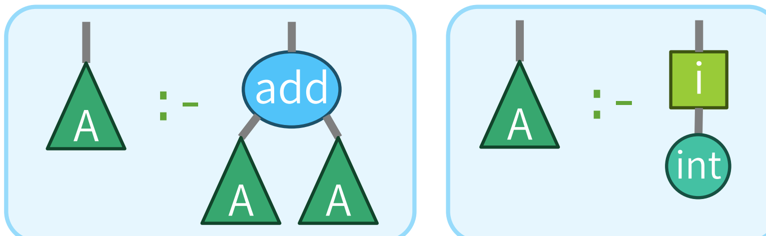
1. Target Language: LMNtal
2. LMNtalGG and Difference Types
3. Classifying LMNtalGGs: Disjoint & Indexed
4. Applications
   - (dynamic) Pattern Matching
   - (static) Type Checking
5. Type Checking of Functional Atoms

Application to pattern matching

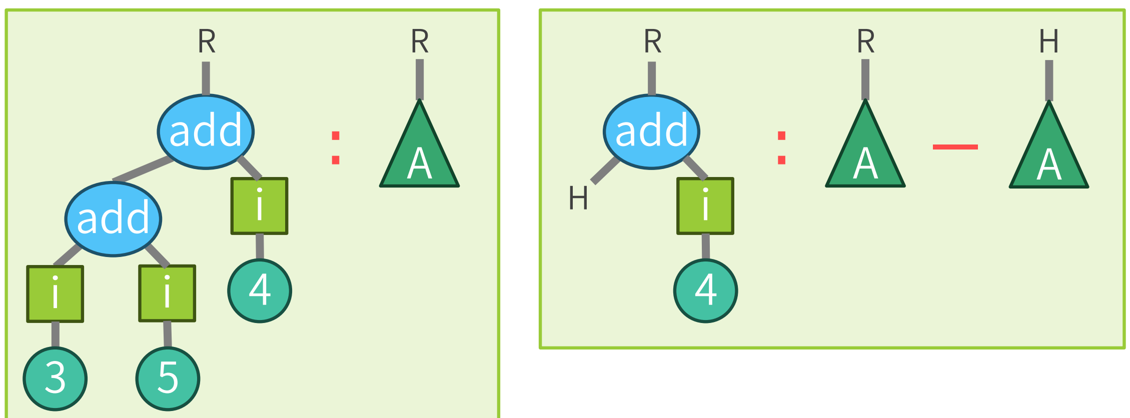
Disjoint LMNtalGG supports tree-shaped (difference) structures

Example: binary trees consisting of add nodes (addtree)

Grammar



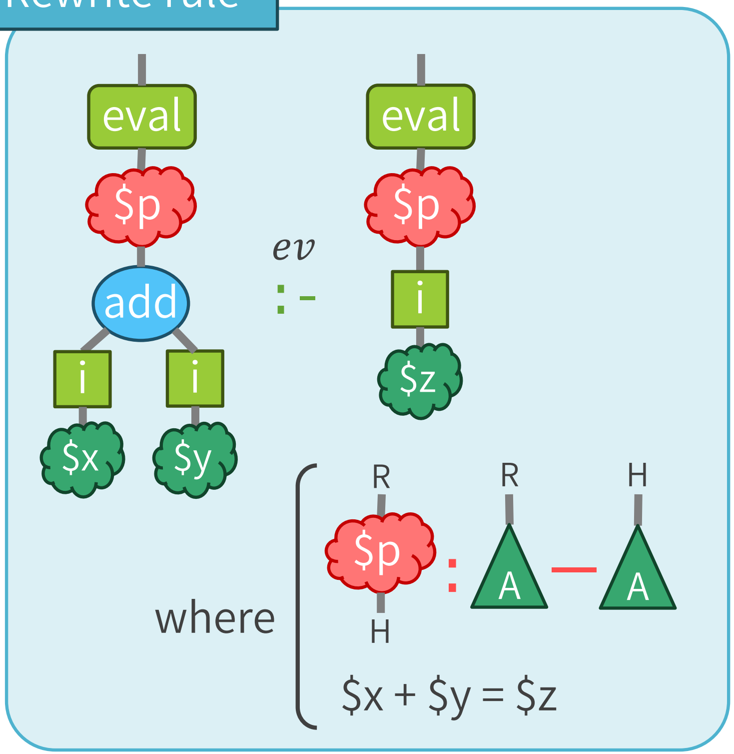
Example Typings



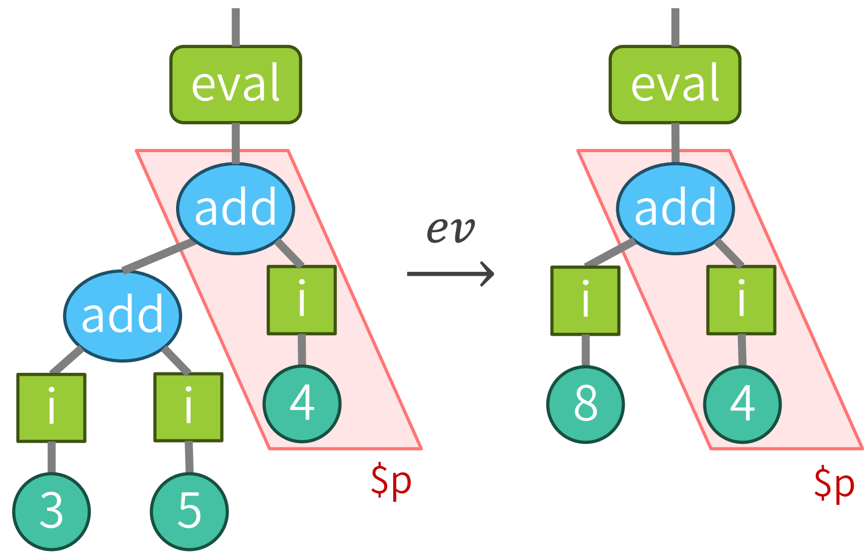
Application to pattern matching

We can describe pattern matching on DDSs with LMNtalGGs

Rewrite rule



Example of application

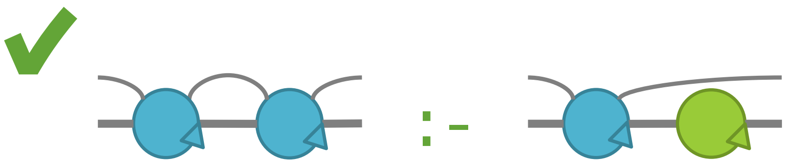


This simulates evaluation contexts

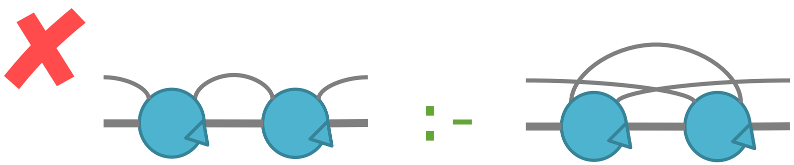
Application to rule type checking

Checks if the application of a rule **preserves** types of graphs

$$\forall G, G'.\quad G : \tau \;\wedge\; G \xrightarrow{R} G' \implies G' : \tau$$



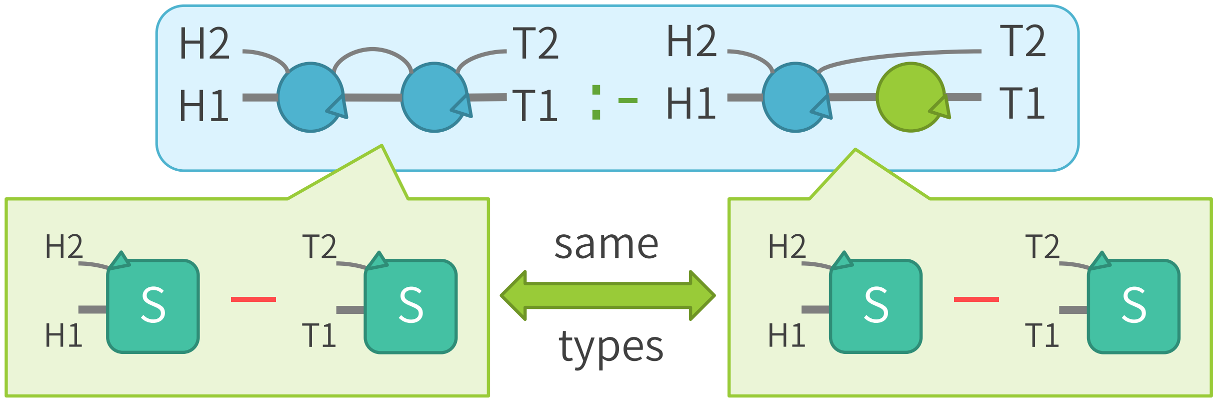
Applying this to a skip list always results in a skip list



Applying this to a skip list does not result in a skip list

Application to rule type checking

To confirm that a given rule **preserves** types of graphs,



check if the LHS and the RHS are of the same type

- simply perform type derivation for both sides
- Intuition: The type of the whole graph will not change because it just rewrites a difference skip list to a difference skip list

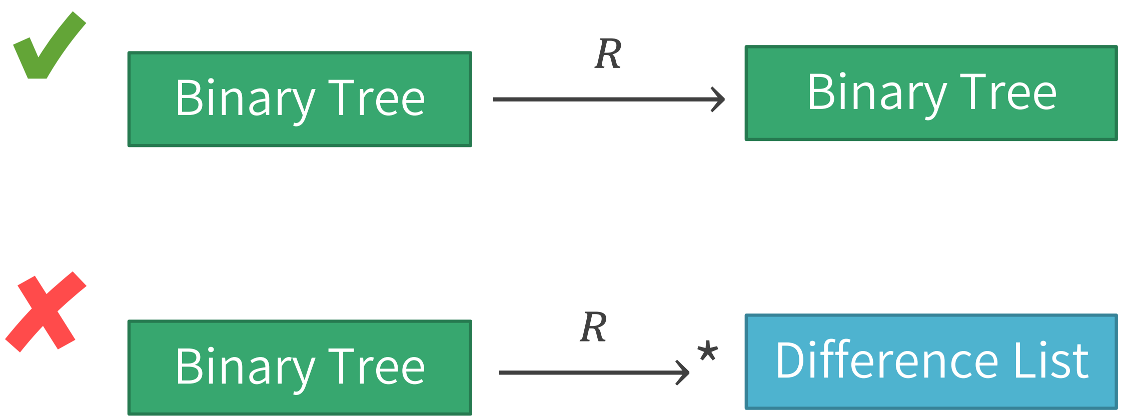
Outline

1. Target Language: LMNtal
2. LMNtalGG and Difference Types
3. Classifying LMNtalGGs: Disjoint & Indexed
4. Applications: Pattern Matching & Static Type Checking
5. Type Checking of Functional Atoms

Multi-step/shape-changing operations

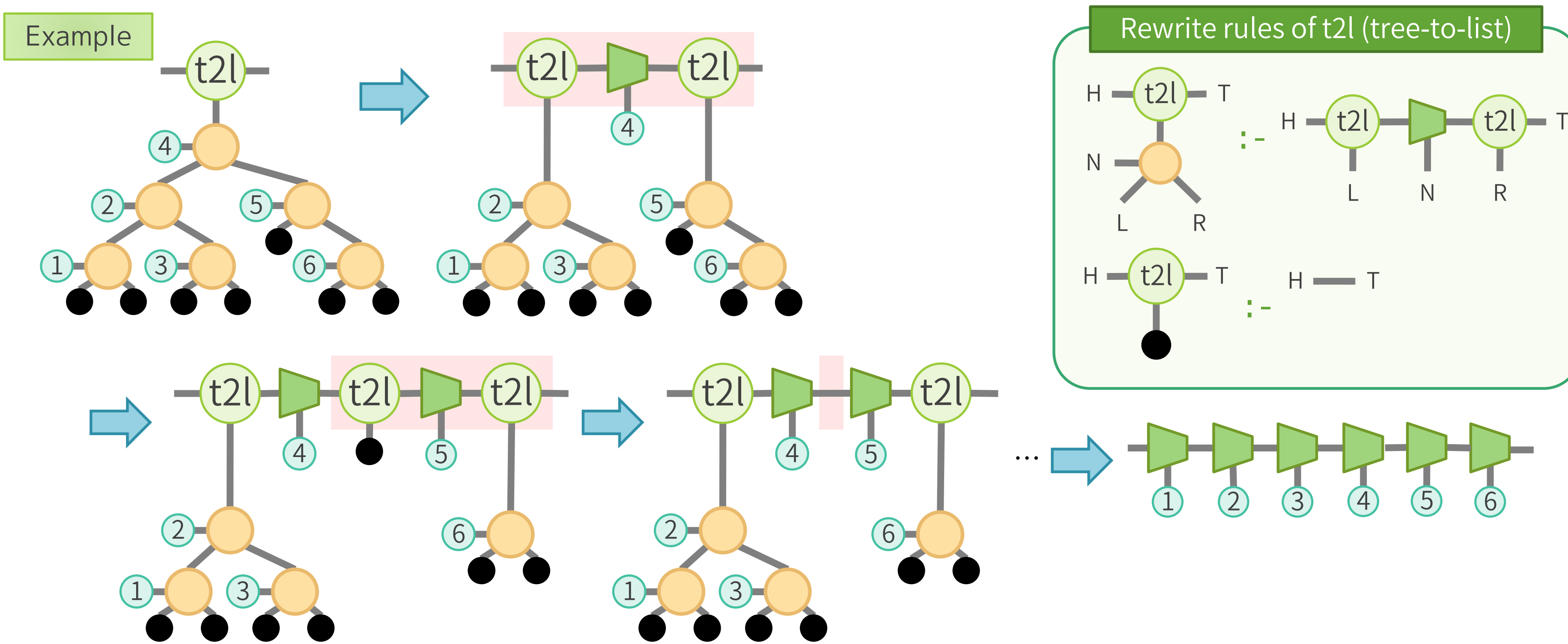
- In most of the existing typing frameworks for graphs,
  - Type Safety: "Rewrite rules will **never destroy** the shape of graphs"

    → Operations that may result in other types were out of scope



Functional Atoms: Example

- Graph nodes that behave like functions (in functional languages)
- Common design pattern (LMNtal has **no functions a-priori**)



Expected Property of Functional Atoms

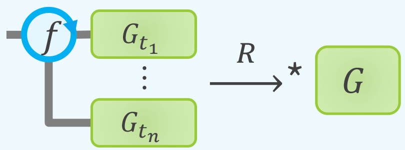
Binary Tree $\xrightarrow{R}{}^{*}$ Difference List

- We expect t2l satisfies …
  - If it receives a binary tree, it eventually returns a difference list
- In general, this property can be formalized as:

f is **a functional atom** that takes types $t_1, \dots, t_n$ and returns type T

$\overset{\text{def}}{\Longleftrightarrow}$

For any graphs $G_{t_1}, \dots, G_{t_n}$ having types $t_1, \dots, t_n$ (resp.),
if $(f, G_{t_1}, \dots, G_{t_n})$ can be reduced to G
that includes no f atoms, then G has the type T.



We write this property as $F : t_1 ; \cdots ; t_n \rightarrowtail T$

e.g., `t2l(P,T,H) : tree(P)` $\rightarrowtail$ `list(H)−list(T)`

Checking Functional Atoms with LMNtalGG

- If the input types include **no** differences (i.e., of the form $\tau - \mathbf{0}$),

  To check that the atom F has the functional property,

  $$F\colon \boxed{\tau_1; \dots; \tau_n} \rightarrowtail \alpha - \beta$$

  $\tau_1, \dots, \tau_n$ are **inputs**

  we assume the following typing

  $$F\colon\ \alpha - (\beta, \boxed{\tau_1, \dots, \tau_n})$$

  $\tau_1, \dots, \tau_n$ are **holes**

  and confirm that the rules preserve types

- For details (esp., correctness), see our previous work†

† N. Yamamoto et al.: Engineering Grammar-based Type Checking for Graph Rewriting Languages. IEEE ACCESS, 10, 2022.

Related Work

- **Graph Types**[†1]: Based on regular expressions
- **Structured Gamma**[†2] and **Shape Types**[†3]
  - Based on context-free graph grammars

a subset that satisfies completeness

- **Refinement Types**[†4]: Types with numeric constraints
  - Implemented on Liquid Haskell with type inference[†5]
- **Typed Prolog**[†6]: Difference lists are typable
  - Types (e.g., list, int) with Modes (direction)

†1 P. Fradet et al.: Structured Gamma, Science of Computer Programming, 31(2), 1998.
†2 N. Klarlund et al.: Graph Types, Proc. POPL'93.
†3 P. Fradet et al.: Shape types, Proc. POPL'97.
†4 N. Vazou et al.: Refinement types for Haskell, SIGPLAN Not., 49(9), 2014.
†5 P.M. Rondon et al.: Liquid types, SIGPLAN Not., 43(6), 2008.
†6 T.K. Lakshman et al.: Typed Prolog: A Semantic Reconstruction of the Mycroft-O'Keefe Type System, Proc. ICLP'90.

Conclusion

1. Proposed
   i. **LMNtalGG** as graph grammar on LMNtal
   ii. **Difference Types** on LMNtalGG to deal with DDSs
2. Introduced two major applications of LMNtalGG:
   i. **Pattern Matching** on DDSs
   ii. **Static Type Checking** of rewrite rules
3. Introduced Functional Atoms
   to handle **multi-step and/or shape-changing operations**

Future Work

1. Full implementation of type checking
   - with indices and functional atoms
2. Expanding the target language
   - Membranes (boxes), hyperedges (HyperLMNtal)
3. Non-terminating programs and infinite structures
4. Index types with indices richer than integers