# Automata and Logics Over Nested Data


*Adriana Baldacchino*

*Supervised by Prof. Andrzej Murawski*


Women in Logic, 14th July 2025

# Automata over a finite alphabet

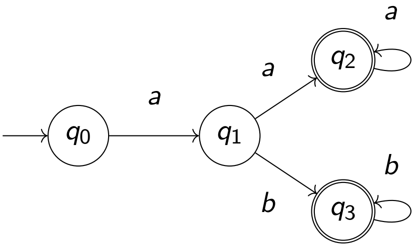
$a_i \in \Sigma$, where $\Sigma$ is finite.

$$a_1 a_2 \dots a_k \longrightarrow \boxed{\begin{array}{c}\text{Automaton}\\ \mathcal{A}\end{array}} \longrightarrow \text{accept/reject}$$

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A} \text{ accepts } w\}$.

# Transitions: Automata over a finite alphabet

Each $a \in \Sigma$ determines the next move



# Data Automata

- $a_i \in \Sigma$, where $\Sigma$ is a finite set of *tags*;
- $d_i \in D$, where $D$ is an infinite set of *data values*.

$$(a_1, d_1)(a_2, d_2) \ldots (a_k, d_k) \longrightarrow \boxed{\begin{array}{c}\text{Data Automaton}\\ \mathcal{A}\end{array}} \longrightarrow \quad \text{accept/reject}$$

# Data Automata

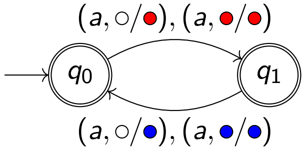
- $a_i \in \Sigma$, where $\Sigma$ is a finite set of *tags*;
- $d_i \in D$, where $D$ is an infinite set of *data values*.

$$(a_1, d_1)(a_2, d_2)\ldots(a_k, d_k) \longrightarrow \boxed{\begin{array}{c}\text{Data Automaton}\\ \mathcal{A}\end{array}} \longrightarrow \quad \text{accept/reject}$$

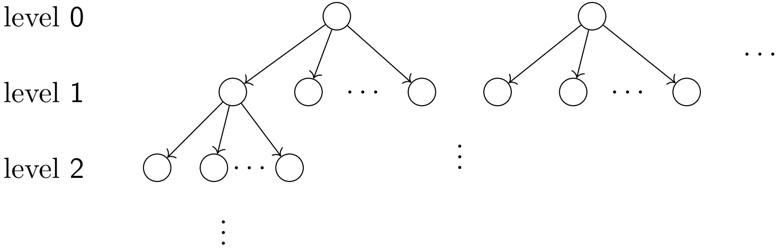
We can't simply specify a transition for each $(a, d) \in \Sigma \times D$ as this set is infinite.

# Transitions: Class Memory Automata

- One way to handle infinite data is to use a finite set of colours (labels) $\mathcal{M} = \{$🔴, 🔵, 🟡$\}$.
- Each data value starts off as unlabelled (○), and is given a colour when encountered.
- These models are called (weak) Class Memory Automata.



# Nested Data

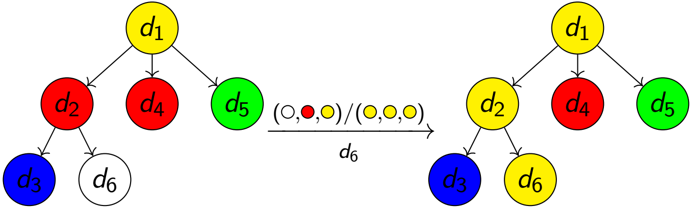


# Transitions for Automata over Nested Data

The most straightforward way to define transitions on nested data is to specify the labels of all the ancestors of a data value. For example, to transition on a level 2 data value, we need to specify its label, the label of its parent and its parent's parent.

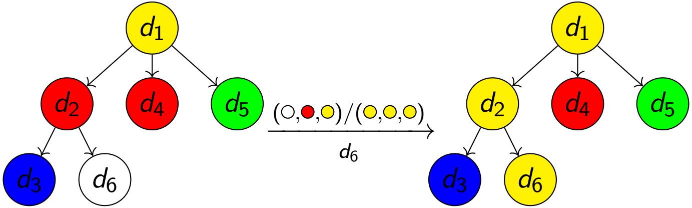
# Transitions for Automata over Nested Data

The most straightforward way to define transitions on nested data is to specify the labels of all the ancestors of a data value. For example, to transition on a level 2 data value, we need to specify its label, the label of its parent and its parent's parent.



# Transitions for Automata over Nested Data

The most straightforward way to define transitions on nested data is to specify the labels of all the ancestors of a data value. For example, to transition on a level 2 data value, we need to specify its label, the label of its parent and its parent's parent.




These automata are called *Nested-Data Class Memory Automata (NDCMA)*, and were introduced in [2].

# Limitations of NDCMA

- Since the number of transitions is *finite* and NDCMA require us to label *all ancestors*, this formulation clearly limits us to process data values up to some bounded level/depth $k$.

# Limitations of NDCMA

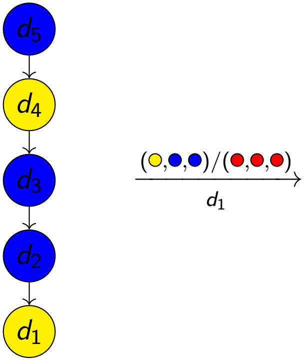
- Since the number of transitions is *finite* and NDCMA require us to label *all ancestors*, this formulation clearly limits us to process data values up to some bounded level/depth $k$.
- As they have a decidable emptiness problem and nice closure properties, deterministic NDCMA were successfully applied to solve some decidability problems for fragments of ML [3, 1].

# Limitations of NDCMA

- Since the number of transitions is *finite* and NDCMA require us to label *all ancestors*, this formulation clearly limits us to process data values up to some bounded level/depth $k$.
- As they have a decidable emptiness problem and nice closure properties, deterministic NDCMA were successfully applied to solve some decidability problems for fragments of ML [3, 1].
- Our goal is to extend this model to handle unbounded depth data values in a way that preserves these properties. We hope that this can help us extend the previous work on programming languages to handle programs with nested loops.

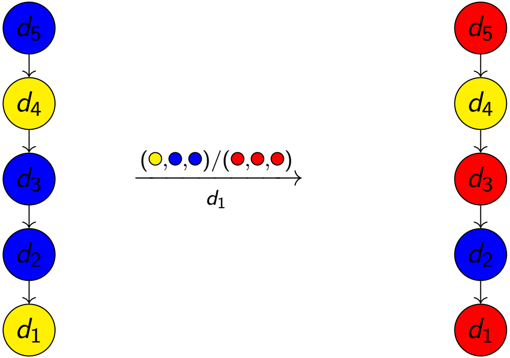
# Memoryless Unbounded NDCMA

One way we can handle unbounded data is by matching our labelling, say (🟡, 🔵, 🔵) to a *sublabelling* of the ancestors of $d$:



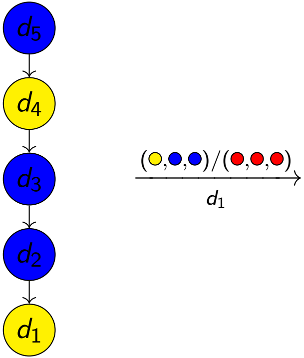
# Memoryless Unbounded NDCMA

One way we can handle unbounded data is by matching our labelling, say (●, ●, ●) to a *sublabelling* of the ancestors of $d$:



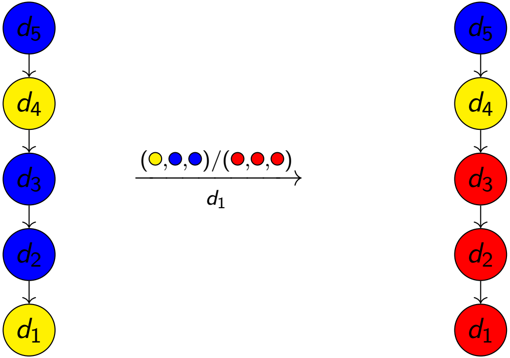
# Memoryless Unbounded NDCMA

One way we can handle unbounded data is by matching our labelling, say (●, ●, ●) to a *sublabelling* of the ancestors of $d$:



# Memoryless Unbounded NDCMA

One way we can handle unbounded data is by matching our labelling, say (🟡, 🔵, 🔵) to a *sublabelling* of the ancestors of $d$:

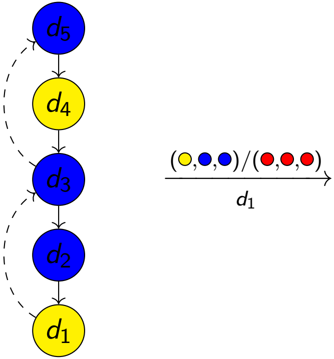


# $k$-Memory Unbounded NDCMA

A more controlled way is to maintain a bounded memory $m(d)$ for each data value.

$m(d_3) = (d_5)$

$m(d_1) = (d_3, d_5)$

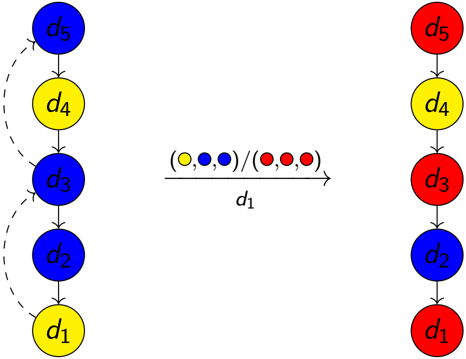


# $k$-Memory Unbounded NDCMA

A more controlled way is to maintain a bounded memory $m(d)$ for each data value.

$m(d_3) = (d_5)$

$m(d_1) = (d_3, d_5)$



# Simulating Loops Example

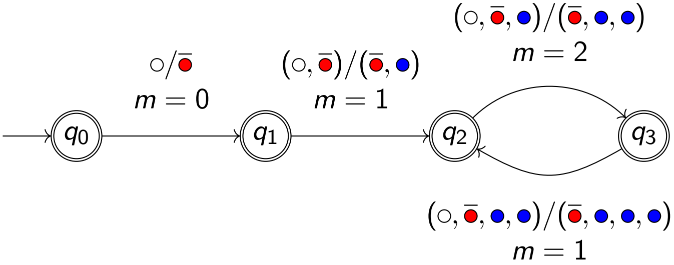
Suppose we had a very simple program,

$$d_0; \textit{while} \text{ true } \textit{do } d_1; d_2$$

and we want to accept all execution traces of this program.

# Simulating Loops Example

We simulate this using an automaton with labels $\mathcal{M} = \{{\color{blue}\bullet}, {\color{red}\overline{\bullet}}\}$, where ${\color{red}\overline{\bullet}}$ represents the 'head' of the program.



# Simulating Loops Example

- It is clear that the words accepted by this $k$-memory NDCMA are of the form $d_1 d_2 d_3 \ldots d_k$ where $d_i$ is the parent of $d_{i+1}$ in the underlying structure.

# Simulating Loops Example

- It is clear that the words accepted by this $k$-memory NDCMA are of the form $d_1 d_2 d_3 \dots d_k$ where $d_i$ is the parent of $d_{i+1}$ in the underlying structure.
- Even though the correct nesting properties are preserved, the language of this automaton is equivalent to the one that would be generated by say *while* true *do* $d_1$.

# Simulating Loops Example

- It is clear that the words accepted by this $k$-memory NDCMA are of the form $d_1 d_2 d_3 \ldots d_k$ where $d_i$ is the parent of $d_{i+1}$ in the underlying structure.
- Even though the correct nesting properties are preserved, the language of this automaton is equivalent to the one that would be generated by say *while* true *do* $d_1$.
- This is to say the loops cannot be deduced from the language itself, allowing us to compare differently shaped programs with the same underlying traces.

# Closure Properties

- Using the theory of well-structured transition systems (WSTS) we showed that the language emptiness problem is decidable for both these models.

# Closure Properties

- Using the theory of well-structured transition systems (WSTS) we showed that the language emptiness problem is decidable for both these models.
- We further present closure and decidability properties for these languages.

| | $L_1 \cup L_2$ | $L_1 \cap L_2$ | $L^c$ | $L_1 \subseteq L_2$ | $L_1 = L_2$ |
|---|---|---|---|---|---|
| Memoryless NDCMA | ✓ | ✓ | X | X | X |
| $k$-memory NDCMA | ✓ | X | X | X | X |
| Det. $k$-memory NDCMA | X | X | ✓ | X | ? |
| Det. Bounded NDCMA | ✓ | ✓ | ✓ | ✓ | ✓ |

# Further Considerations

- The question of whether equivalence is decidable for deterministic $k$-memory NDCMA is still open. It is still possibly decidable, as there exist automata with decidable equivalence/undecidable inclusion — DPDA.
- Logics for data words with unbounded data have not yet been investigated thoroughly. It would be interesting to see an analogous result as in [4] — where satisfiability of a logic over data words (NDLTL) was shown using an equivalent model to bounded NDCMA.
- We hope to also extend the work in [1] to find an analogous translation of FOSC with loops to $k$-memory NDCMA.

Thank you!

# References I


[1] Benedict Bunting and Andrzej Murawski. "Contextual Equivalence for State and Control via Nested Data". In: *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '24. Tallinn, Estonia: Association for Computing Machinery, 2024. ISBN: 9798400706608. DOI: 10.1145/3661814.3662109.

[2] Conrad Cotton-Barratt, Andrzej Murawski, and C.-H. Luke Ong. "Weak and Nested Class Memory Automata". In: *Language and Automata Theory and Applications*. Ed. by Adrian-Horia Dediu et al. Cham: Springer International Publishing, 2015, pp. 188–199. ISBN: 978-3-319-15579-1.

[3] Conrad Cotton-Barratt et al. "Fragments of ML decidable by nested data class memory automata". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2015, pp. 249–263.

# References II


[4] Normann Decker et al. "Ordered Navigation on Multi-attributed Data Words". In: *CONCUR 2014 – Concurrency Theory*. Ed. by Paolo Baldan and Daniele Gorla. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 497–511. ISBN: 978-3-662-44584-6.