

Logics on words and trees with data

Ranko Lazić & Diego Figueira
U. Warwick CNRS, LaBRI

ESSLLI 2016 - Bolzano

words with infinite alphabets
trees & finite

data words

a b a b a b b b a b b
2 4 7 6 9 3 9 7 6 2 2 9

(a data word)

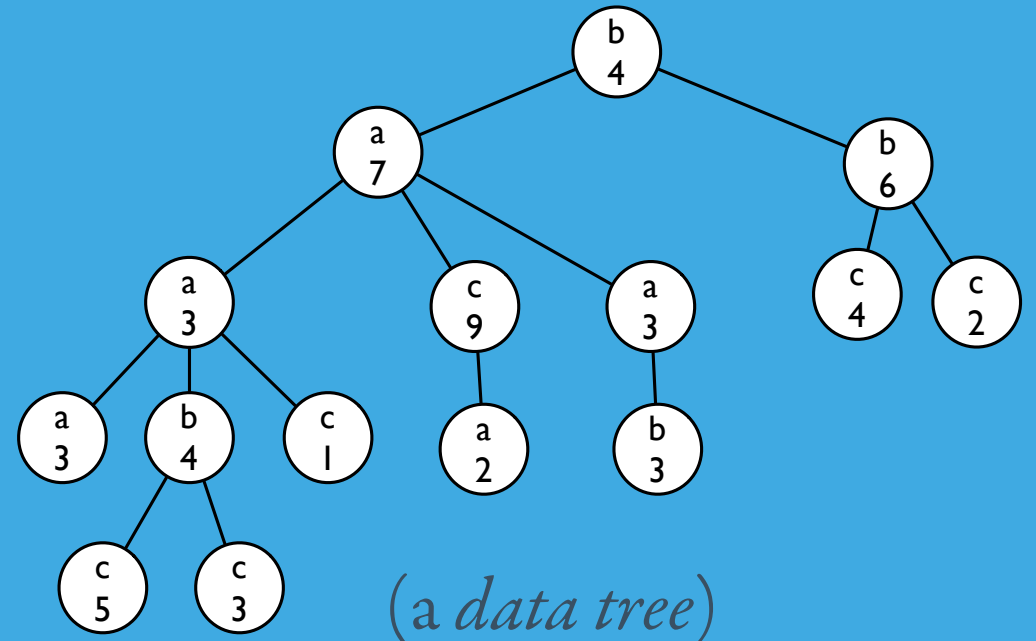
words with infinite alphabets
trees & finite

data words

a b a b a b b b b a b b
2 4 7 6 9 3 9 7 6 2 2 9

(a data word)

data trees



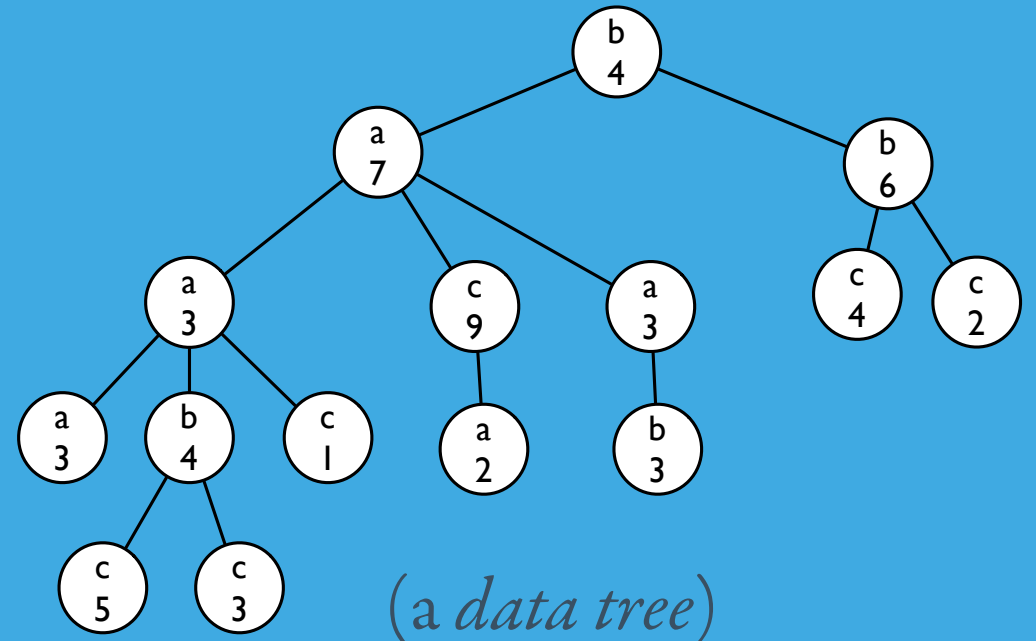
words with infinite alphabets
trees & finite

data words

a b a b a b b b b a b b
2 4 7 6 9 3 9 7 6 2 2 9

(a data word)

data trees



questions

How to reason about these structures?

What properties are hard/easy?

Decidability bounds?

Connections with other areas?



Agenda

- Monday (Diego, Ranko): Introduction, data words
- Tuesday (Ranko): Data words, first-order logic
- Wednesday (Ranko): Data words, temporal logics
- Thursday (Diego): Data trees, path-based logics
- Friday (Diego): Data trees, other formalisms

PLEASE

ASK

(easy)

QUESTIONS

data words

data words

data
word $a \ b \ a \ b \ a \ b \ b \ b \ b \ a \ b \ b \in (\{a, b\} \times \mathbb{N})^*$
 2 4 7 6 9 3 9 7 6 2 2 9

data words

*data
word* $\begin{matrix} a & b & a & b & a & b & b & b & b & a & b & b \\ 2 & 4 & 7 & 6 & 9 & 3 & 9 & 7 & 6 & 2 & 2 & 9 \end{matrix} \in (\{a, b\} \times \mathbb{N})^*$

“for every a there is a b with same data value to its right”

data words


data word $\begin{matrix} a & b & a & b & a & b & b & b & b & a & b & b \\ 2 & 4 & 7 & 6 & 9 & 3 & 9 & 7 & 6 & 2 & 2 & 9 \end{matrix} \in (\{a, b\} \times \mathbb{N})^*$

“for every a there is a b with same data value to its right”

What does it represent?

data words

data word $\begin{matrix} a & b & a & b & a & b & b & b & b & a & b & b \\ 2 & 4 & 7 & 6 & 9 & 3 & 9 & 7 & 6 & 2 & 2 & 9 \end{matrix} \in (\{a, b\} \times \mathbb{N})^*$



“for every a there is a b with same data value to its right”

execution of concurrent processes,

\mathcal{W} what does it represent?

data words

<i>data</i>	$r(A)$	$r(B)$	$w(A)$	$w(B)$	<i>commit</i>	$r(A)$	$w(A)$
<i>word</i>	4	7	4	7	4	7	7

execution of concurrent processes,

W *hat does it represent?*

data words

<i>data</i>	$r(A)$	$r(B)$	$w(A)$	$w(B)$	<i>commit</i>	$r(A)$	$w(A)$
<i>word</i>	4	7	4	7	4	7	7

“between $w(A)$ and commit of process P there are no $w(A)$ from any other process P ”

execution of concurrent processes,

W

hat does it represent?

data words

<i>data word</i>	$r(A)$ 4	$r(B)$ 7	$w(A)$ 4	$w(B)$ 7	<i>commit</i> 4	$r(A)$ 7	$w(A)$ 7
----------------------	-------------	-------------	-------------	-------------	--------------------	-------------	-------------

“between $w(A)$ and commit of process P there are no $w(A)$ from any other process P ”

What does it represent?

*execution of concurrent processes,
usage of some unbounded resources,
timed words,
temporal databases,
runs of counter automata (or inf. state aut.),
...*

Reasoning on data-words

Given a logic \mathcal{L} on data-words,

Satisfiability problem

Input: $\phi \in \mathcal{L}$

Output: is there a data-word w so that $w \models \phi$?

Implication problem

Input: $\phi, \psi \in \mathcal{L}$

Output: is it true that $w \models \phi$ implies $w \models \psi$ for every data word w ?

Reasoning on data-words

Given a logic \mathcal{L} on data-words,

Satisfiability problem

Input: $\phi \in \mathcal{L}$

Output: is there a data-word w so that $w \models \phi$?

Implication problem

Input: $\phi, \psi \in \mathcal{L}$

Output: is it true that $w \models \phi$ implies $w \models \psi$ for every data word w ?

If \mathcal{L} closed under boolean operators:

- $\text{implication}(\phi, \psi) \equiv \neg \text{sat}(\phi \wedge \neg \psi)$
- $\text{sat}(\phi) \equiv \neg \text{implication}(\phi, \perp)$

Why *reasoning*?

Why *reasoning*?

- It's fun! 😊

Why *reasoning*?

- It's fun! 😊
- Basic question for understanding a formalism: Does this mean anything at all? Is this a property?

Why *reasoning*?

- It's fun! 😊
- Basic question for understanding a formalism: Does this mean anything at all? Is this a property?
- Query optimisation:
 - If $Q \equiv Q'$ then it is "safe" to replace Q with a more efficient Q'
 - If Q is unsatisfiable (it contains a contradiction): its computation can be avoided

Why *reasoning*?

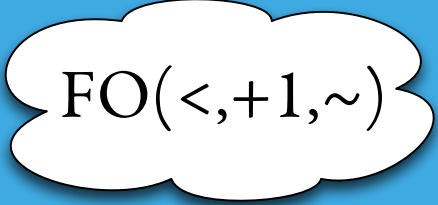
- It's fun! 😊
- Basic question for understanding a formalism: Does this mean anything at all? Is this a property?
- Query optimisation:
 - If $Q \equiv Q'$ then it is "safe" to replace Q with a more efficient Q'
 - If Q is unsatisfiable (it contains a contradiction): its computation can be avoided
- In general: verify statically whether a program satisfies a specification (eg, query accessing forbidden info)

Proviso

- We consider **closure under isomorphism** of data values (ie, only equality/inequality)
- We will mostly focus on **finite** structures
- We will mostly focus on **logics** (closed under boolean connectives)
- **Boolean** formulas (ie, 'properties' instead of 'queries')

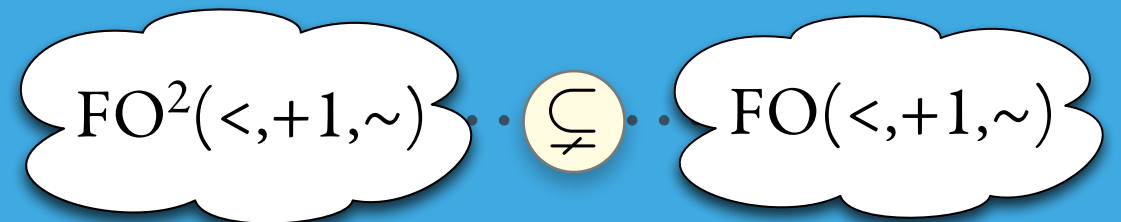
A zoo of formalisms on data words

A zoo of formalisms on data words

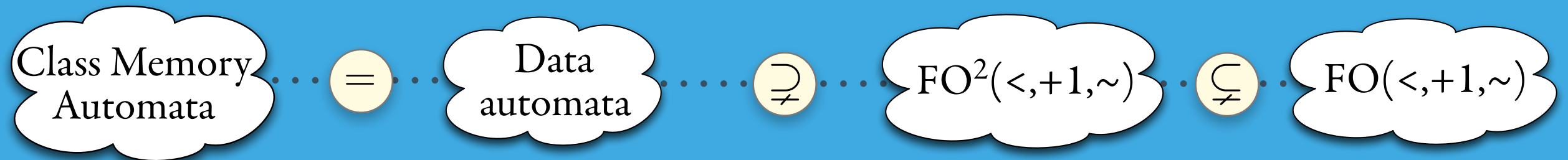


$\text{FO}(<, +1, \sim)$

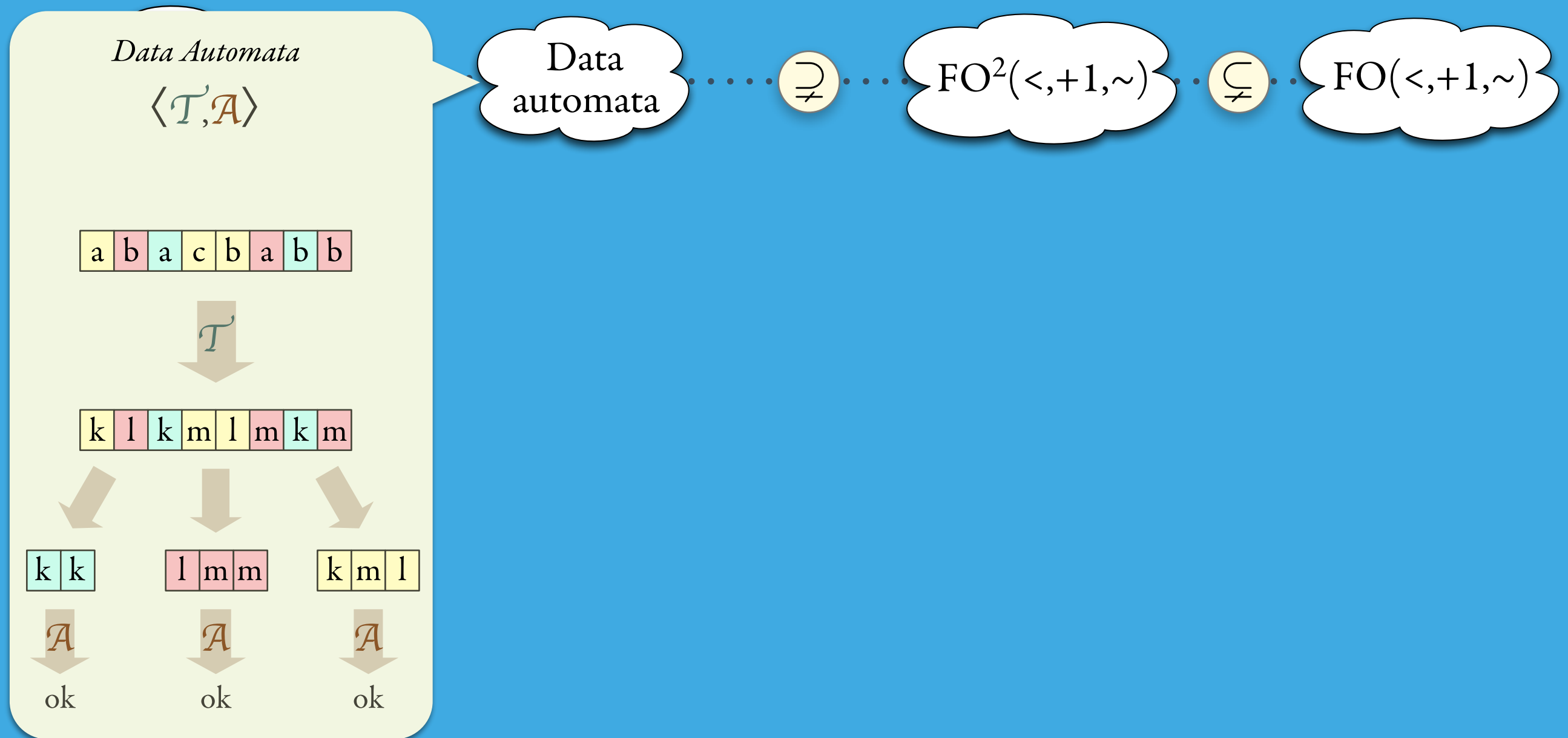
A zoo of formalisms on data words



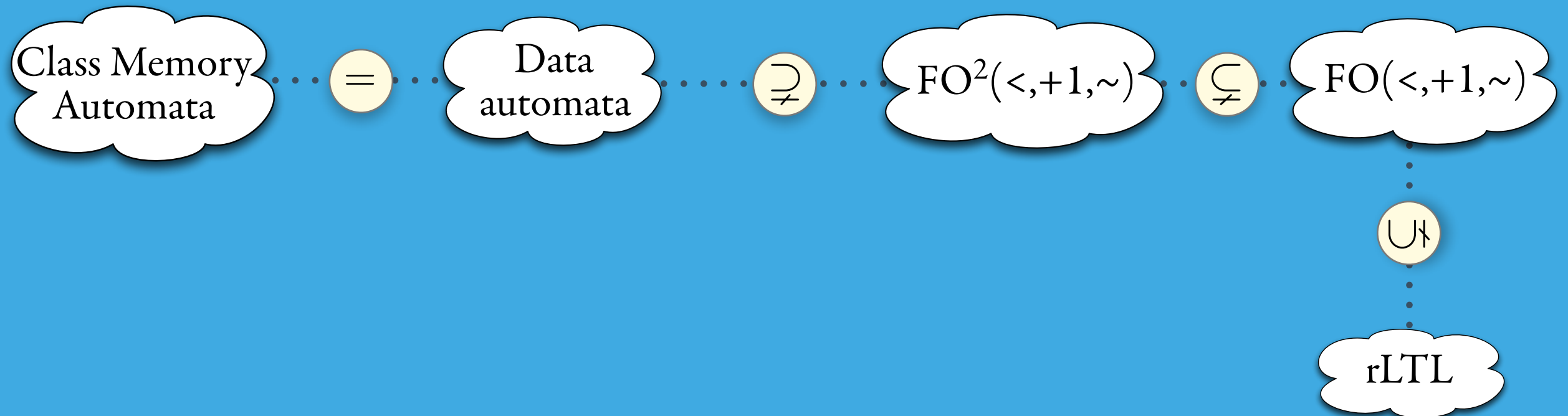
A zoo of formalisms on data words



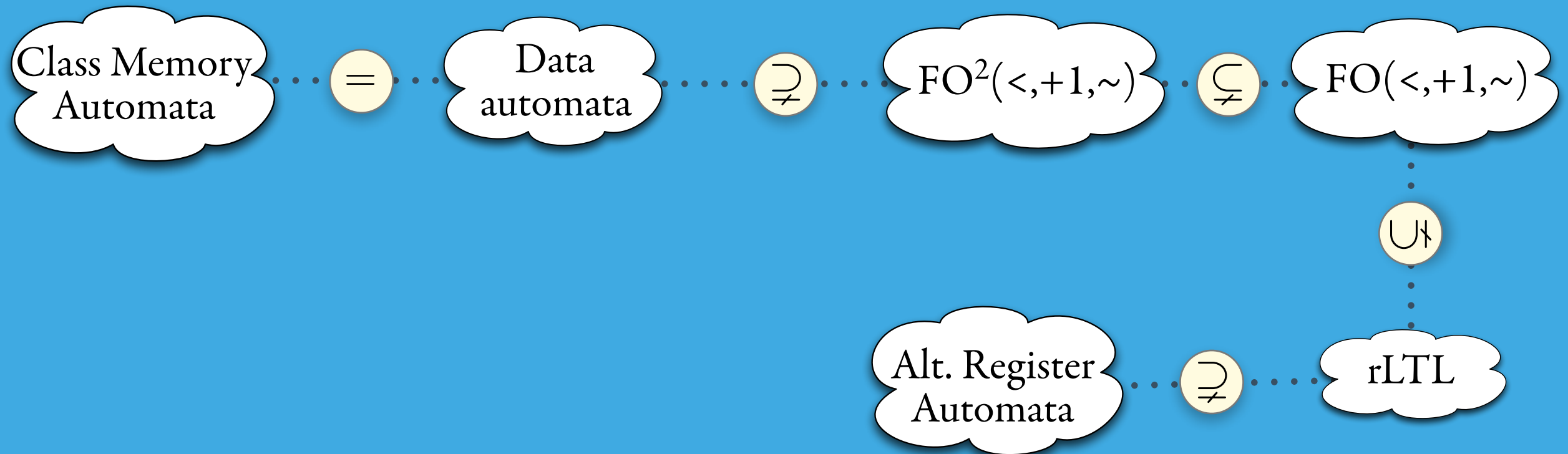
A zoo of formalisms on data words



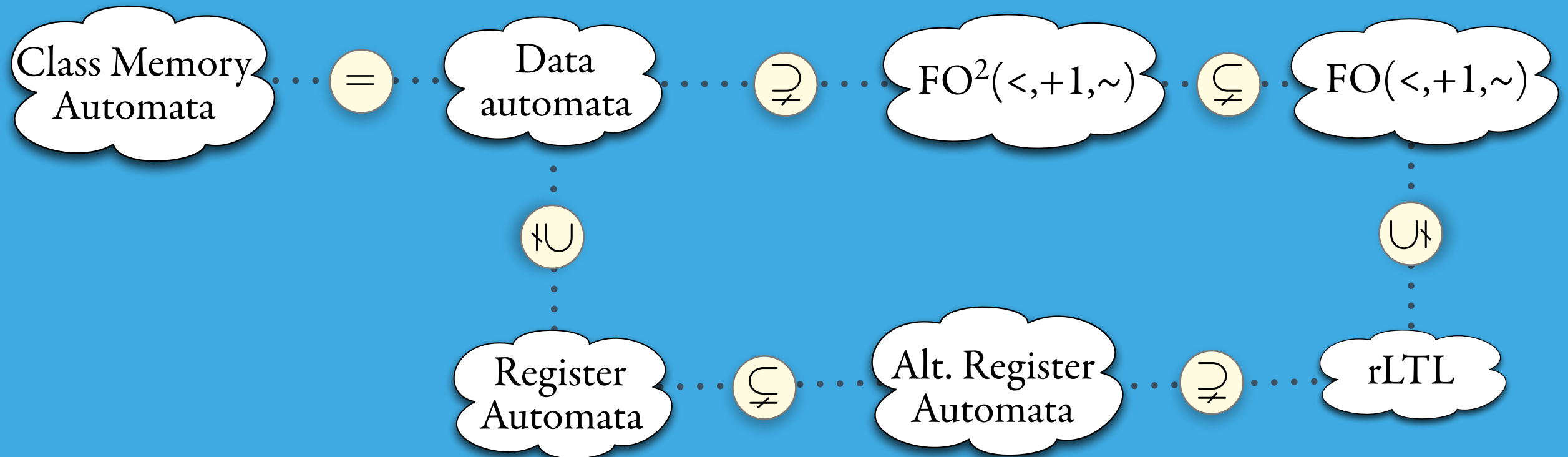
A zoo of formalisms on data words



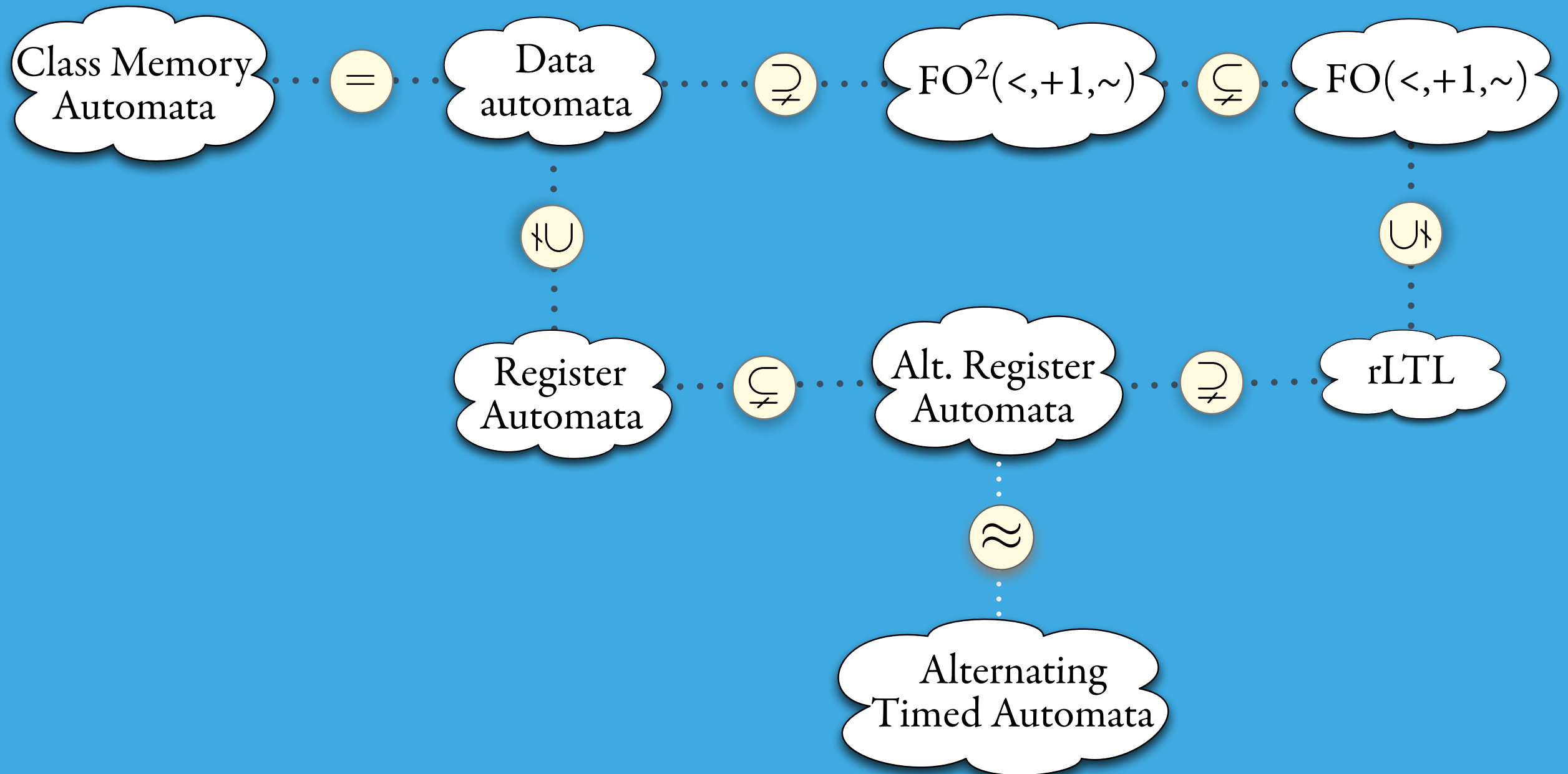
A zoo of formalisms on data words



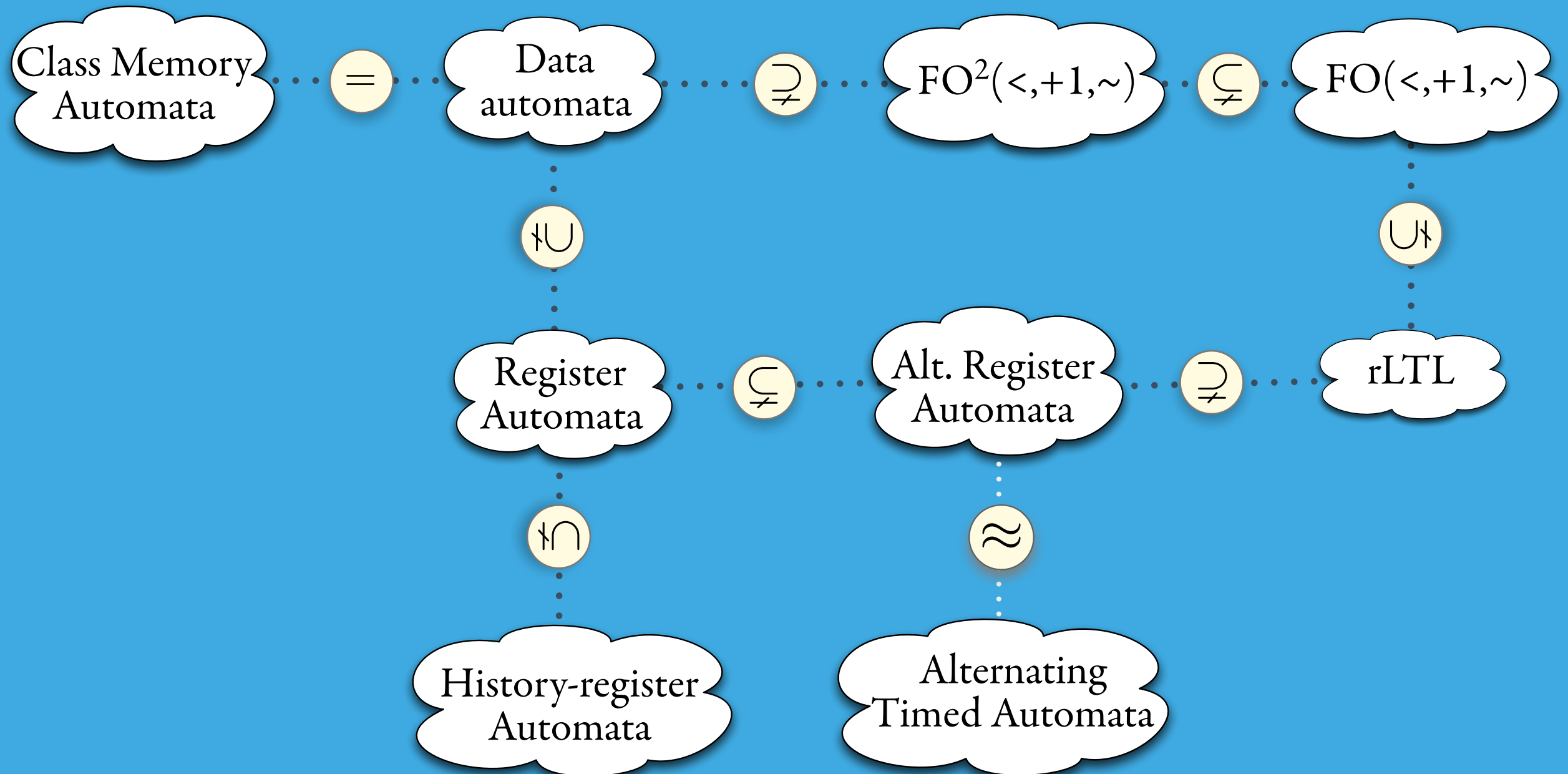
A zoo of formalisms on data words



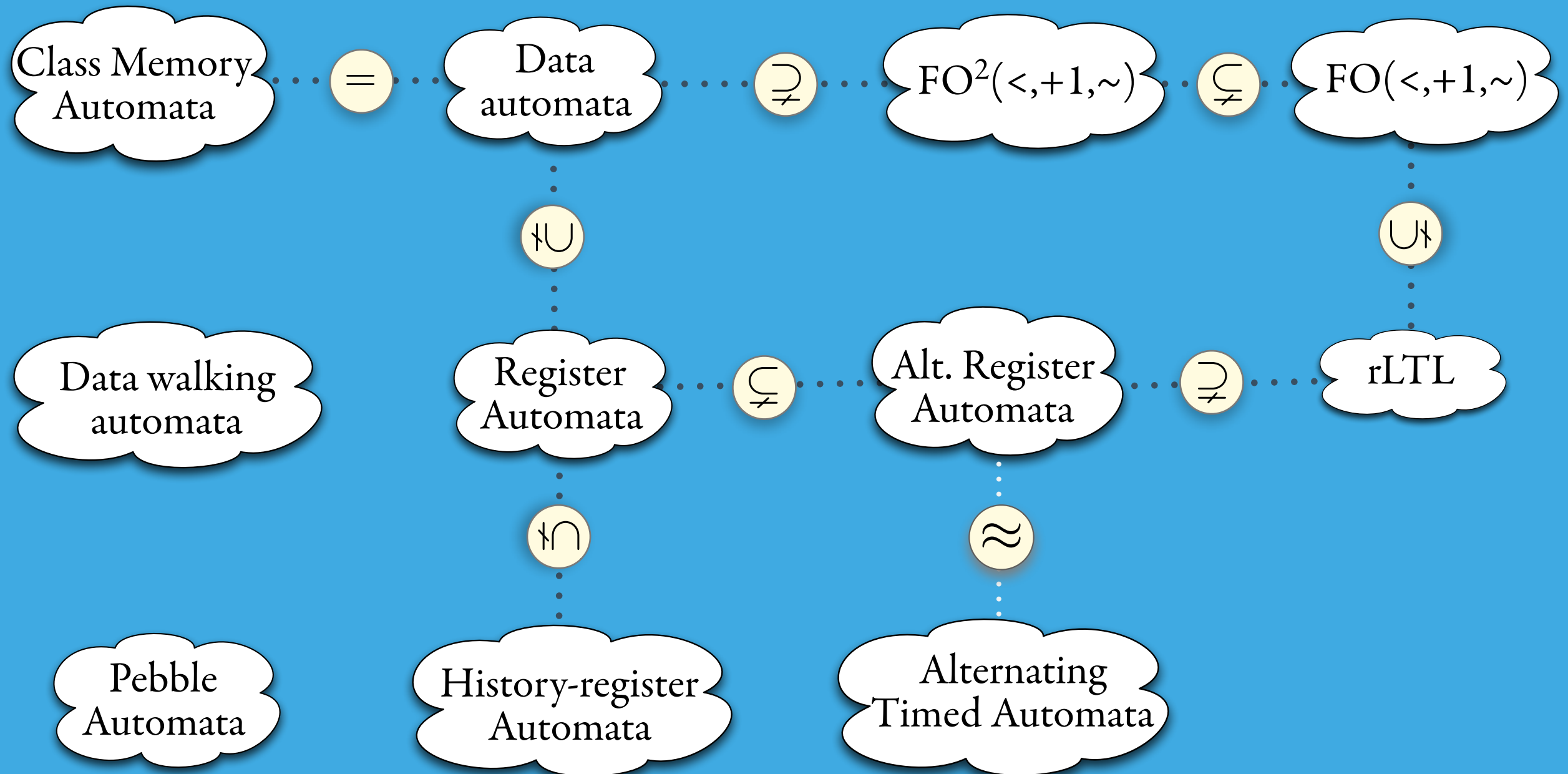
A zoo of formalisms on data words



A zoo of formalisms on data words



A zoo of formalisms on data words

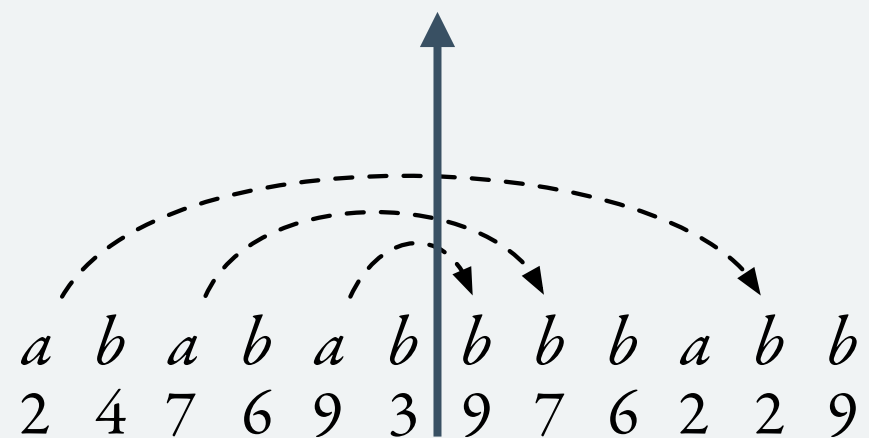


reasoning with
infinite alphabets \approx counting

<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>b</i>
2	4	7	6	9	3	9	7	6	2	2	9

*“for every *a* there is a *b* with same data value to its right”*

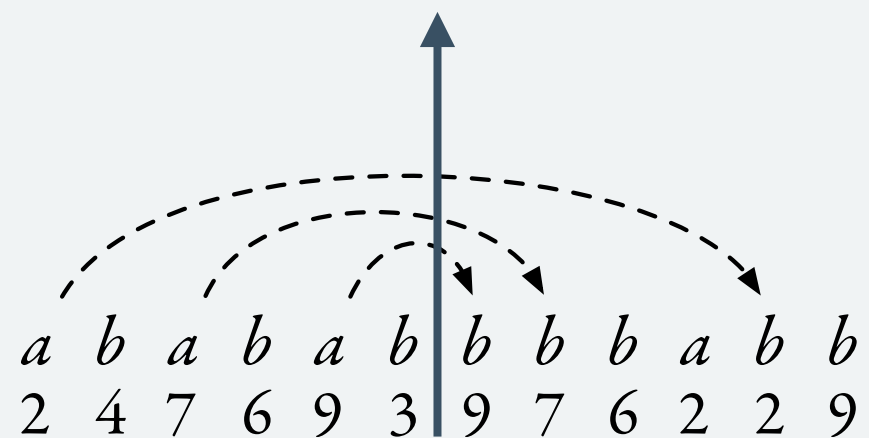
reasoning with
infinite alphabets \approx counting



*“for every *a* there is a *b* with same data value to its right”*

reasoning with *infinite* alphabets \approx counting

*there must be 3 distinct
data values to the right*



“for every a there is a b with same data value to its right”

Counting automata!



Counter systems

Counter systems

A solid red circle is positioned to the left of the text "Minsky machine".

Minsky machine

machine with counters and test for zero

Counter systems



Minsky machine

machine with counters and test for zero



Counter machine

we don't allow test for zero

Counter systems



Minsky machine

machine with counters and test for zero



Counter machine

we don't allow test for zero



Gainy counter machine

the machine broke down! Increments along the run

Counter systems

Counter systems



Reachability problem

is there a computation ending with counters in 0?

Counter systems




Reachability problem

is there a computation ending with counters in 0?



Control-state reachability problem

is there one ending with a given state?



**Wait! Why are we
talking about counter
automata?!**

Wait! Why are we
talking about counter
automata?!

Is this ESSLLI?

Am I in the
course about data
logics..?

Wait! Why are we
talking about counter
automata?!

Is this ESSLLI?

Am I in the
course about data
logics..?

logics on data words
 \approx
counter automata

Oh! Allright... (?)

Who's that guy?

Minsky Machine

- Minsky Machine = non-det. finite automata + **counters**
- A counter can only store a natural number (≥ 0)
- Operations on counters
 - Check if counter if zero
 - Increment counter by one
 - Decrement counter by one (only if $\neq 0$)

Minsky Machine

- $\mathcal{A} = (Q, q_0, \delta, k)$, automaton with k counters over finite statespace Q
- **Instructions:** $\delta \subseteq Q \times \{\text{inc}, \text{dec}, \text{tz}\} \times \{1, \dots, k\} \times Q$
- **Configurations:** $c \in Q \times \mathbb{N}^k$ *eg: $(q, (3, 0, 2))$*
- **Run:** defined by relation $(q, v) \rightsquigarrow (q', v')$ if there is $(q, \text{inst}, i, q') \in \delta$ so that v' is the result of applying instruction **inst** to counter **i**.
eg: $(q, (3, 0, 2)) \rightsquigarrow (q', (2, 0, 2))$ using $(q, \text{dec}(1), q') \in \delta$.

Minsky Machine

- Example: $\mathcal{A} = (\{q_0, q_1\}, q_0, \delta, 2)$, where
$$\delta = \{(q_0, \text{inc}(1), q_1), (q_1, \text{inc}(2), q_0)\}.$$
- A possible run:

$$(q_0, (0, 0)) \rightsquigarrow (q_1, (1, 0)) \rightsquigarrow (q_0, (1, 1)) \rightsquigarrow (q_1, (0, 0)) \rightsquigarrow \dots$$

Reachability problems

- **Reachability**: Given a counter automaton \mathcal{A} and a configuration (q,v) : is there a run leading to (q,v)
- **Control-state reachability**: Given a counter automaton \mathcal{A} and a state q : is there a run leading to (q,v) for some v ?

Reachability problems

Control-state reachability for Minsky Machines is **undecidable**, already for two counters.

Reachability problems

Control-state reachability for Minsky Machines is **undecidable**, already for two counters.

*What about
reachability?*

Reachability problems

2-counter Minsky machines are **Turing-complete**:

- ★ A TM can be simulated by **two stacks** (infinite tape is cut in half)
- ★ A stack can be simulated by **two counters** (one of the counters is the binary representation of the bits on the stack)
- ★ Four counters can be simulated by two counters (factorization of one of the counters is $2^a 3^b 5^c 7^d$)

Decidable restrictions

Two basic ways of turning Minsky Machines into a decidable model:

1. **no tests for zero**, or
2. allow a "faulty" behaviour, where counters can **non-deterministically increment** their value.

Counter Machine

- A counter machine = A Minsky machine without zero tests.
- Equivalent to: Vector Addition Systems (VAS), Petri Nets.
- Reachability and control-state reachability problems are decidable.
- Best bound for reachability: non-primitive recursive (hard proof).
[Sacerdote, Tenney, Mayr, Kosaraju, ...]
- Complexity of control-state reachability: ExpSpace-complete.
[Rackoff, Lipton]

Gainy Counter Machine

- It is defined as a Minsky Machine but inside a run there can be **non-deterministic increments** to any counter.
- Reachability / control-state reachability for Gainy Counter Machines is **decidable**, with (provably) non-primitive recursive complexity.

[Schoebelen, Abdulla & Jonsson, Finkel & Cécé & Iyer]

Minsky machine

Counter machine

Gainy counter machine

Reachability problem

Control-state reachability problem



Satisfiability problem for data logics