What is static analysis by abstract interpretation?

Example of static analysis (input)

```
n := n0;
i := n;
while (i <> 0 ) do
      j := 0;
      while (j <> i) do
            j := j + 1
      od;
      i := i - 1
od
```

Example of static analysis (output)

```
{n0>=0}
   n := n0;
{n0=n,n0>=0}
   i := n;
{n0=i,n0=n,n0>=0}
   while (i <> 0 ) do
      {n0=n,i>=1,n0>=i}
         j := 0;
      {n0=n,j=0,i>=1,n0>=i}
         while (j <> i) do
            {n0=n,j>=0,i>=j+1,n0>=i}
               j := j + 1
            {n0=n,j>=1,i>=j,n0>=i}
         od;
      {n0=n,i=j,i>=1,n0>=i}
         i := i - 1
      {i+1=j,n0=n,i>=0,n0>=i+1}
   od
{n0=n,i=0,n0>=0}
```

Example of static analysis (safety)

```
{n0>=0}
   n := n0;
{n0=n,n0>=0}
   i := n;
{n0=i,n0=n,n0>=0}
   while (i <> 0 ) do
      {n0=n,i>=1,n0>=i}
         j := 0;
      {n0=n,j=0,i>=1,n0>=i}
         while (j <> i) do
            {n0=n,j>=0,i>=j+1,n0>=i}
               j := j + 1        ←— j < n0 so no upper overflow
            {n0=n,j>=1,i>=j,n0>=i}
         od;
      {n0=n,i=j,i>=1,n0>=i}
         i := i - 1              ←— i > 0 so no lower overflow
      {i+1=j,n0=n,i>=0,n0>=i+1}
   od
{n0=n,i=0,n0>=0}
```

n0 must be initially nonnegative (otherwise the program does not terminate properly)

Static analysis by abstract interpretation

Verification: define and prove automatically a property of the possible behaviors of a complex computer program;

Abstraction: the reasoning/calculus can be done on an abstraction of these behaviors dealing only with those elements of the behaviors related to the considered property;

Theory: abstract interpretation.

Example of static analysis

Verification: absence of runtime errors;

Abstraction: polyhedral abstraction (affine inequalities);

Theory: abstract interpretation.

Potential impact of runtime errors

- 50% of the security attacks on computer systems are through buffer overruns [1]!
- Embedded computer system crashes easily result from overflows of various kinds.

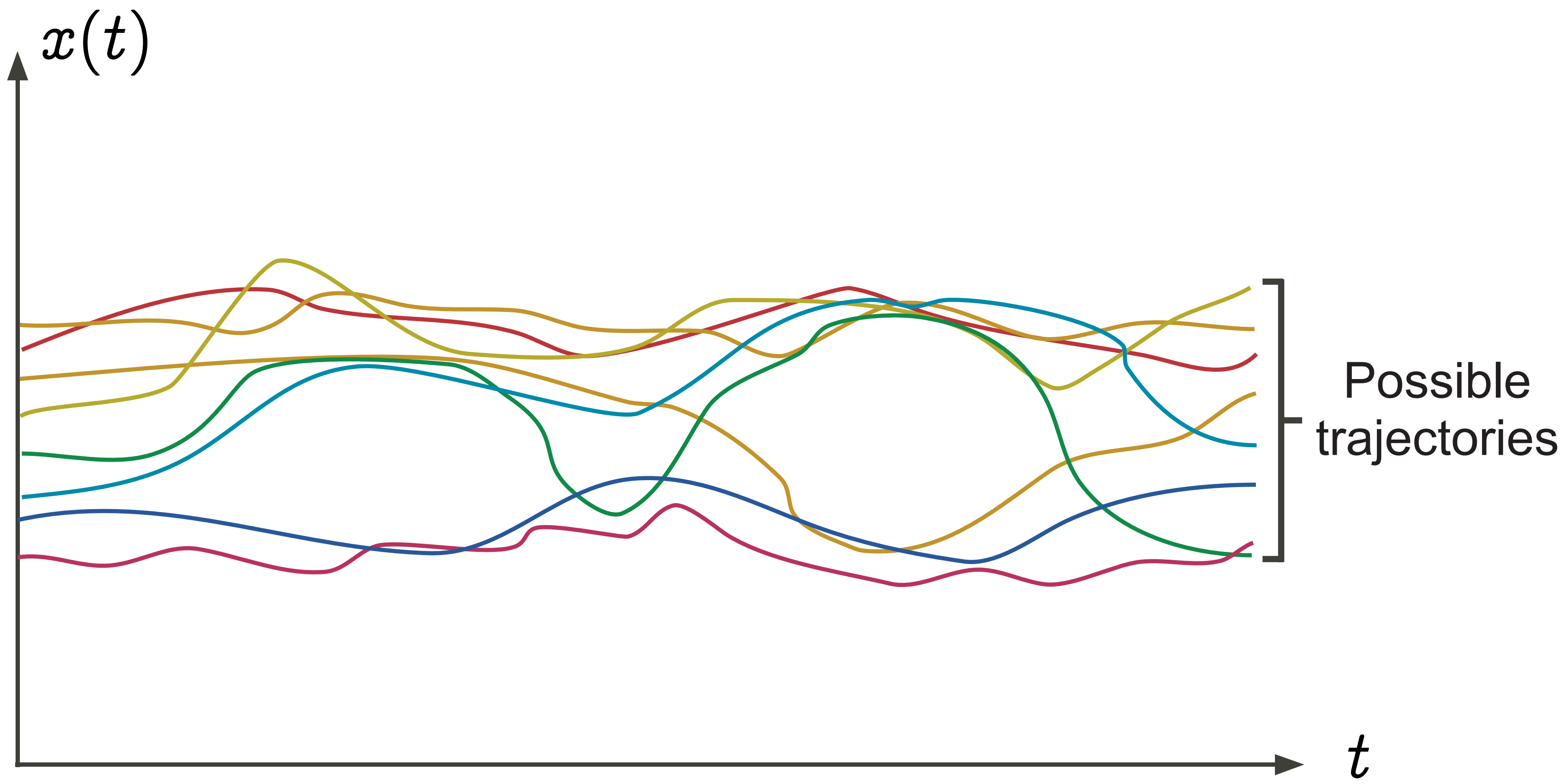
[1] See for example the Microsoft Security Bulletins MS02-065, MS04-011, etc.

A very <u>informal</u> introduction to the principles of abstract interpretation

Semantics

The *concrete semantics* of a program formalizes (is a mathematical model of) the set of all its possible executions in all possible execution environments.

Graphic example: Possible behaviors



Undecidability

- The concrete mathematical semantics of a program is an "infinite" mathematical object, *not computable*;
- All non trivial questions on the concrete program semantics are *undecidable*.

Example: Kurt Gödel argument on termination

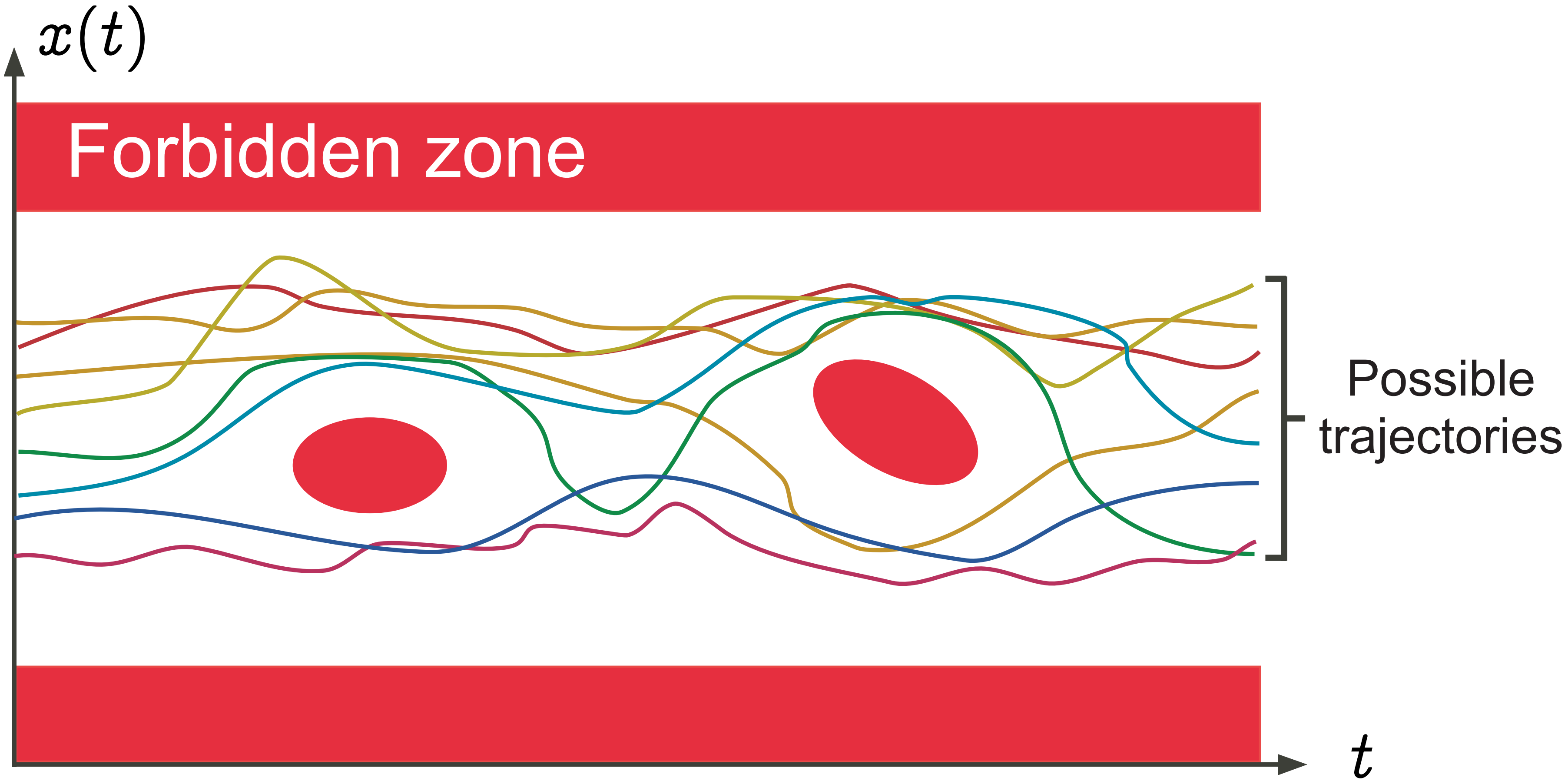
- Assume `termination(P)` would always terminates and returns `true` iff `P` always terminates on all input data;
- The following program yields a contradiction

$$\texttt{P} \equiv \texttt{while termination(P) do skip od.}$$

Graphic example: Safety properties

The *safety properties* of a program express that no possible execution in any possible execution environment can reach an erroneous state.

Graphic example: Safety property



Safety proofs

– A safety proof consists in proving that the intersection of the program concrete semantics and the forbidden zone is empty;

– Undecidable problem (the concrete semantics is not computable);

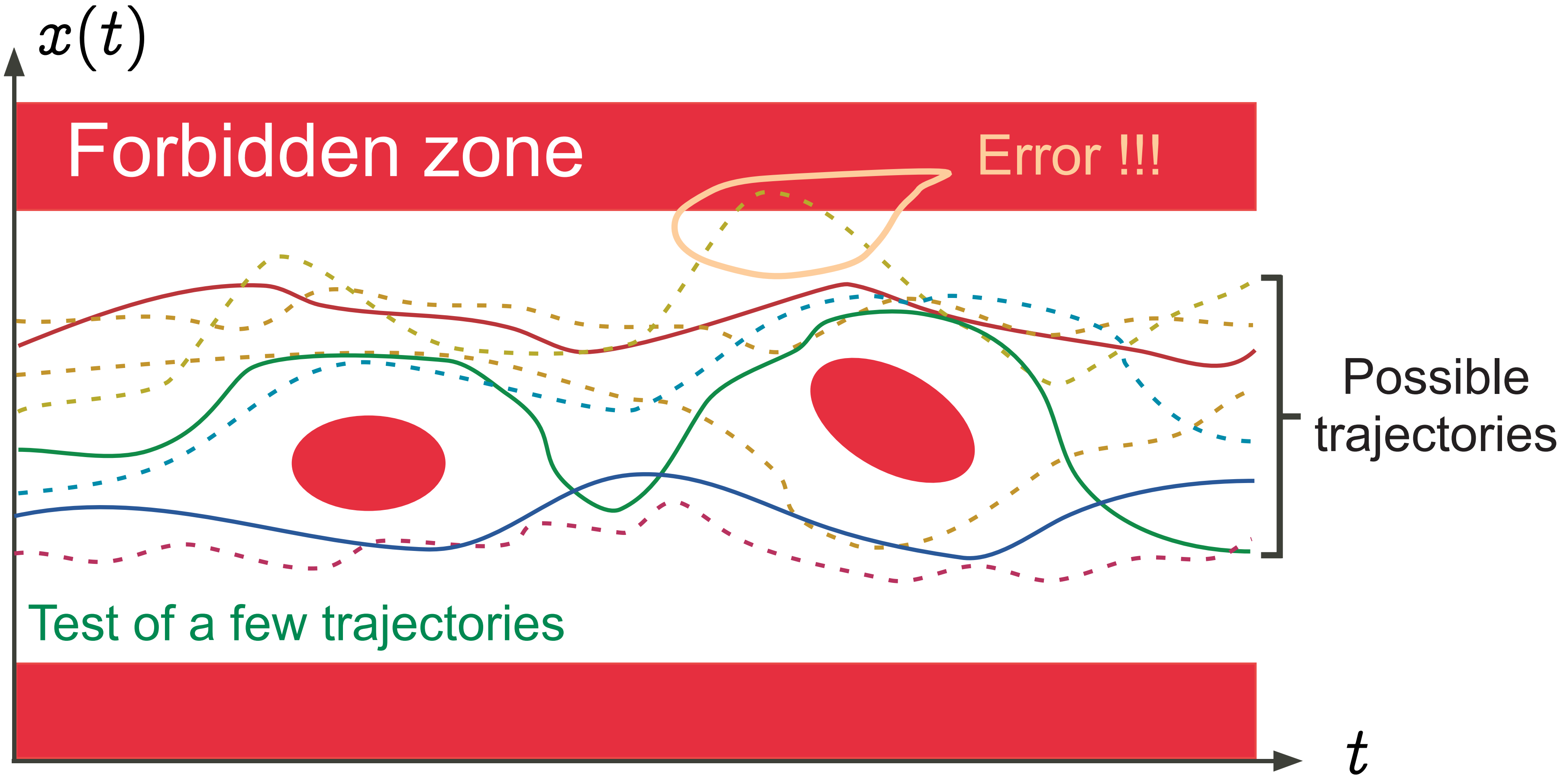
– Impossible to provide completely automatic answers with finite computer resources and neither human interaction nor uncertainty on the answer [2].

[2] e.g. probabilistic answer.

Test/debugging

- consists in considering a subset of the possible executions;
- not a correctness proof;
- absence of coverage is the main problem.

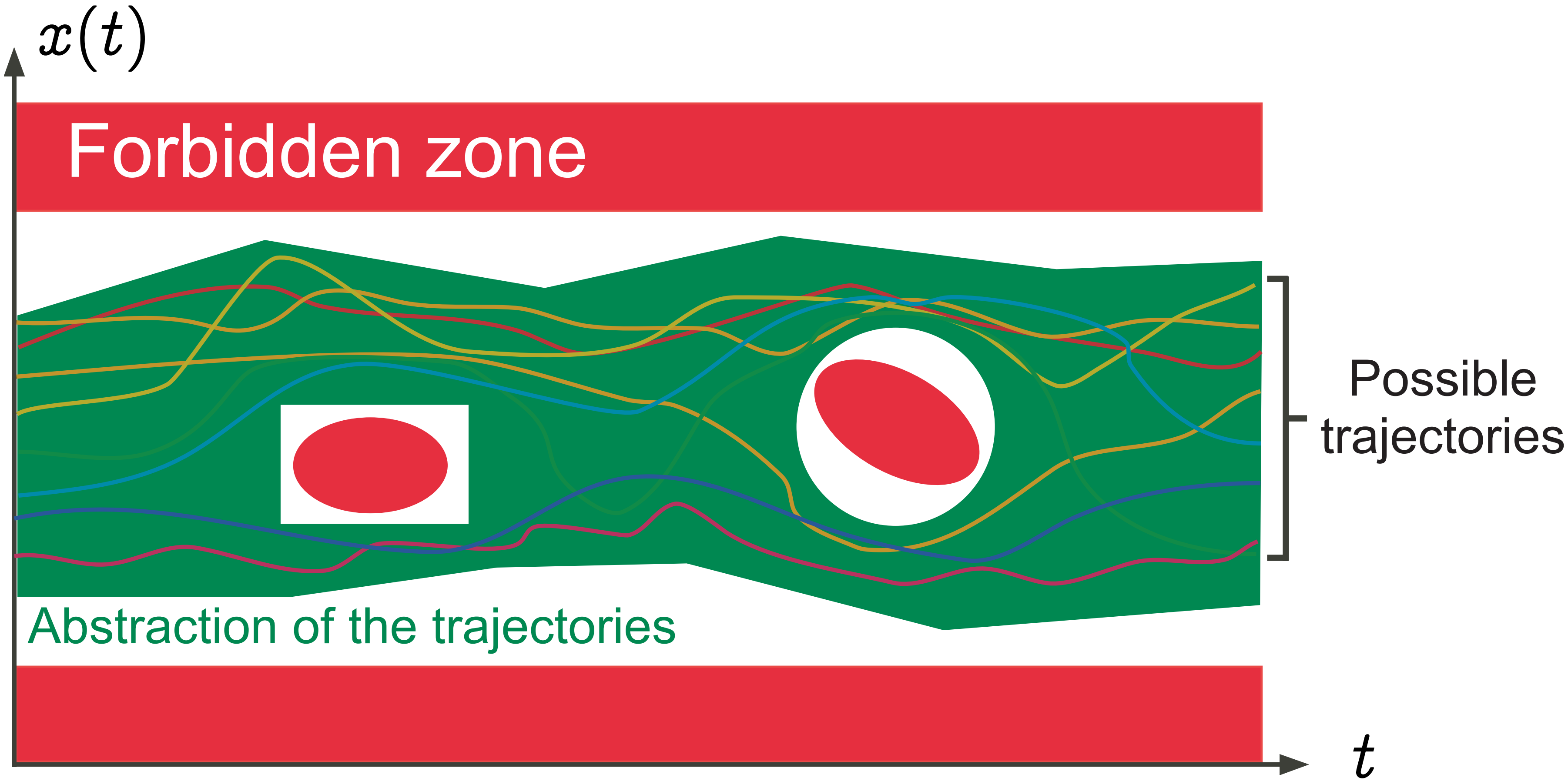
Graphic example: Property test/simulation



Abstract interpretation

- consists in considering an *abstract semantics*, that is to say a superset of the concrete semantics of the program;
- hence the abstract semantics covers all possible concrete cases;
- correct: if the abstract semantics is safe (does not intersect the forbidden zone) then so is the concrete semantics.

Graphic example: Abstract interpretation



Formal methods

Formal methods are abstract interpretations, which differ in the way to obtain the abstract semantics:

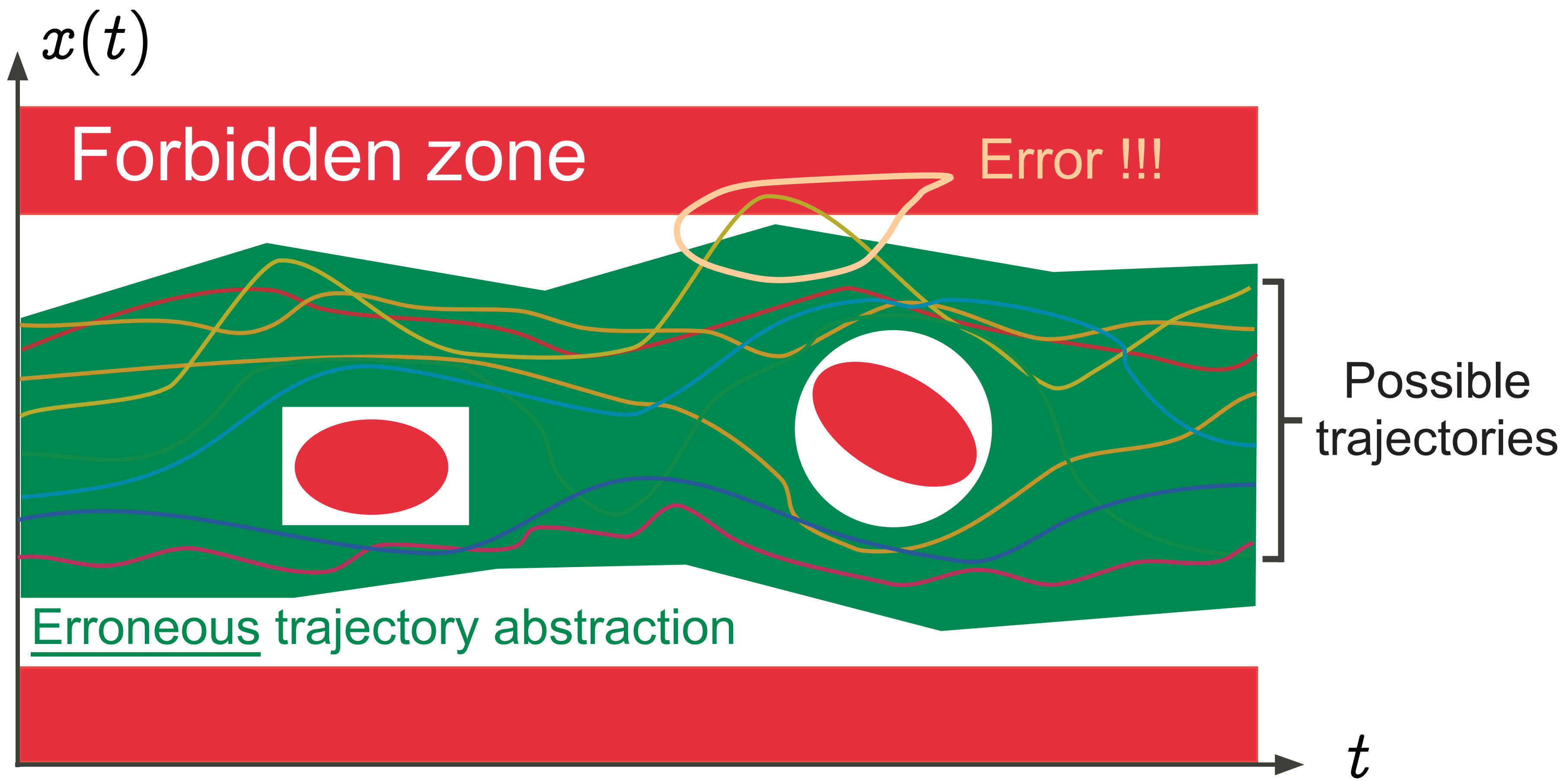
– "*model checking*":
  - the abstract semantics is given manually by the user;
  - in the form of a finitary model of the program execution;
  - can be computed automatically, by techniques relevant to static analysis.

- “*deductive methods*”:
  - the abstract semantics is specified by verification conditions;
  - the user must provide the abstract semantics in the form of inductive arguments (e.g. invariants);
  - can be computed automatically by methods relevant to static analysis.
- “*static analysis*”: the abstract semantics is computed automatically from the program text according to predefined abstractions (that can sometimes be tailored automatically/manually by the user).

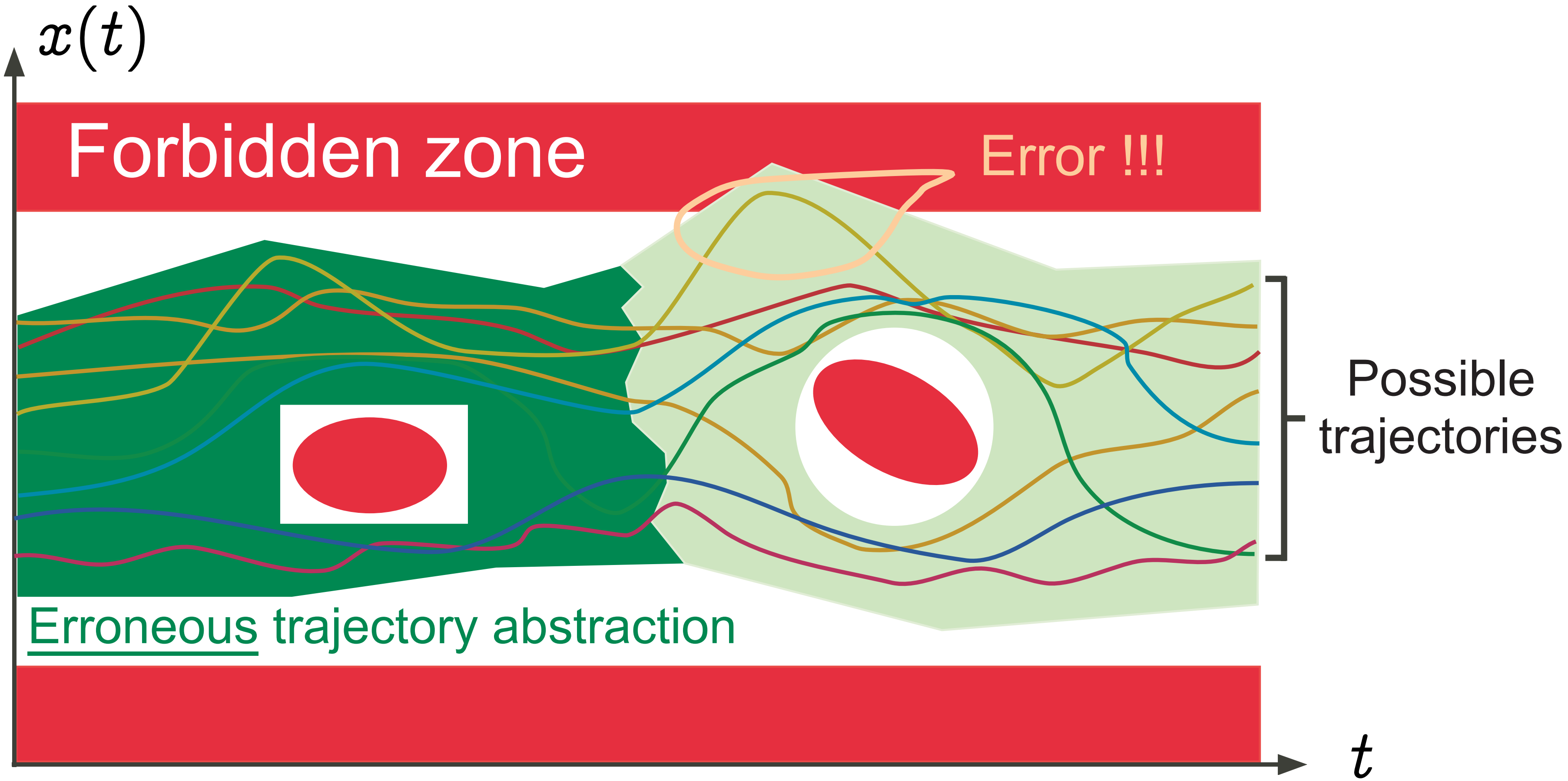
Required properties of the abstract semantics

– sound so that no possible error can be forgotten;
– precise enough (to avoid false alarms);
– as simple/abstract as possible (to avoid combinatorial explosion phenomena).

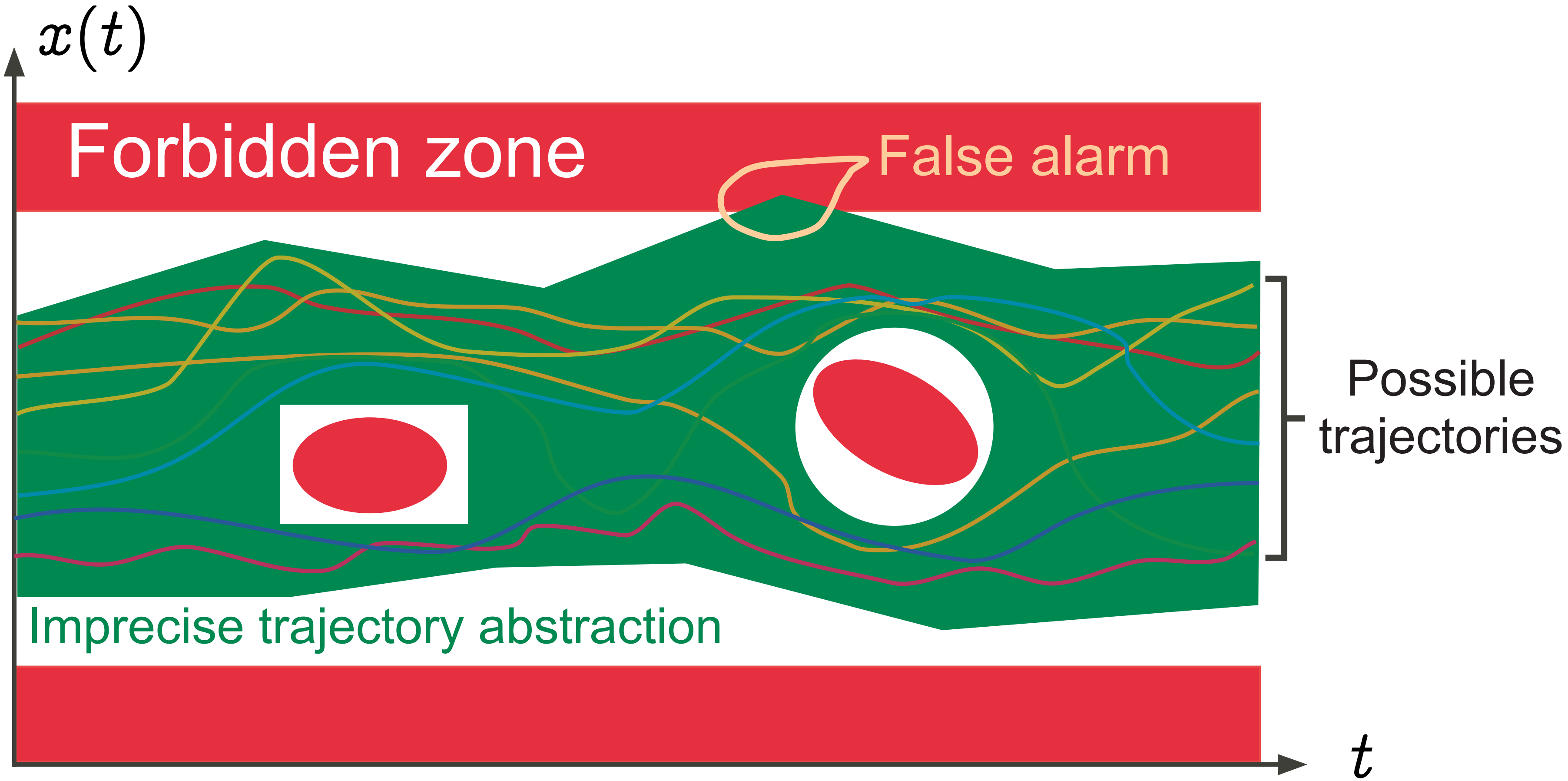
Graphic example: Erroneous abstraction — I



Graphic example: Erroneous abstraction — II



Graphic example: Imprecision $\Rightarrow$ false alarms



Abstract domains

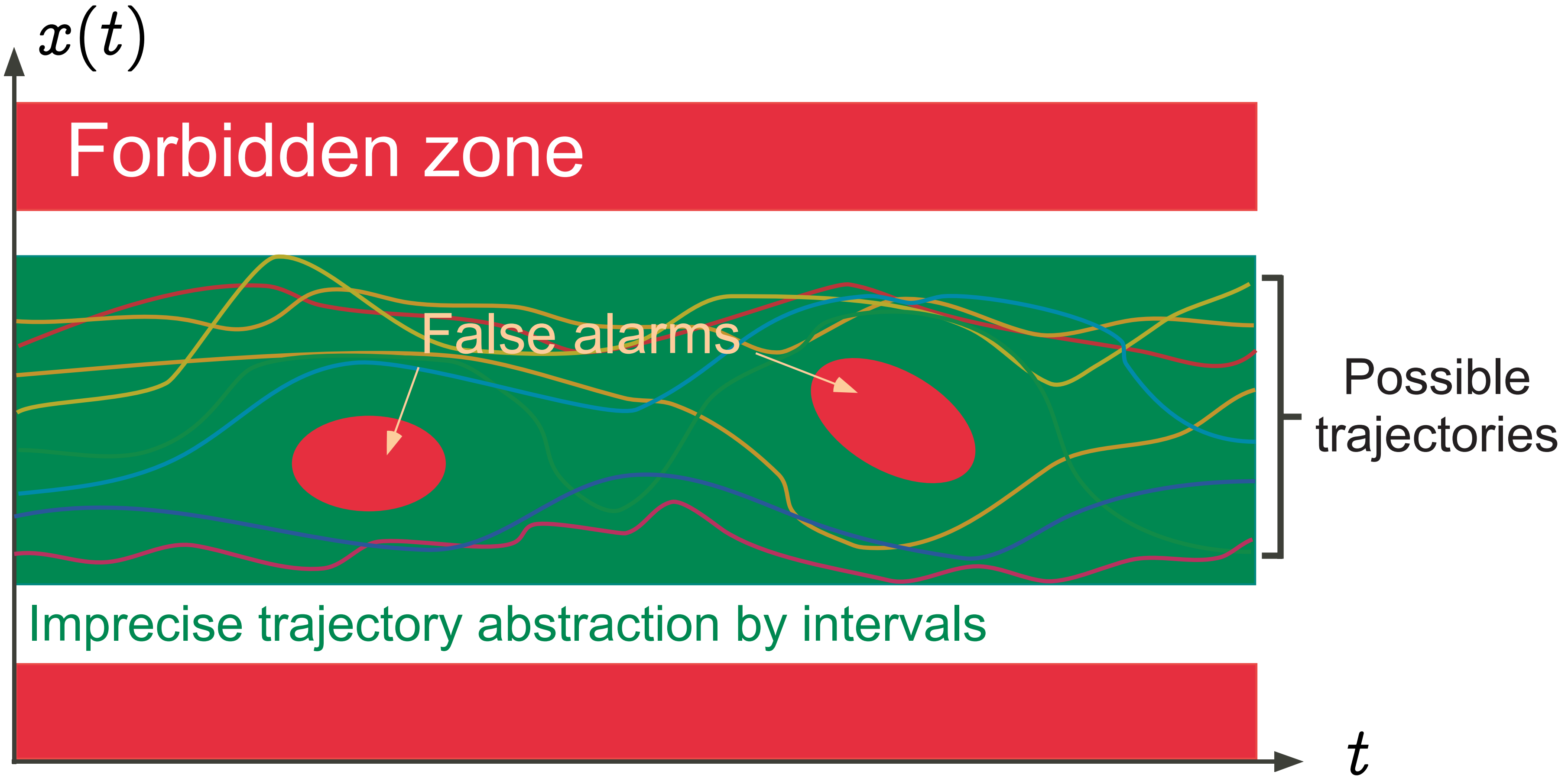
Standard abstractions

– that serve as a basis for the design of static analyzers:

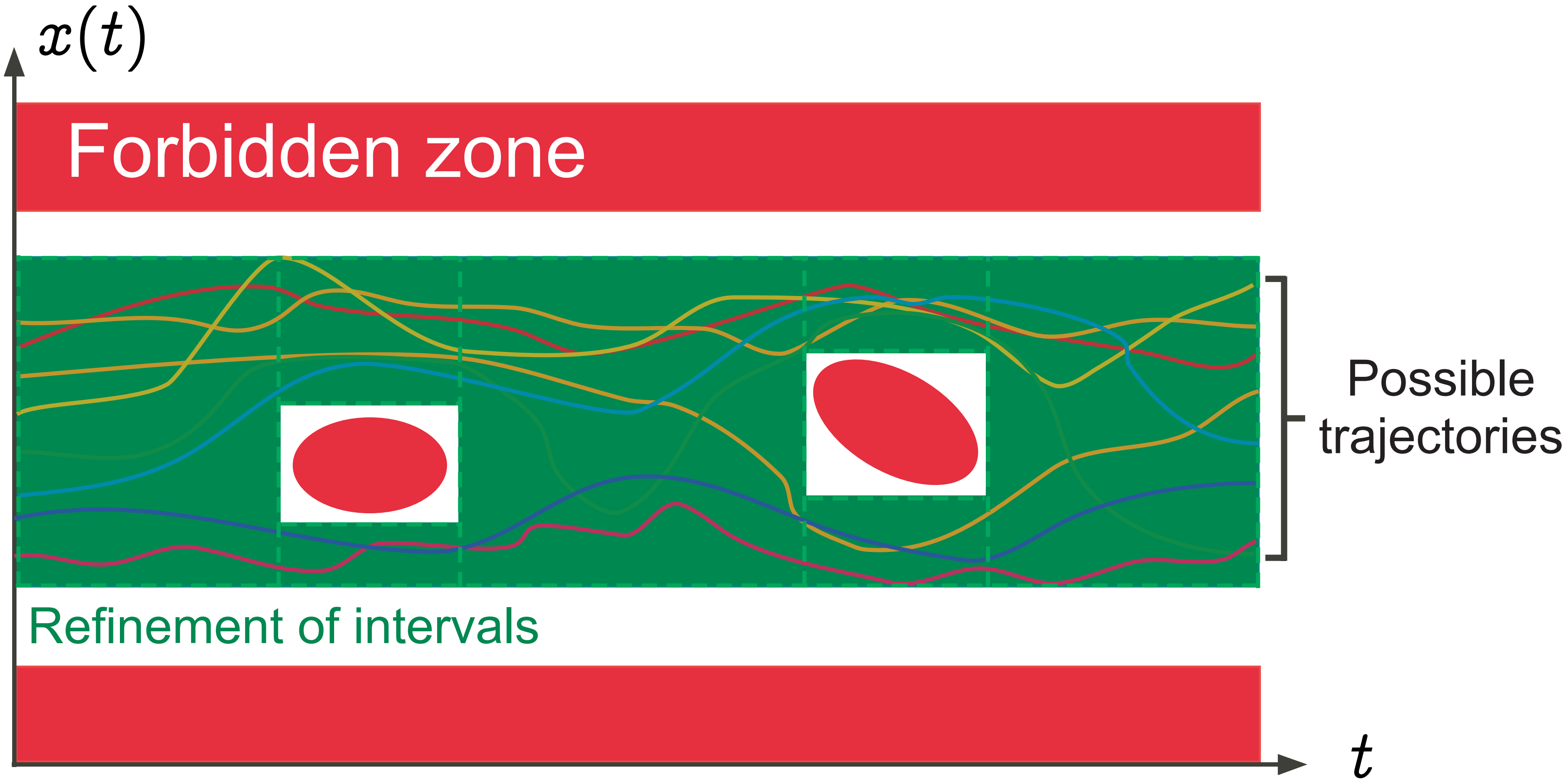
  - abstract program data,
  - abstract program basic operations;
  - abstract program control (iteration, procedure, concurrency, . . .);

– can be parametrized to allow for manual adaptation to the application domains.

Graphic example: Standard abstraction by intervals



Graphic example: A more refined abstraction



A very <u>informal</u> introduction to static analysis algorithms

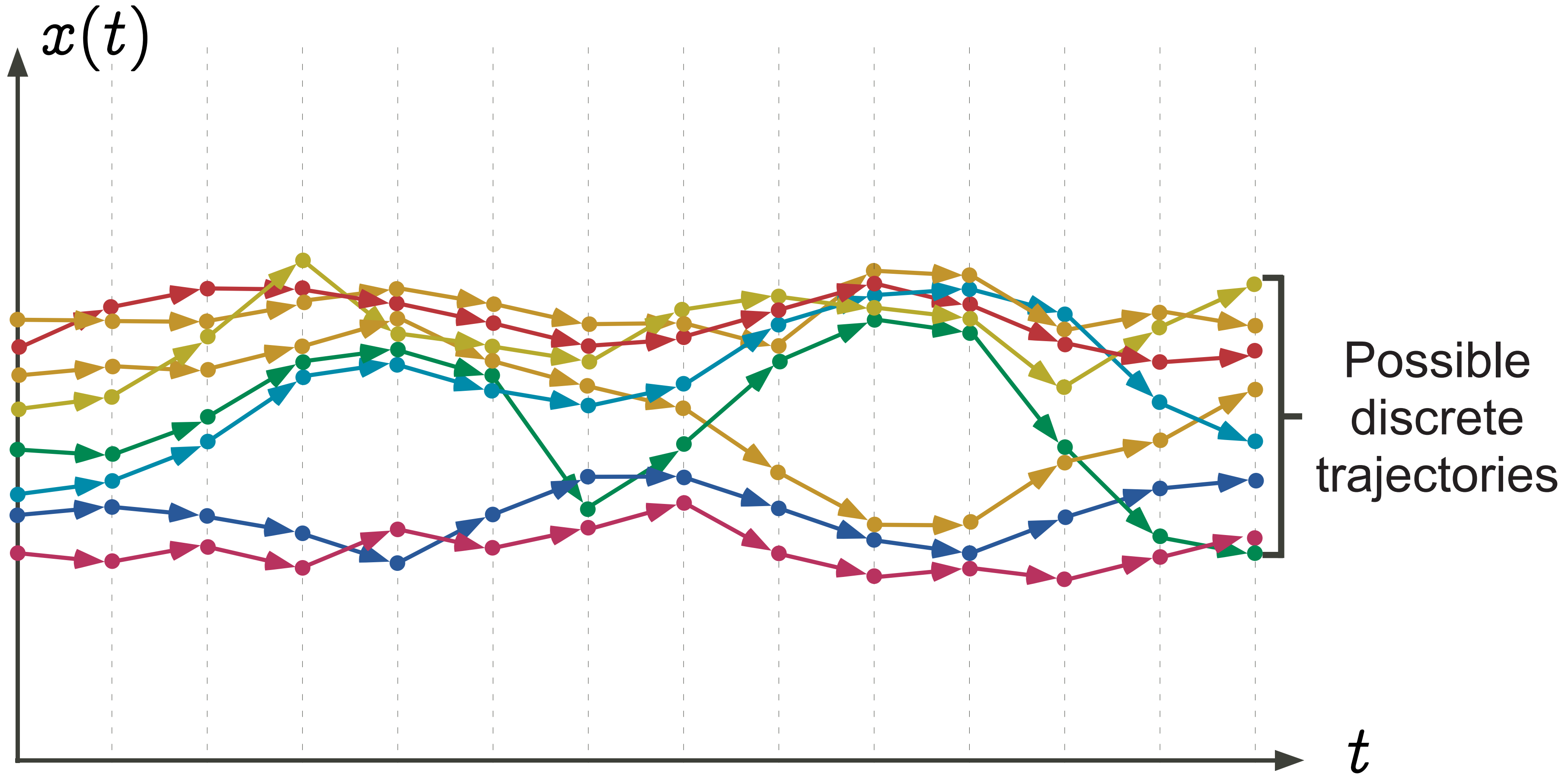
Trace semantics

Trace semantics

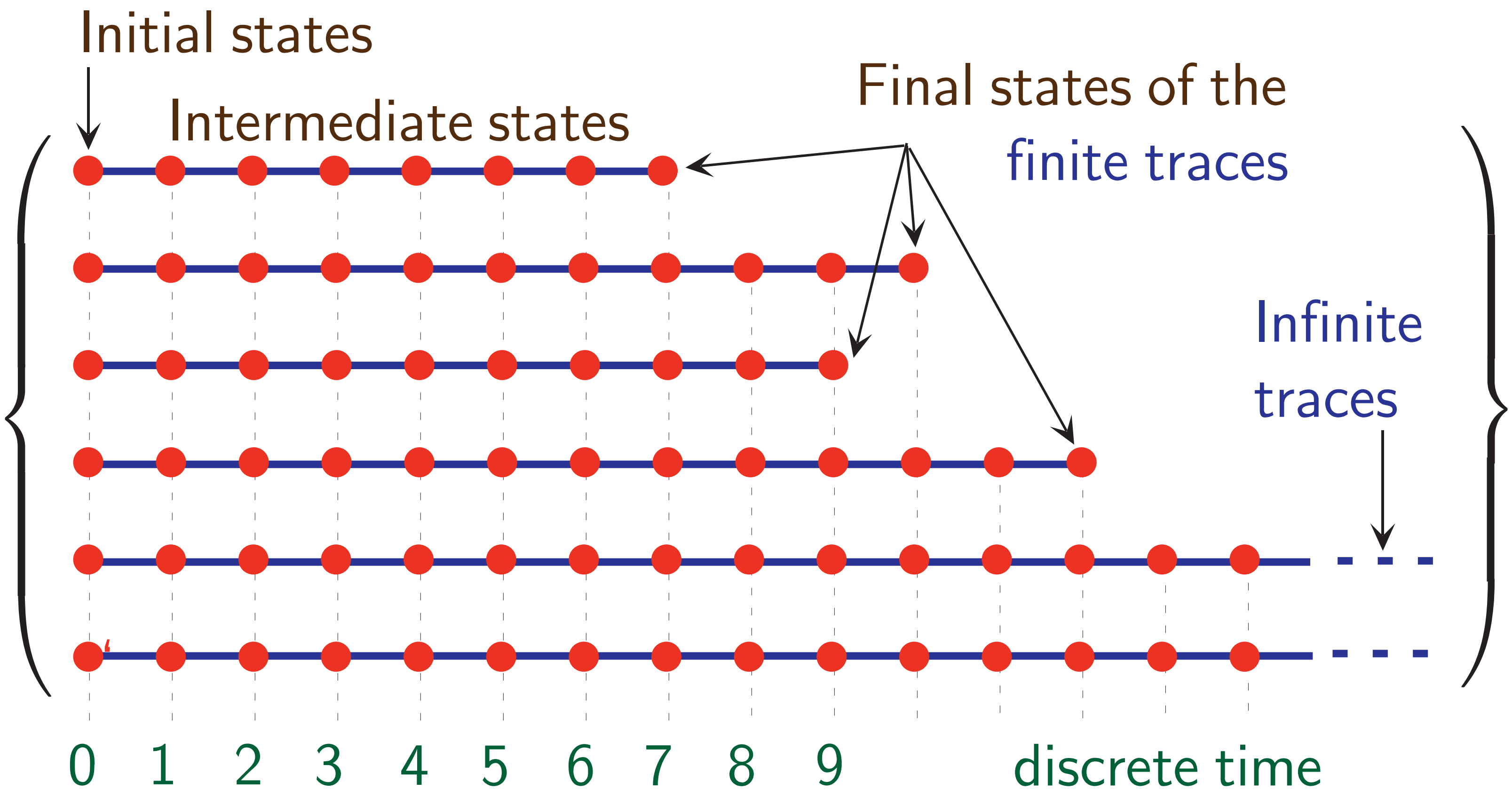
– Consider (possibly infinite) traces that is series of states corresponding to executions described by discrete transitions;

– The collection of all such traces, starting from the initial states, is the trace semantics.

Graphic example: Small-steps transition semantics

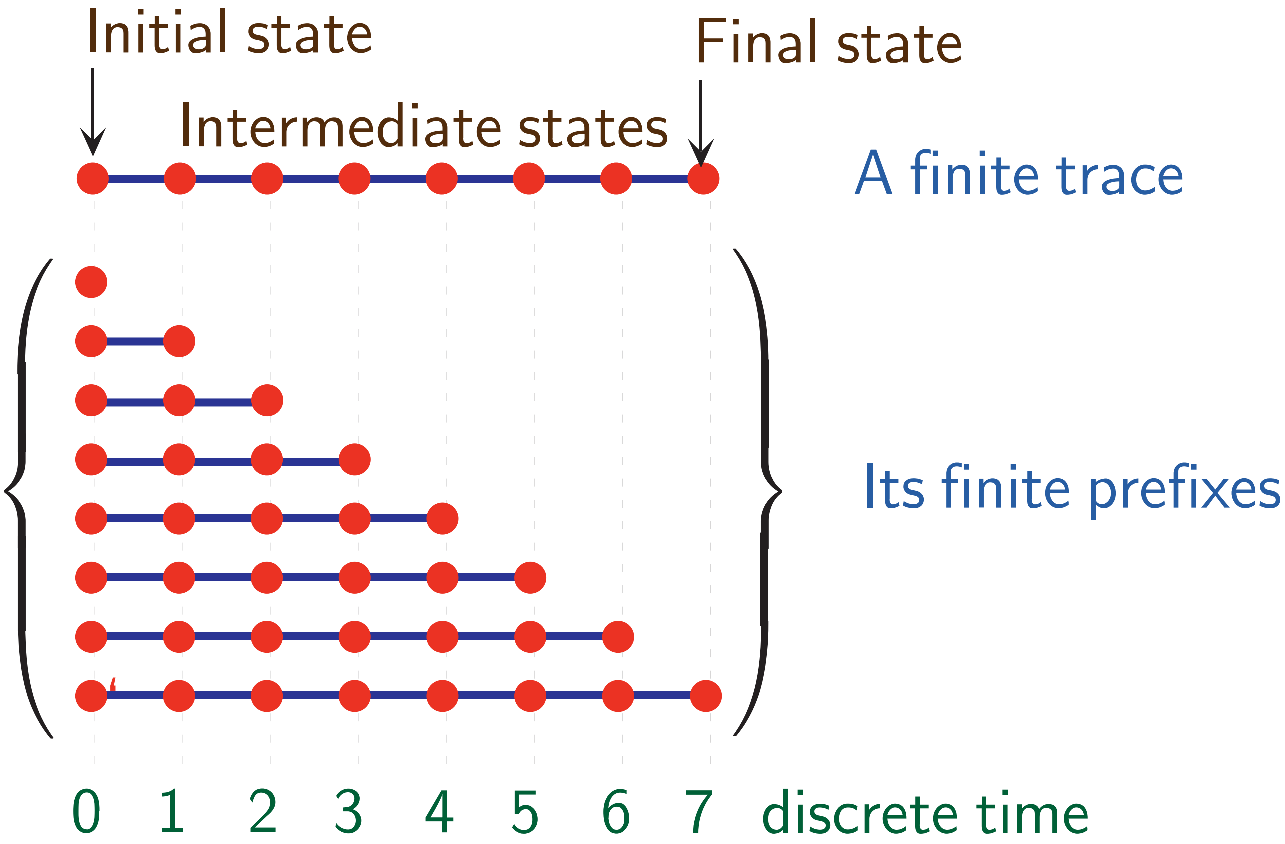


Trace semantics, intuition

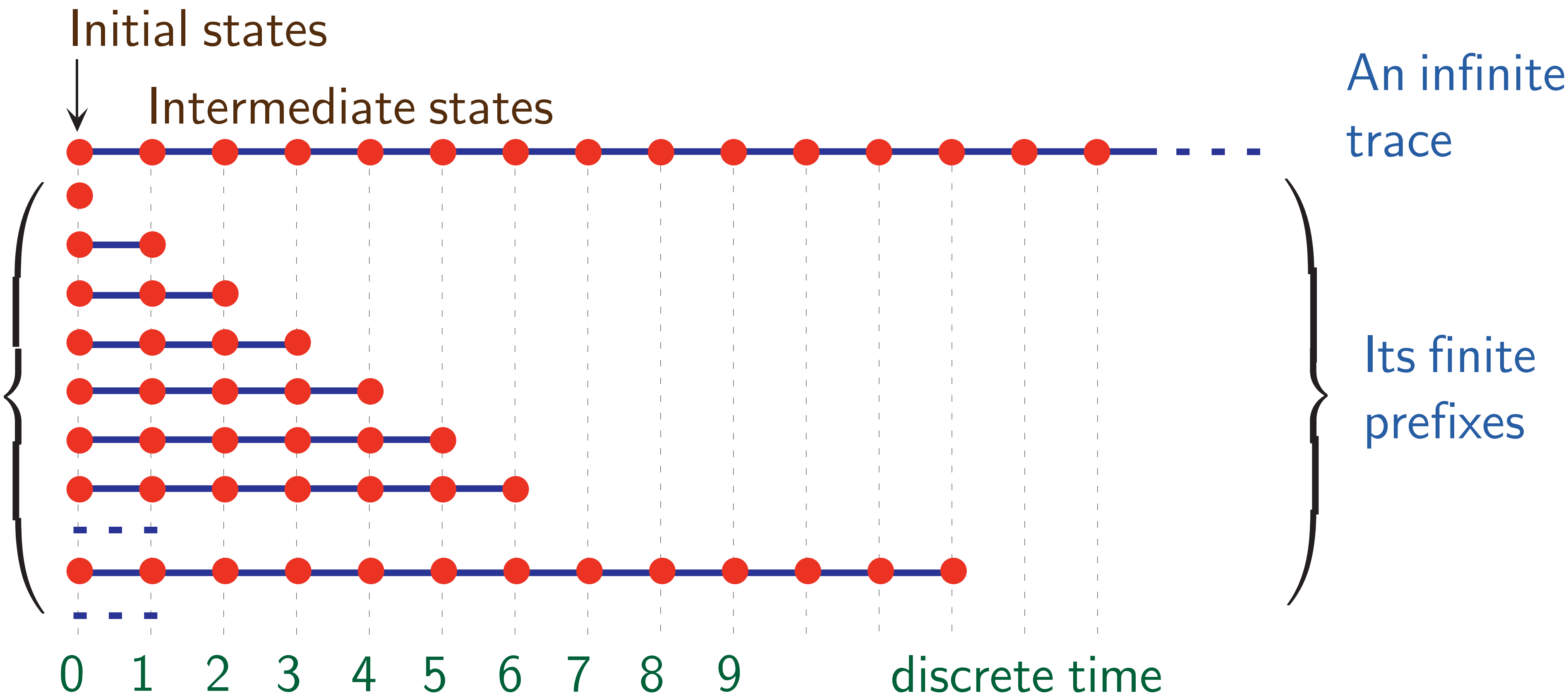


Prefix trace semantics

Prefixes of a finite trace



Prefixes of an infinite trace



Prefix trace semantics

Trace semantics: maximal finite and infinite behaviors

Prefix trace semantics: finite prefixes of the maximal behaviors

Abstraction

This is an abstraction. For example:

Trace semantics: $\{a^n b \mid n \geq 0\}$

Prefix trace semantics: $\{a^n \mid n \geq 0\} \cup \{a^n b \mid n \geq 0\}$

Is there of possible behavior with infinitely many successive a?

- Trace semantics: no
- Prefix trace semantics: I don't know

Prefix trace semantics in fixpoint form

Least <u>Fixpoint</u> Prefix Trace Semantics

$$
\begin{aligned}
\textbf{Prefixes} = \; & \{\bullet \mid \bullet \text{ is an initial state}\} \\
\cup \; & \{\bullet\!\!-\!\!-\!\!-\ldots-\!\!-\!\!-\!\!\bullet\!\!-\!\!-\!\!-\!\!\bullet \mid \bullet\!\!-\!\!-\!\!-\ldots-\!\!-\!\!-\!\!\bullet \in \textbf{Prefixes} \\
& \quad \& \quad \bullet\!\!-\!\!-\!\!-\!\!\bullet \text{ is a transition step}\}
\end{aligned}
$$

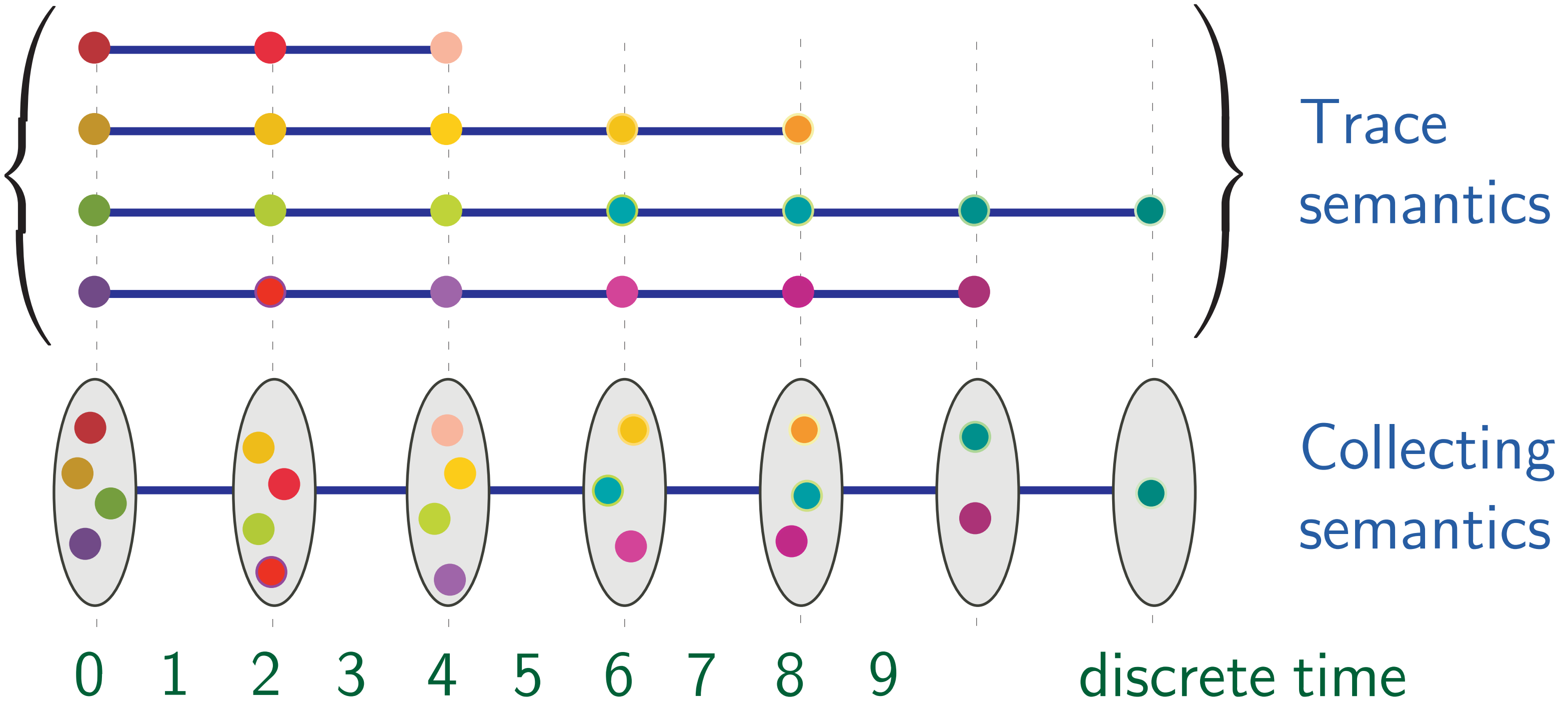
- In general, the equation $\textbf{Prefixes} = F(\textbf{Prefixes})$ may have multiple solutions;
- Choose the least one for subset inclusion $\subseteq$.

- Abstractions of this equation lead to effective iterative analysis algorithms.

Collecting semantics

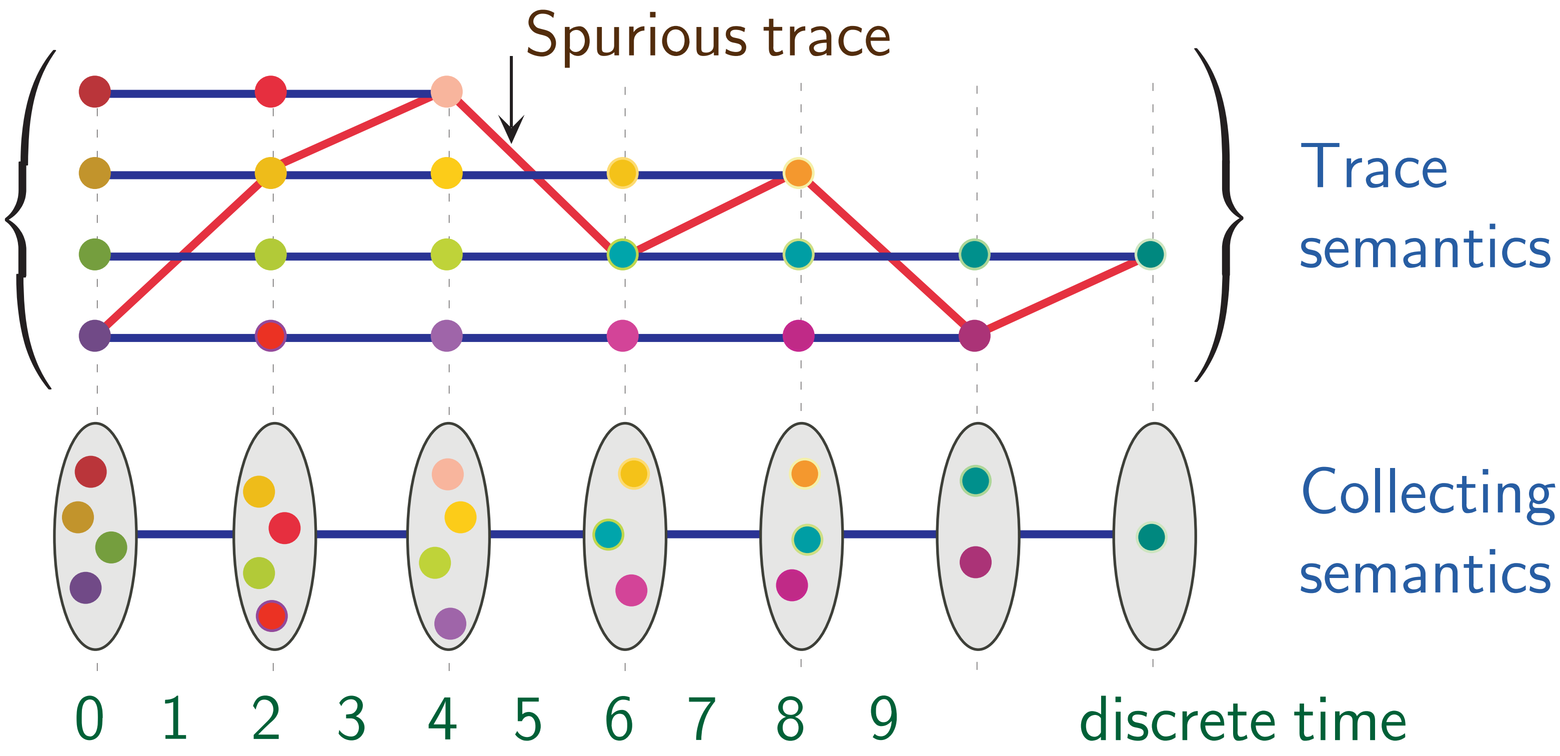
Collecting semantics

– Collect all states that can appear on some trace at any given discrete time:

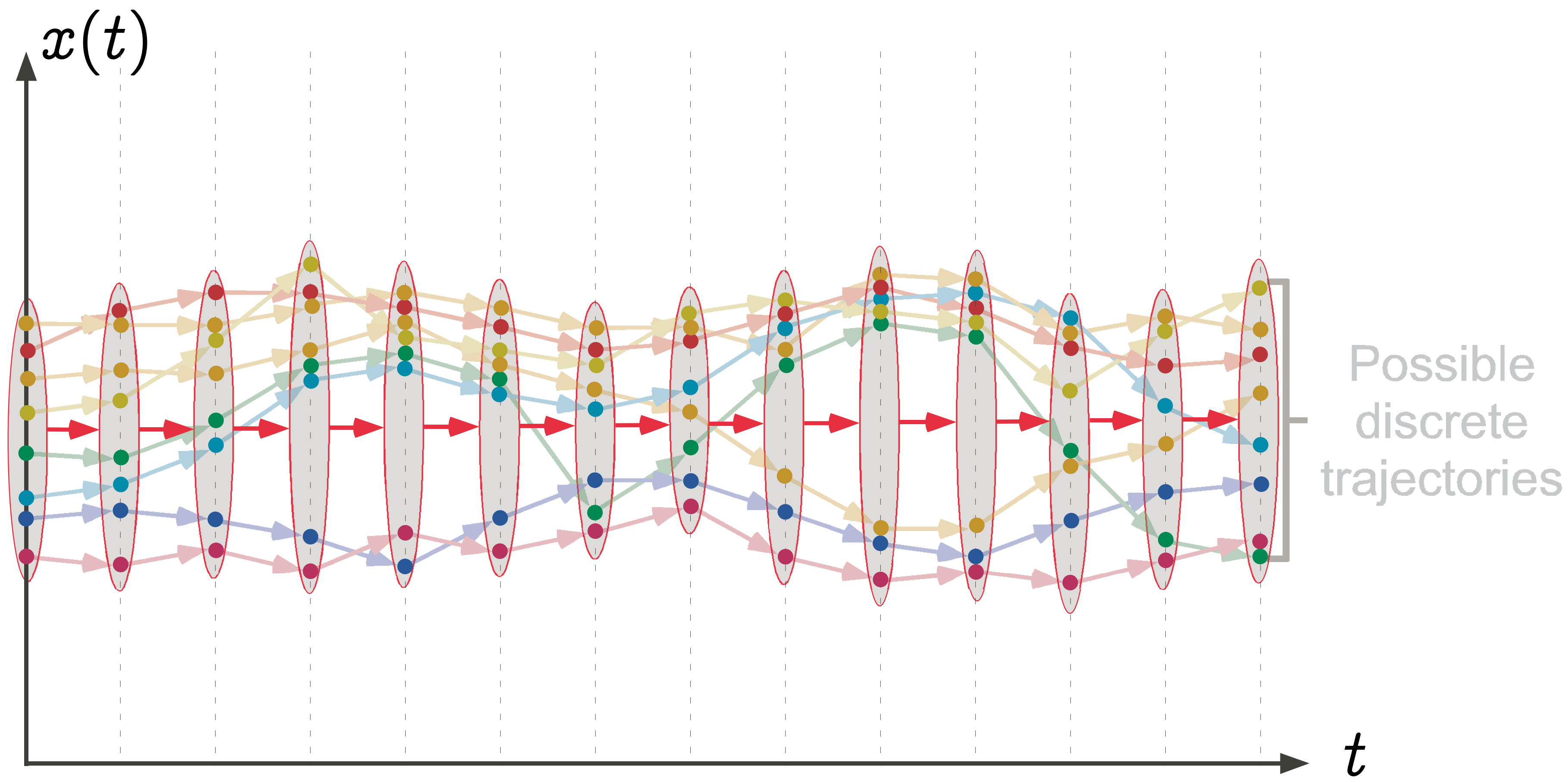


Collecting abstraction

– This is an abstraction. Does the red trace exists? Trace semantics: no, collecting semantics: I don't know.

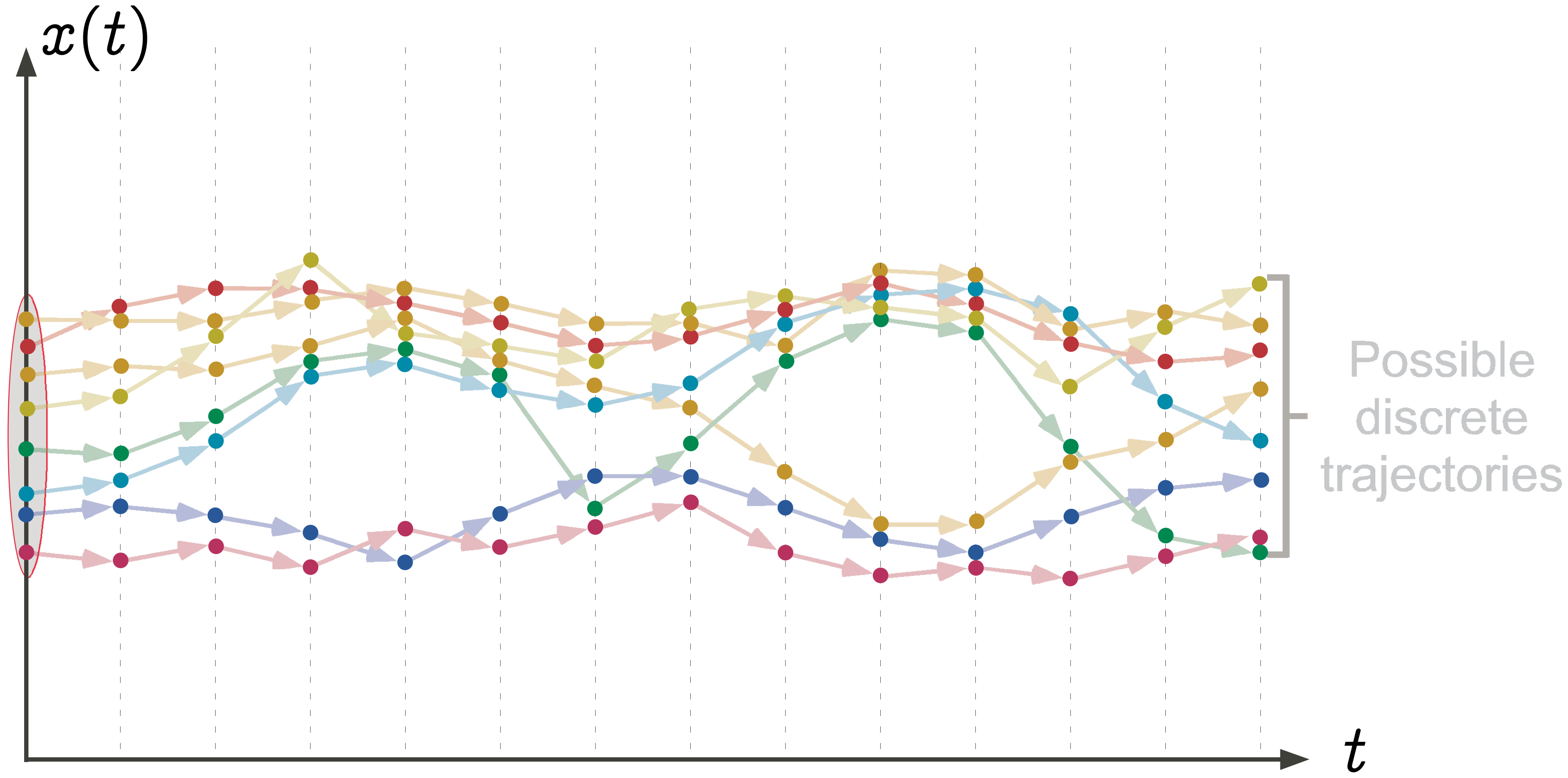


Graphic example: collecting semantics

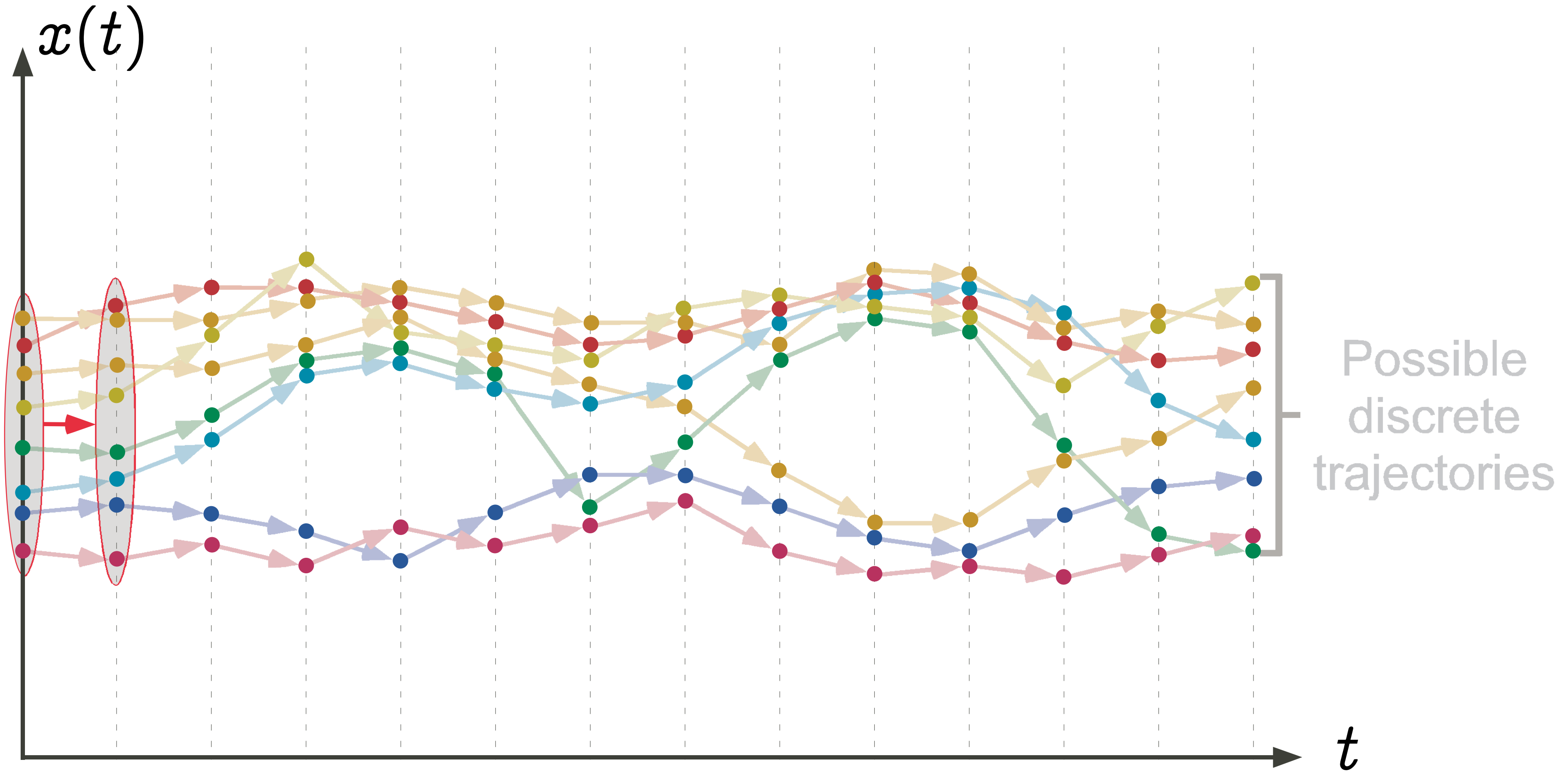


Collecting semantics in fixpoint form

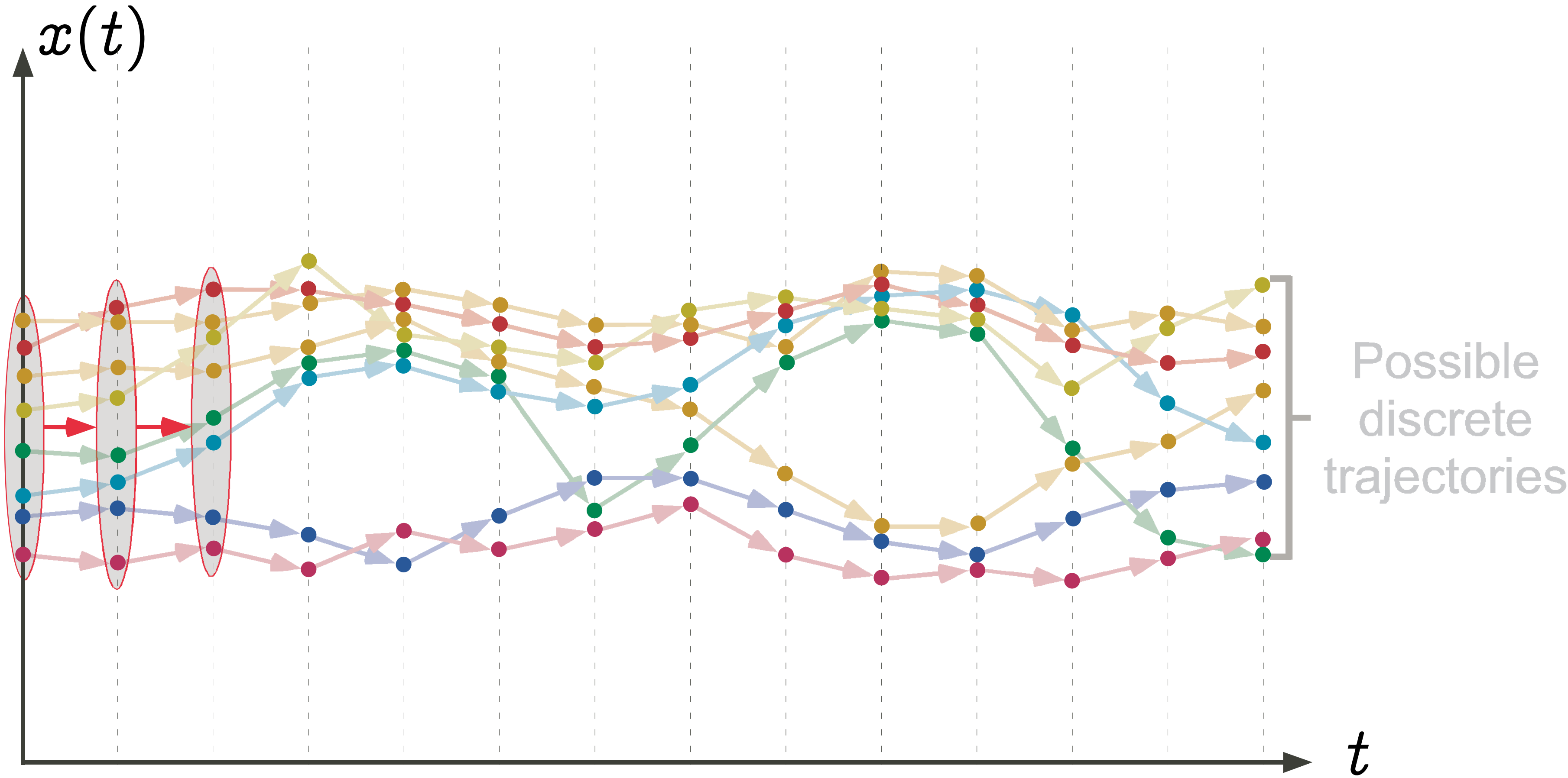
Graphic example: collecting semantics in fixpoint form



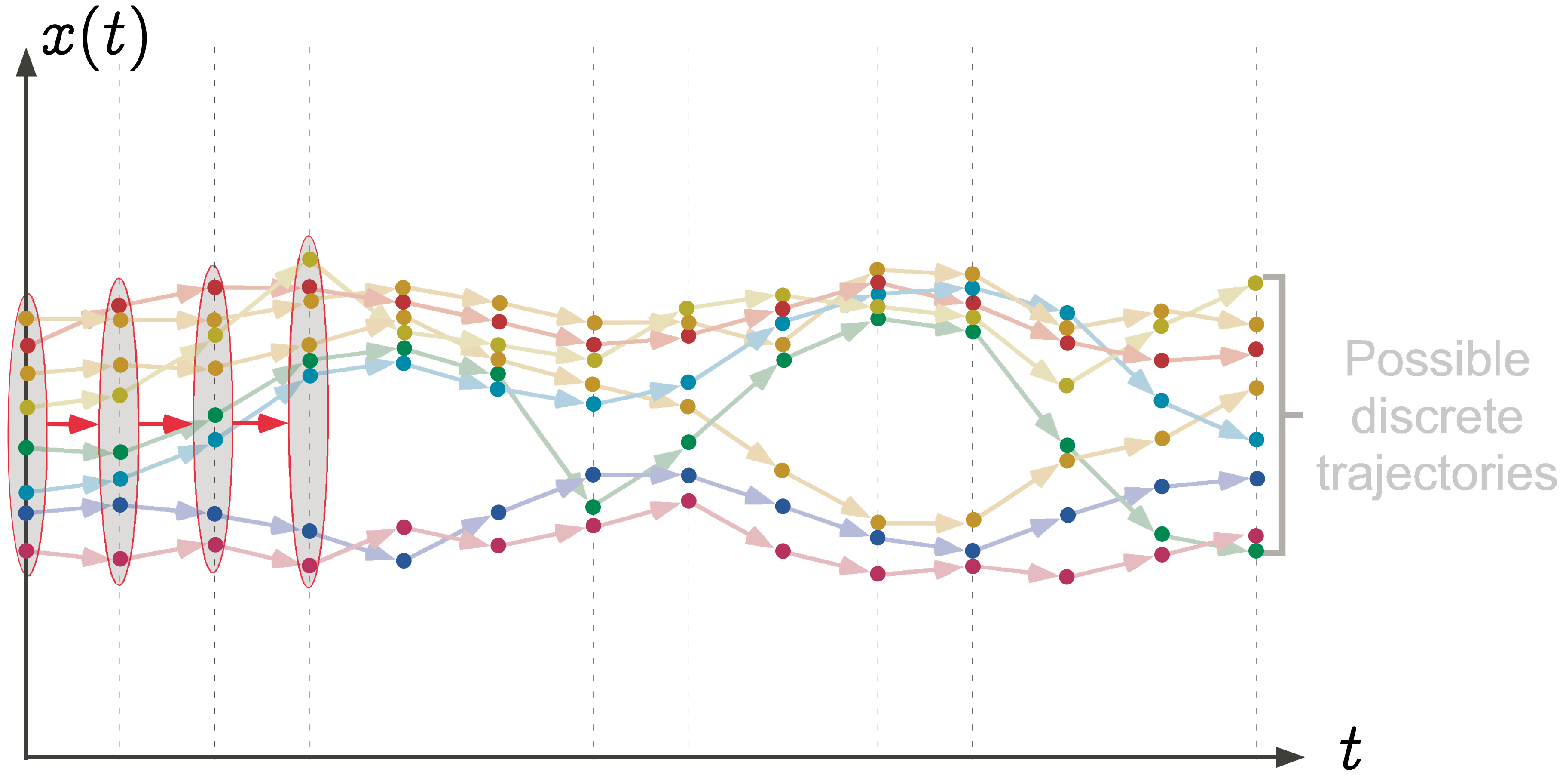
Graphic example: collecting semantics in fixpoint form



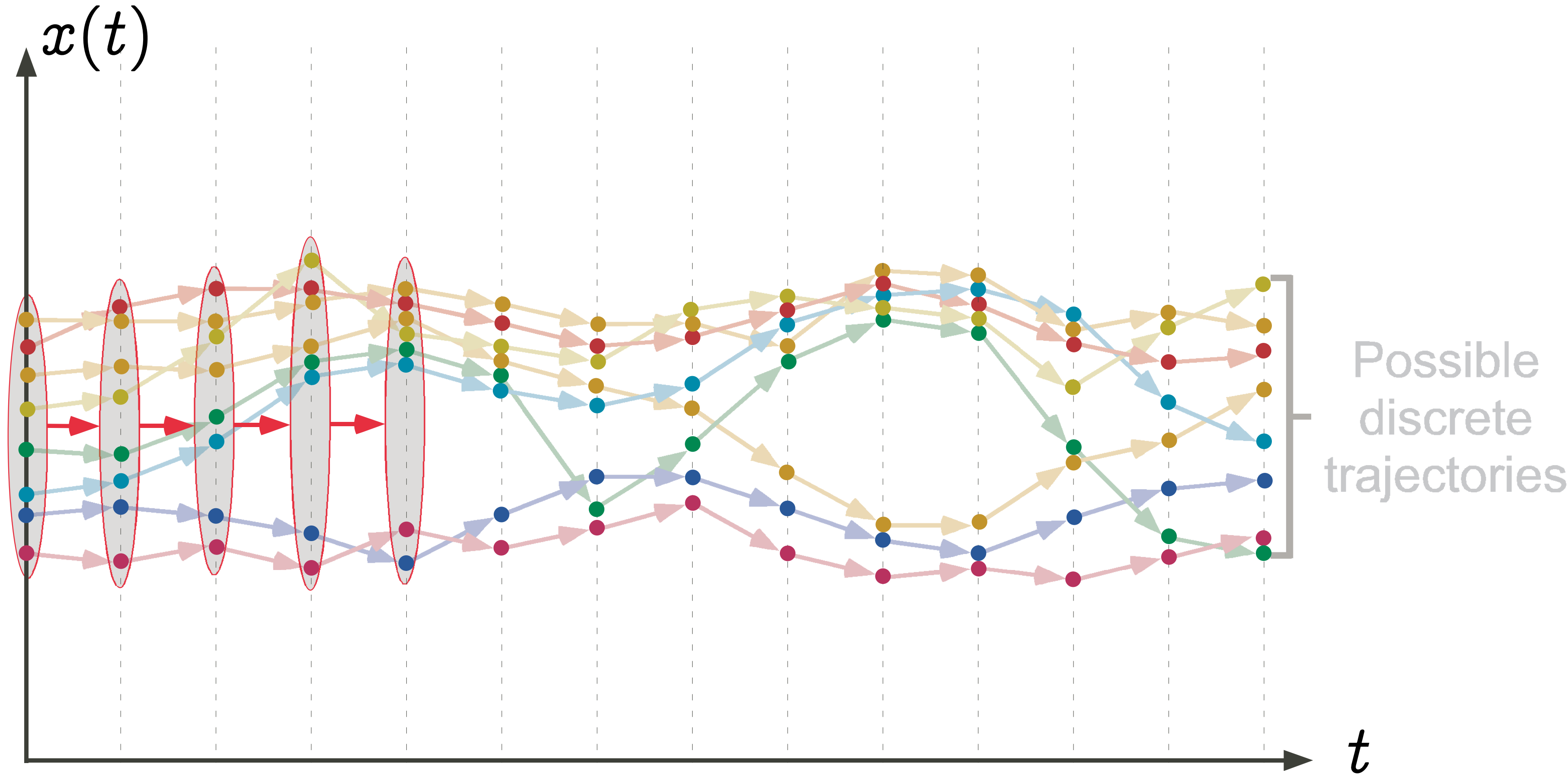
Graphic example: collecting semantics in fixpoint form



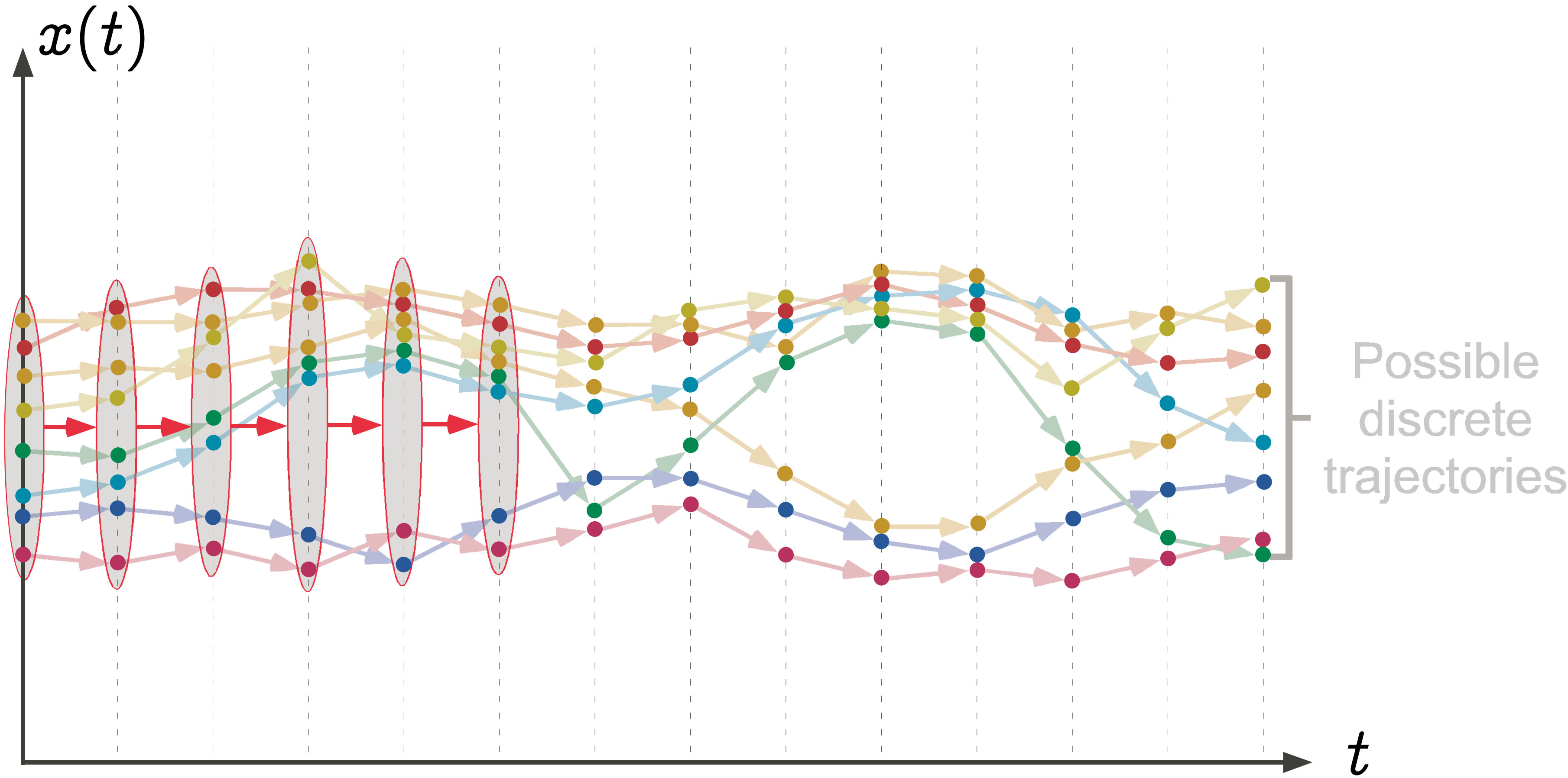
Graphic example: collecting semantics in fixpoint form



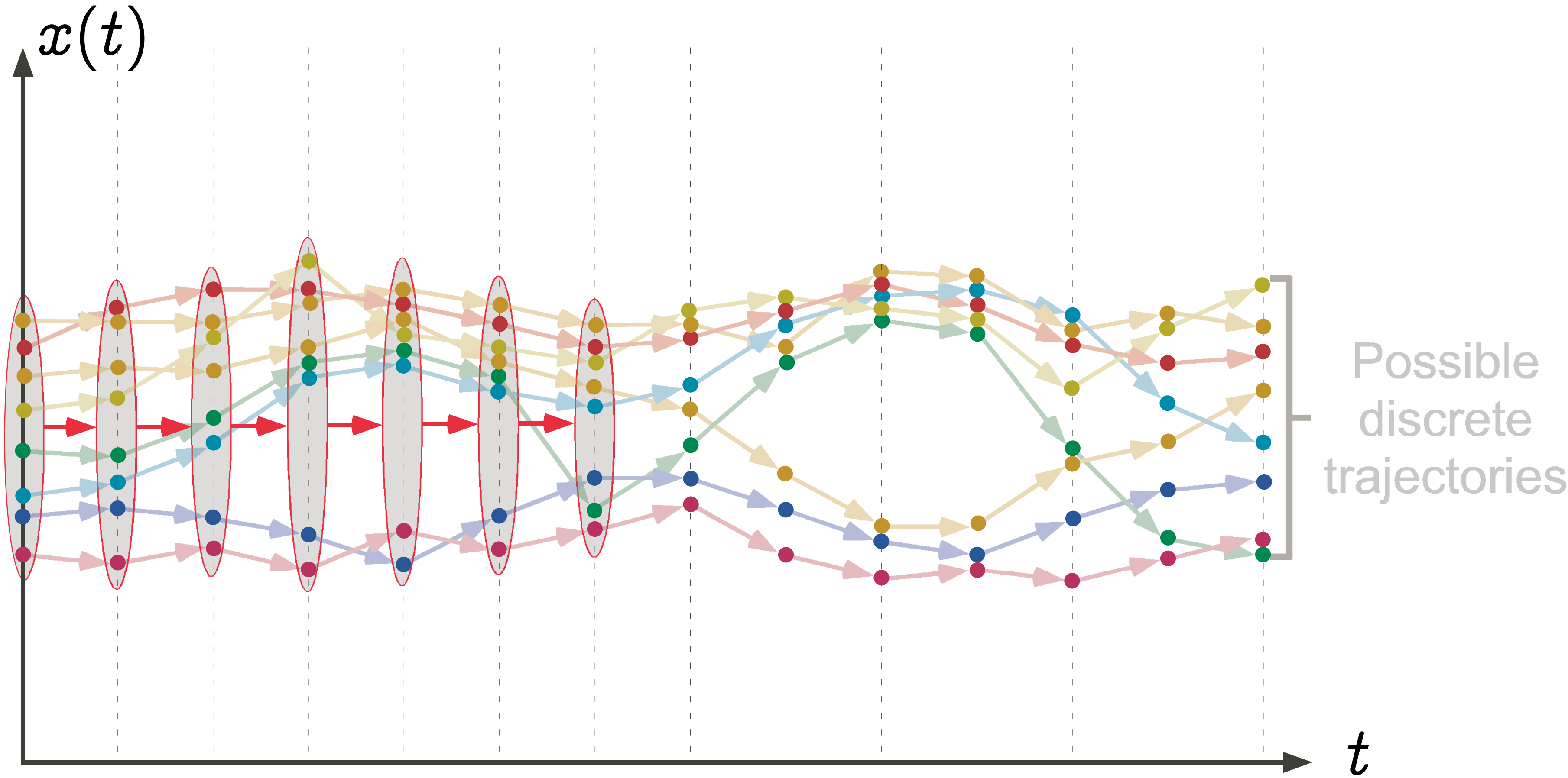
Graphic example: collecting semantics in fixpoint form



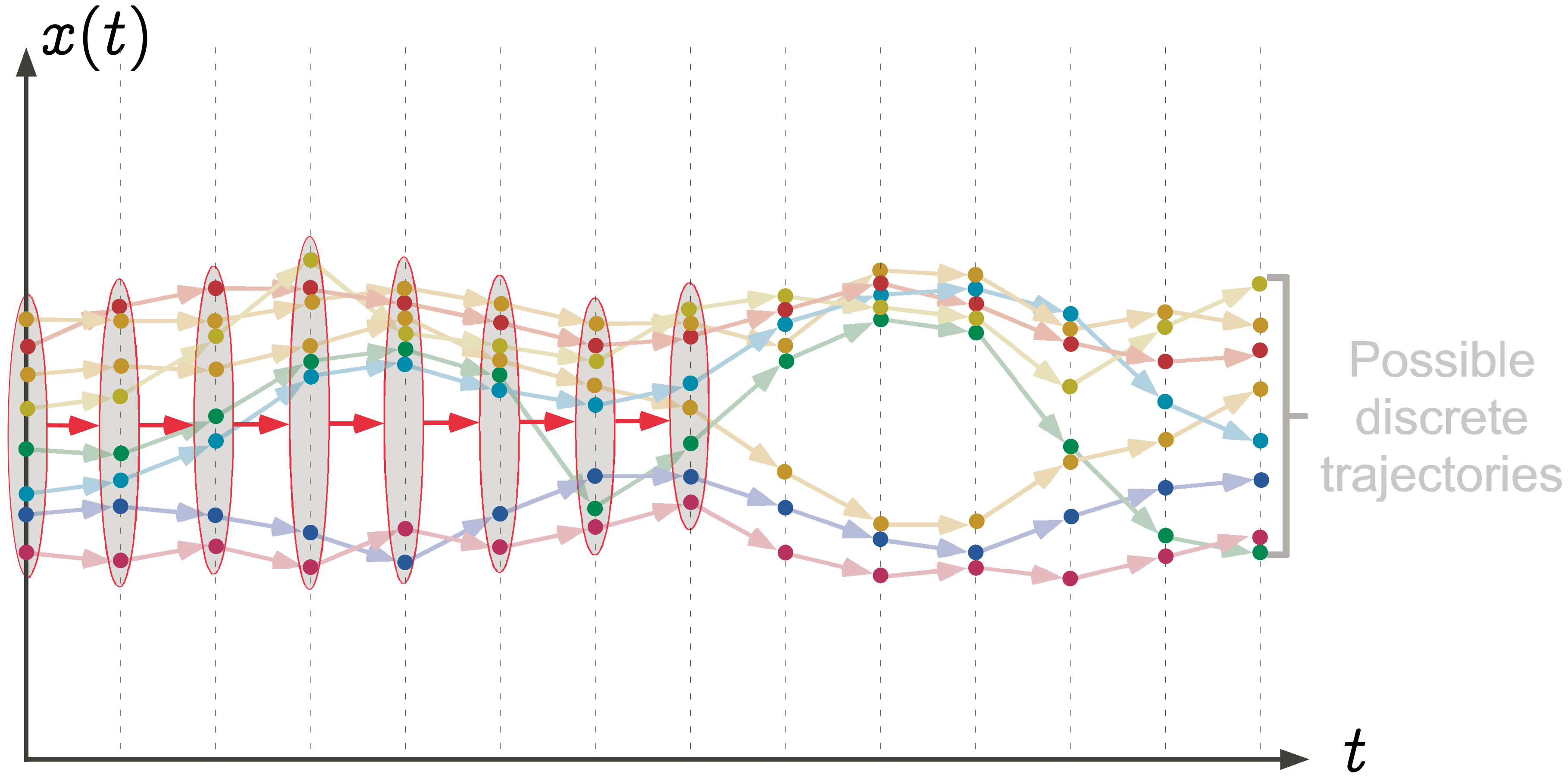
Graphic example: collecting semantics in fixpoint form



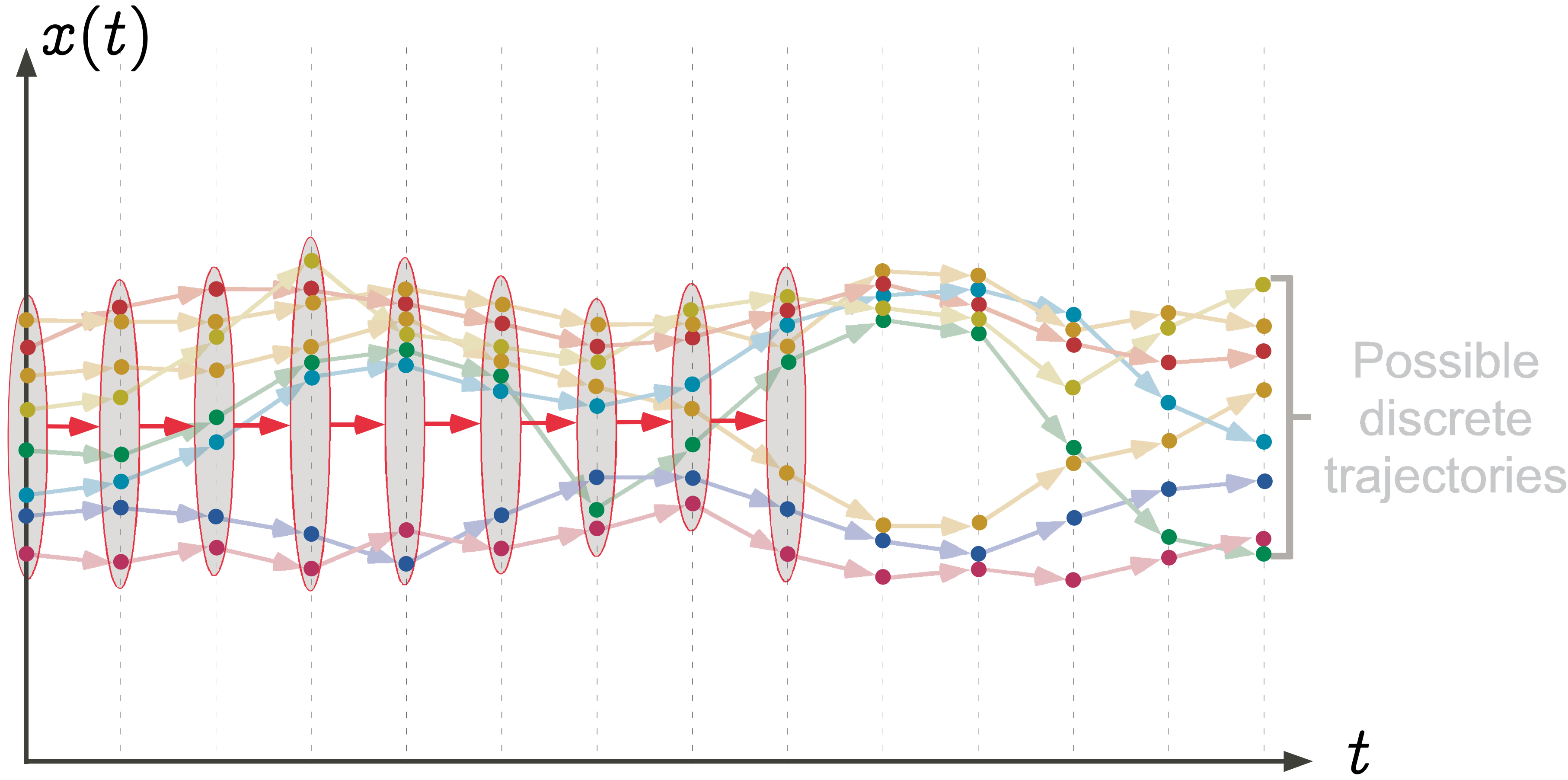
Graphic example: collecting semantics in fixpoint form



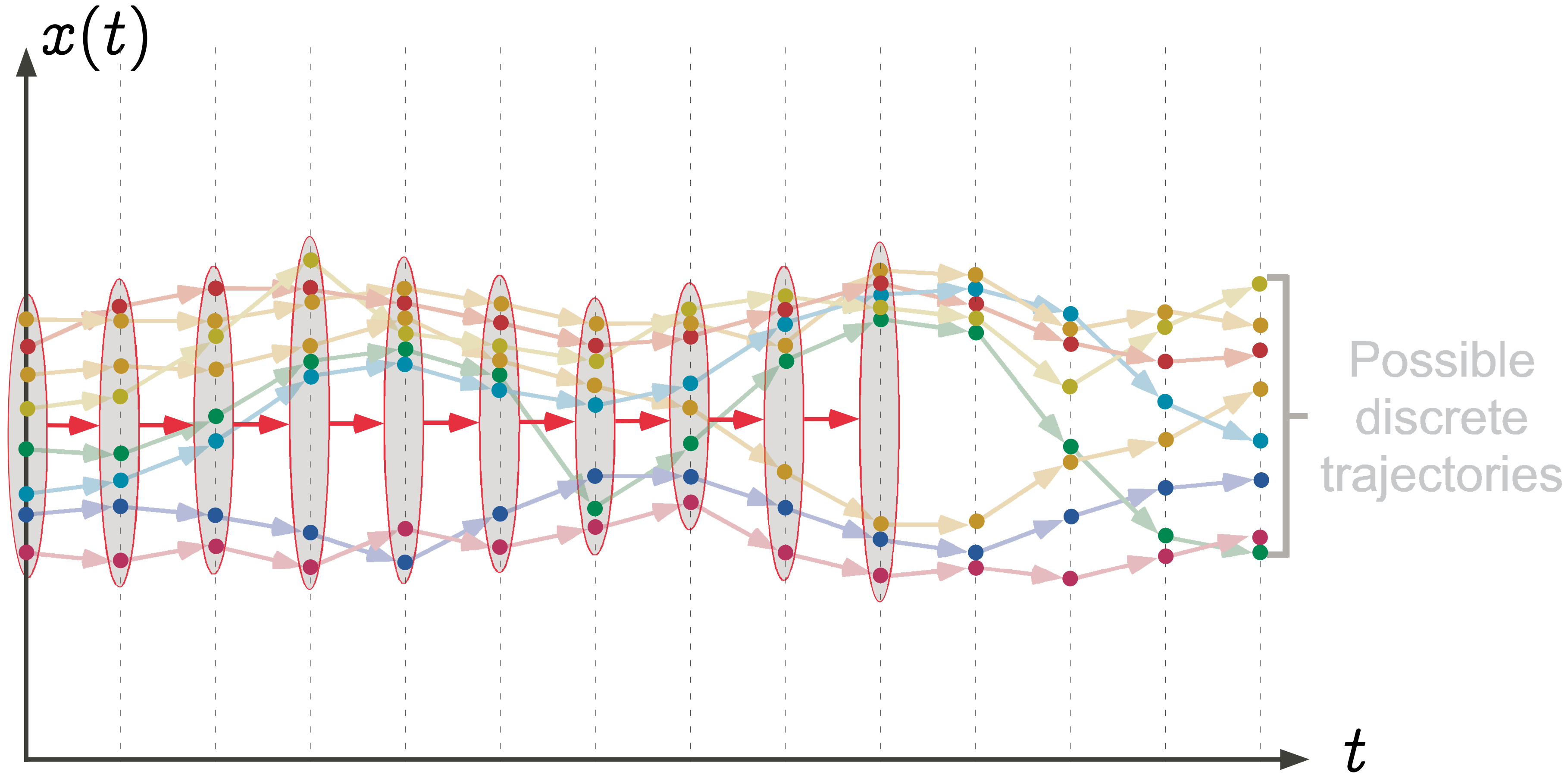
Graphic example: collecting semantics in fixpoint form



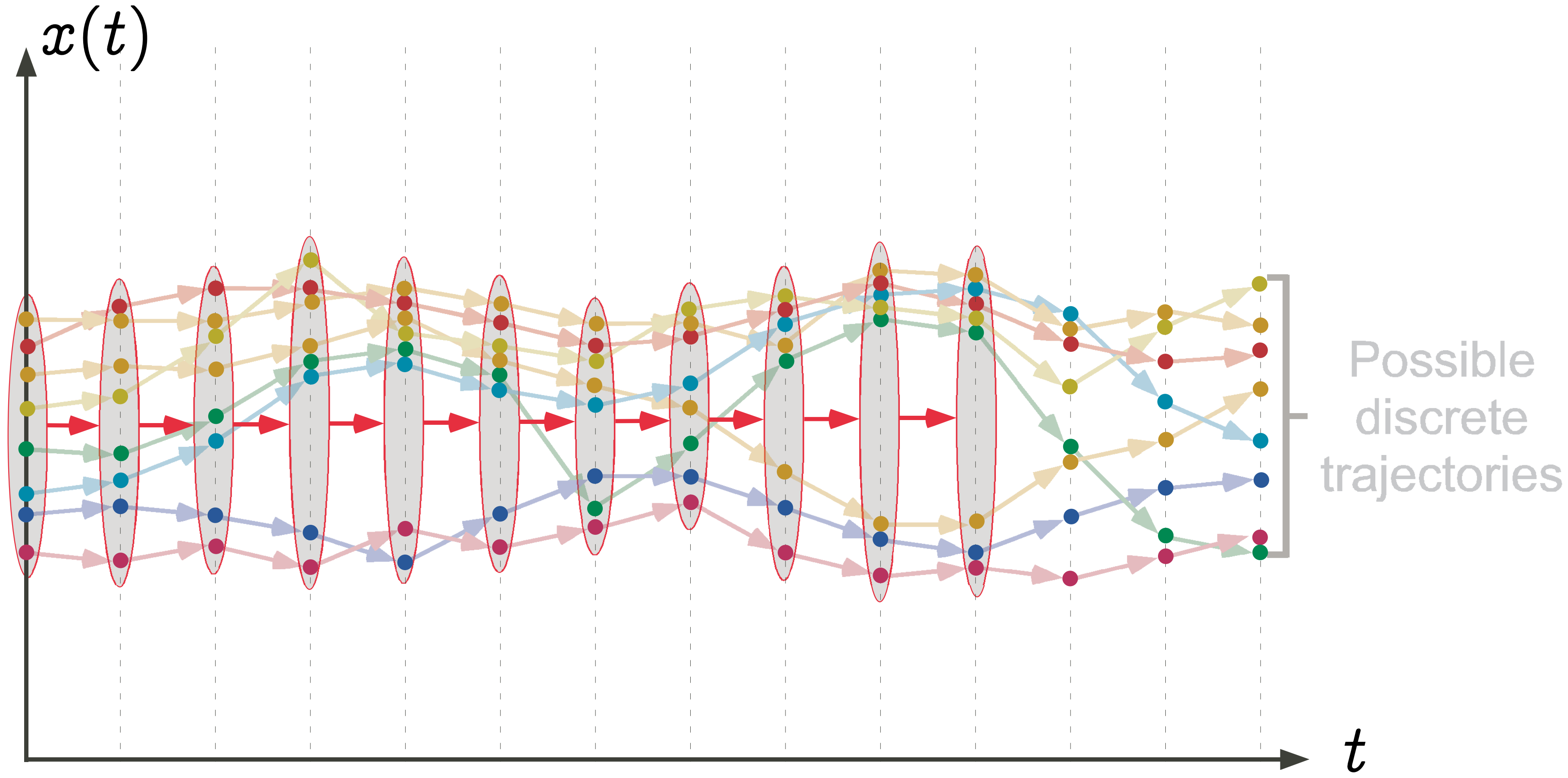
Graphic example: collecting semantics in fixpoint form



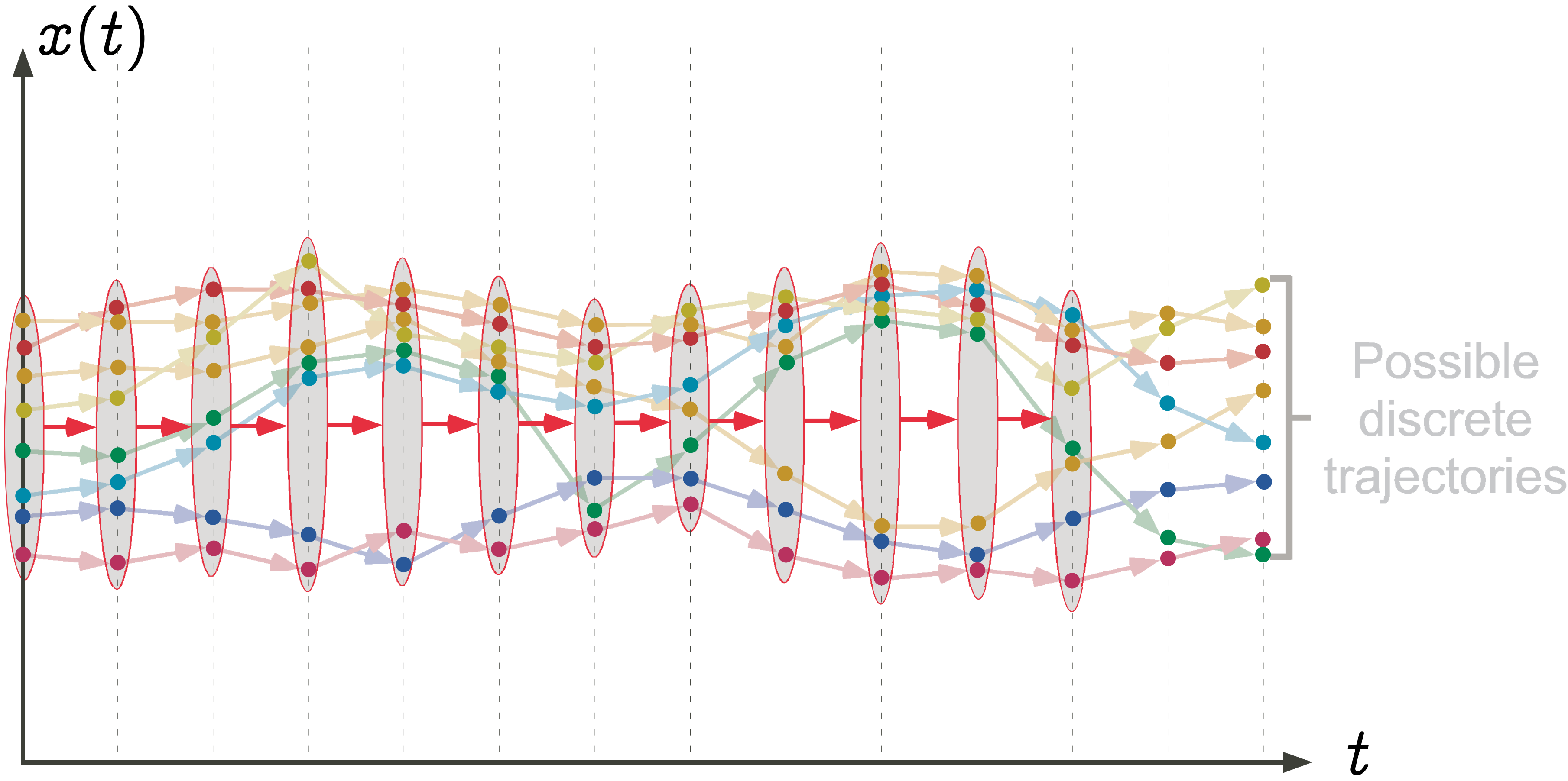
Graphic example: collecting semantics in fixpoint form



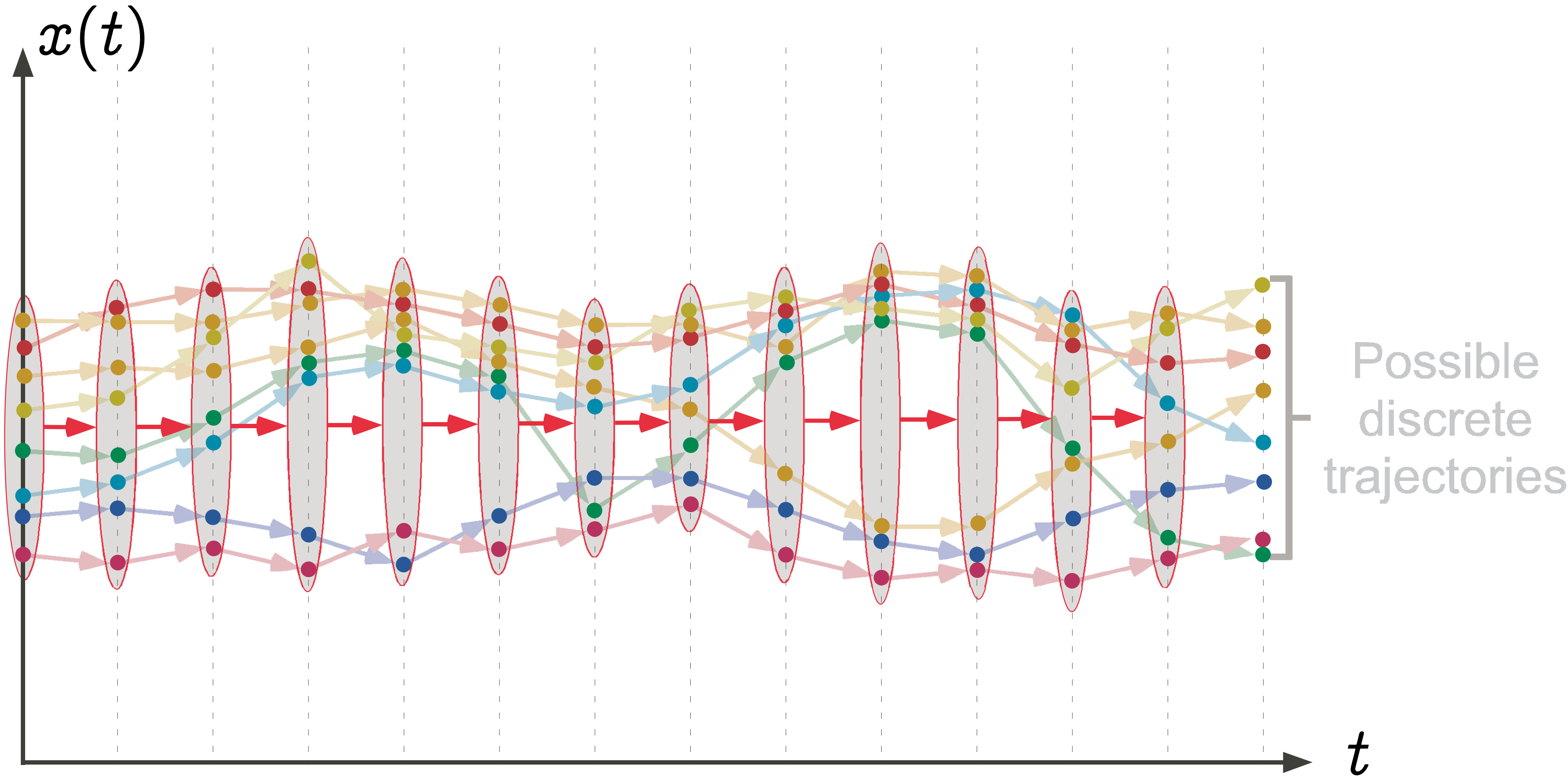
Graphic example: collecting semantics in fixpoint form



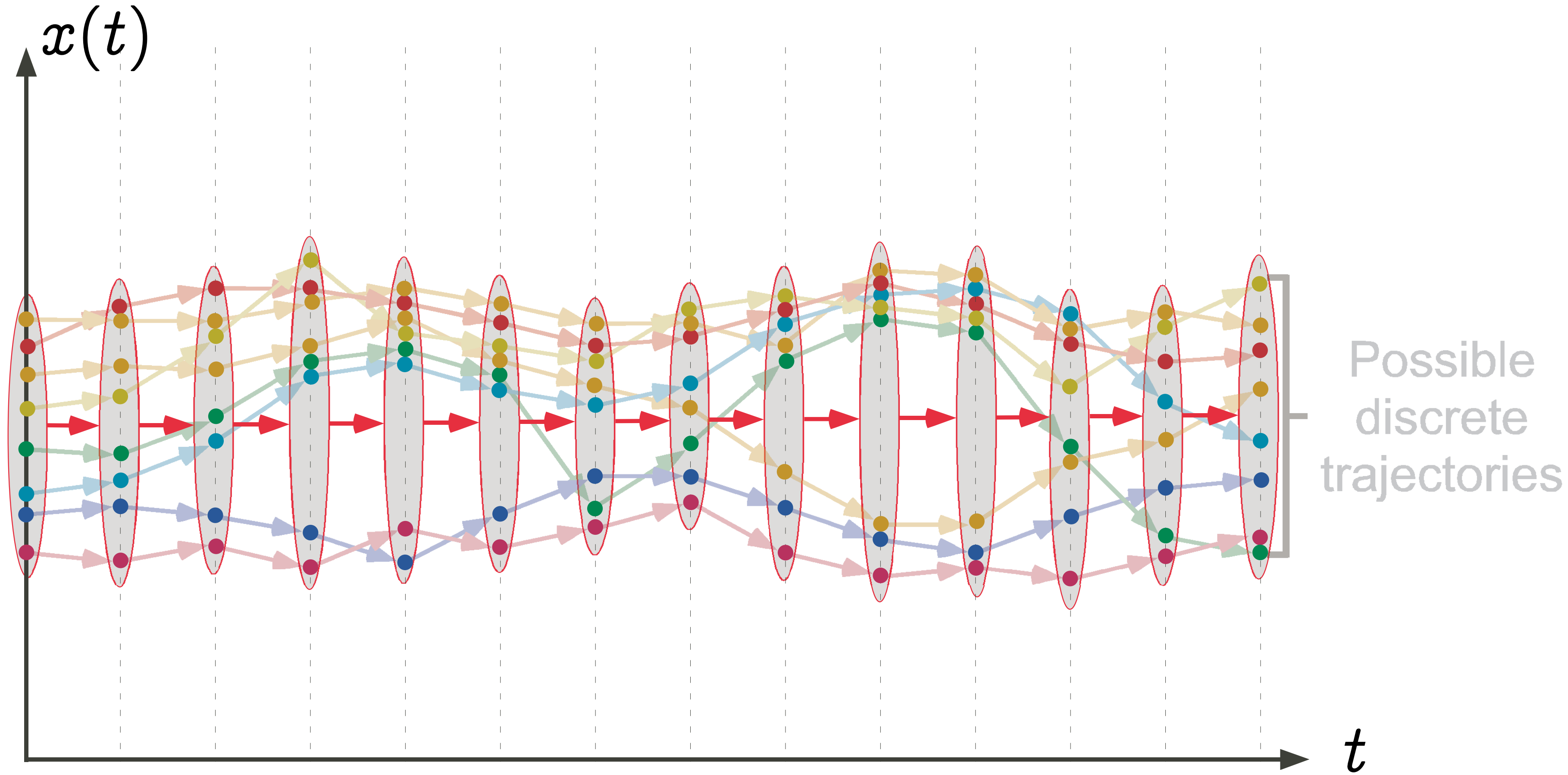
Graphic example: collecting semantics in fixpoint form



Graphic example: collecting semantics in fixpoint form

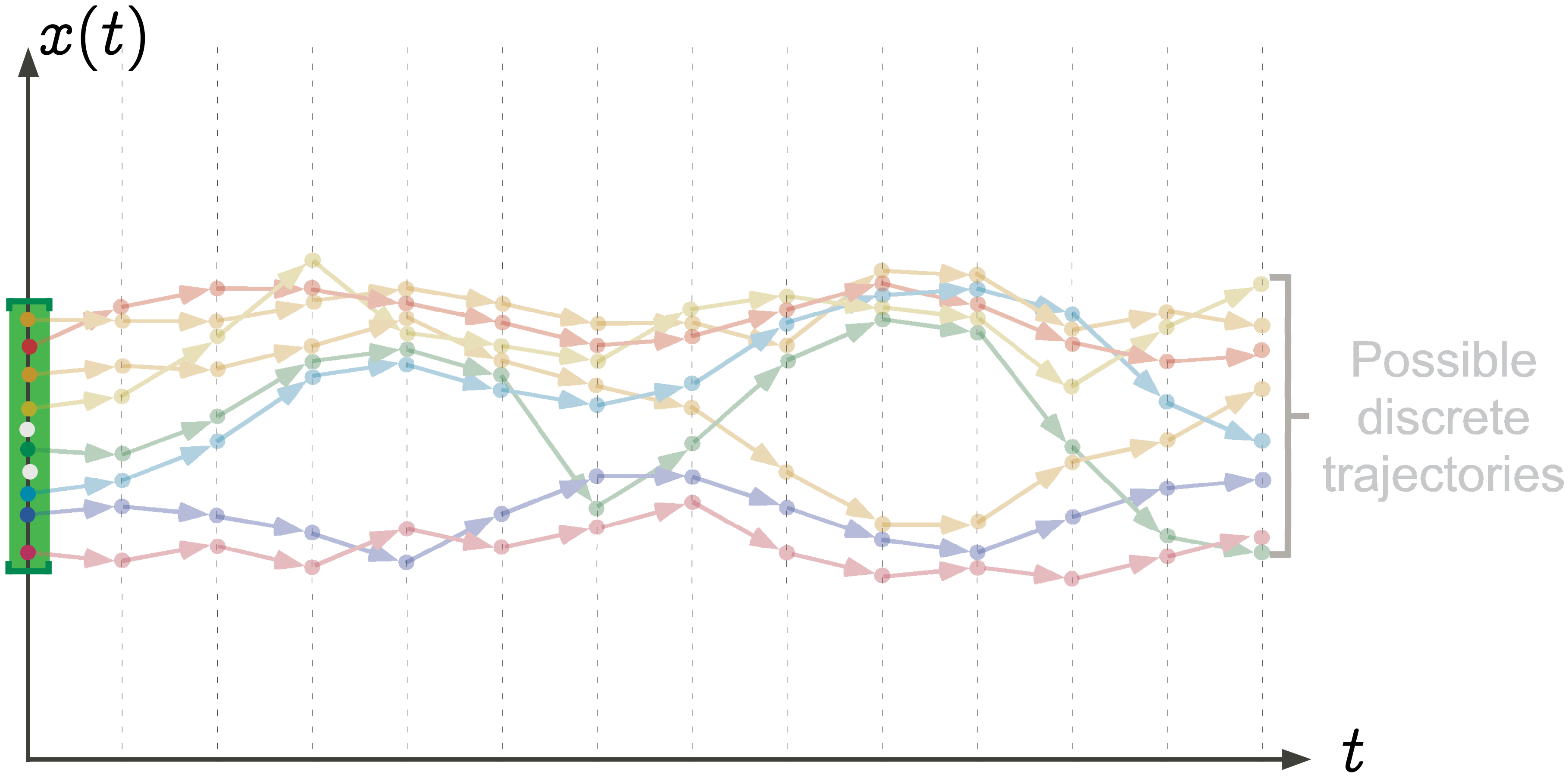


Graphic example: collecting semantics in fixpoint form

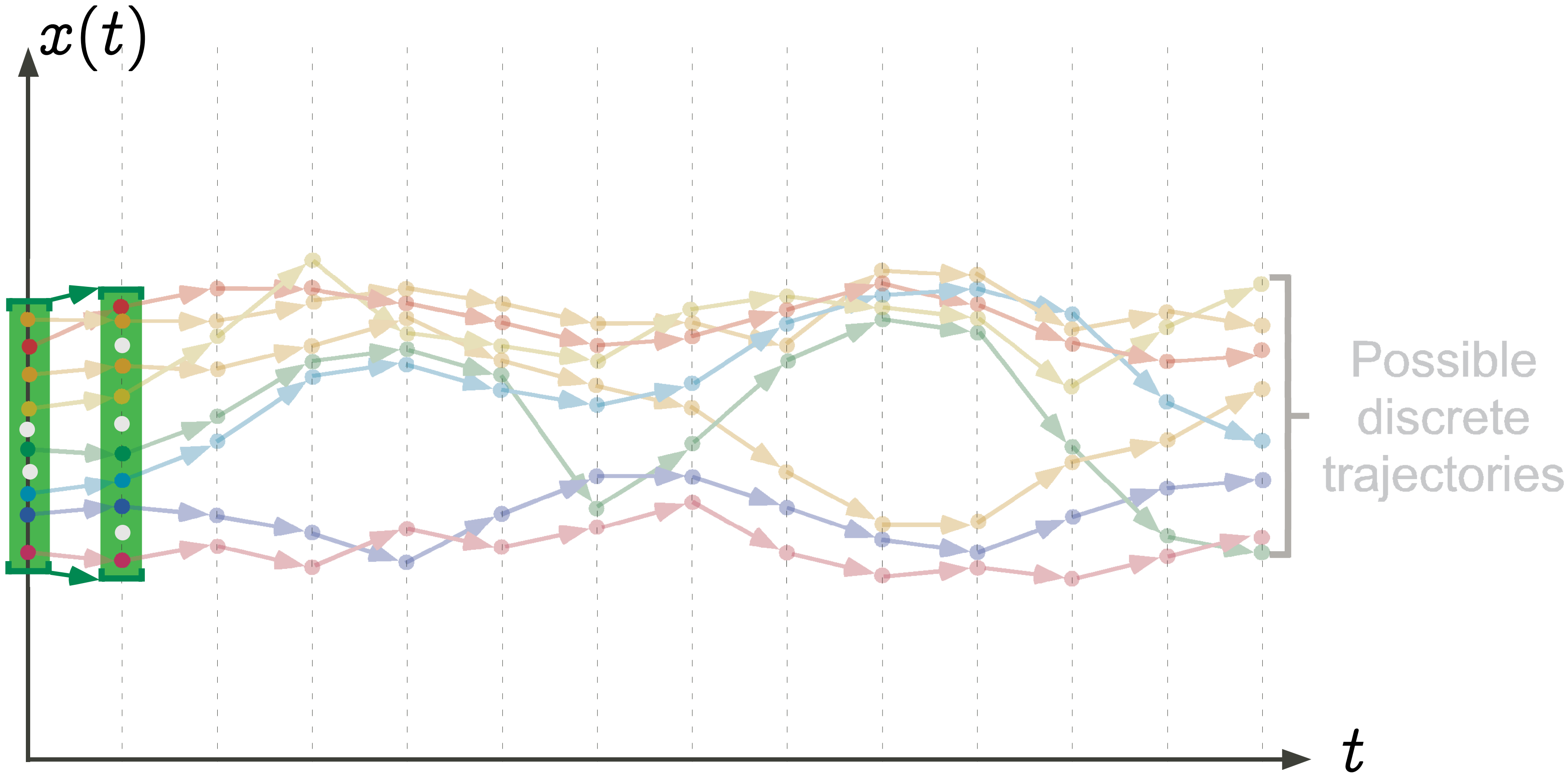


Interval Abstraction (in iterative fixpoint form)

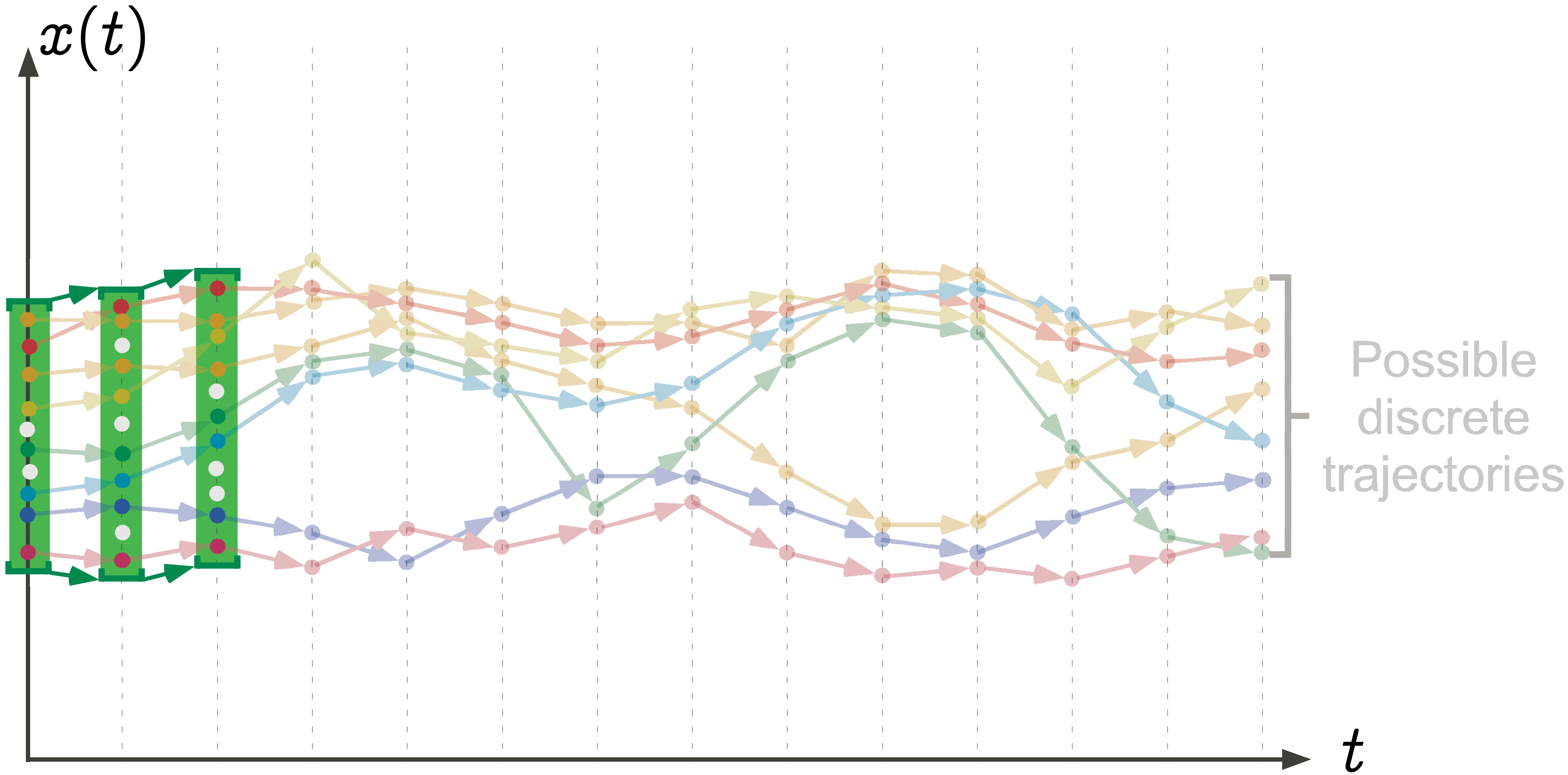
Graphic example: traces of intervals in fixpoint form



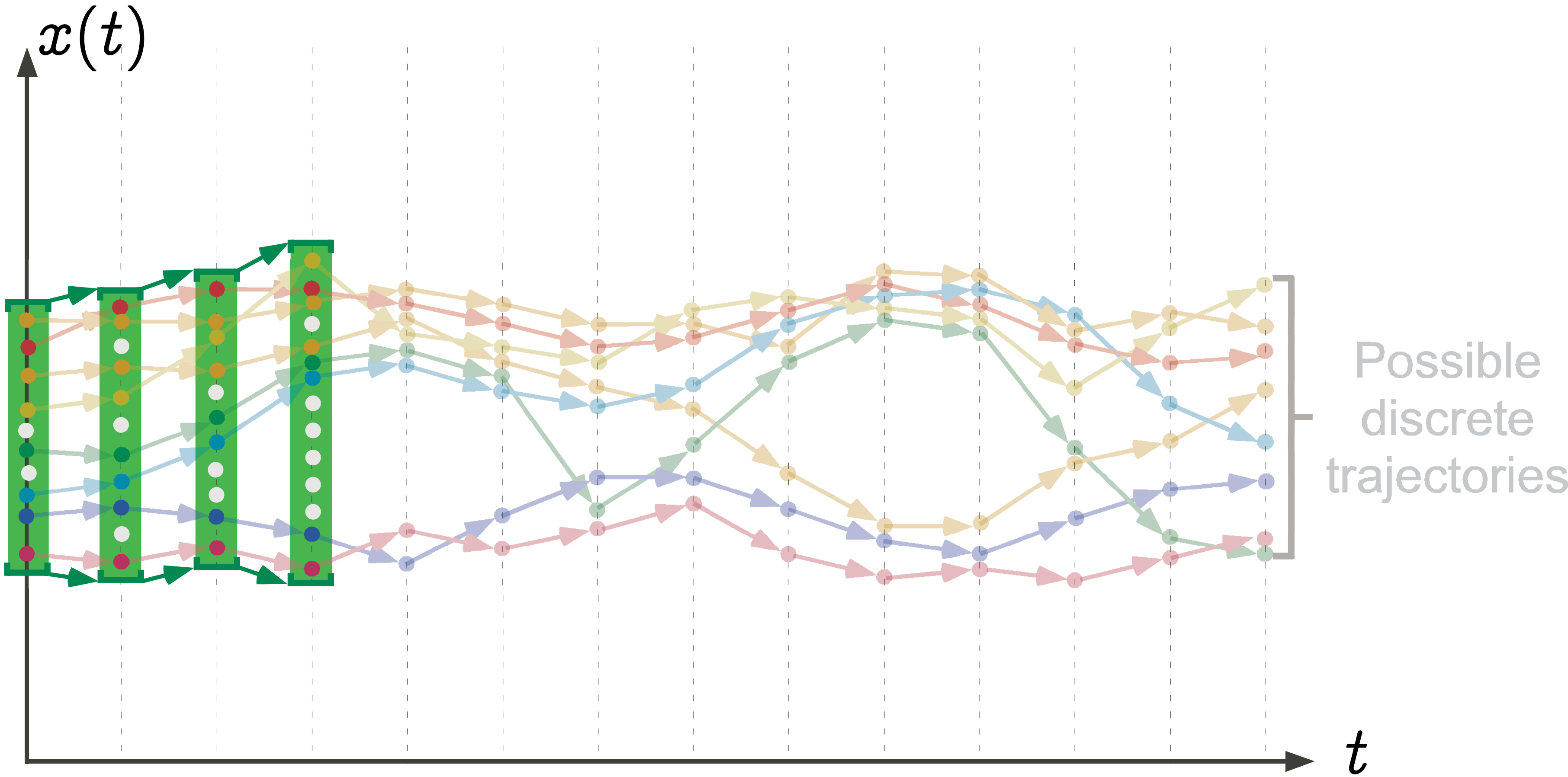
Graphic example: traces of intervals in fixpoint form



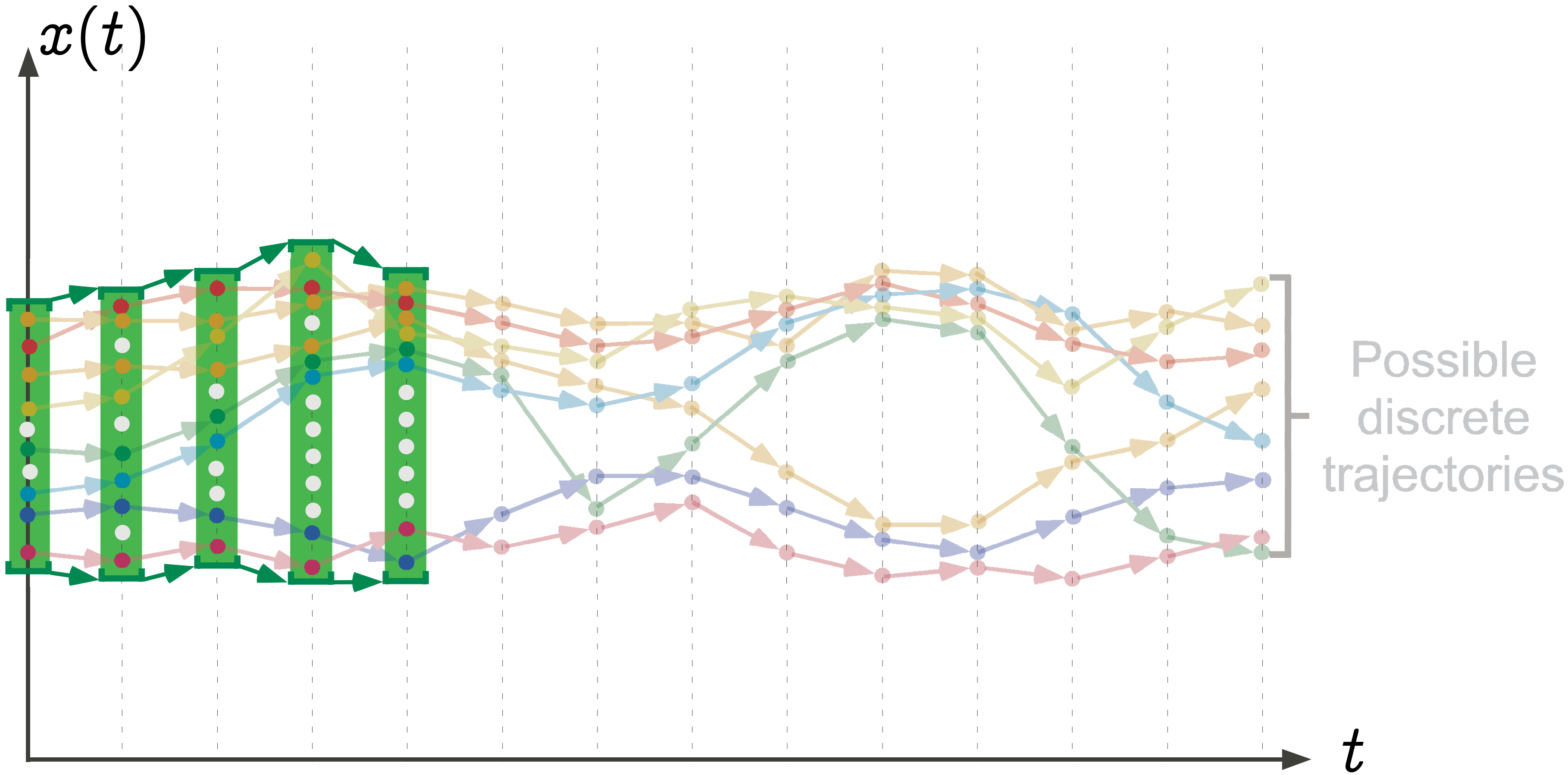
Graphic example: traces of intervals in fixpoint form



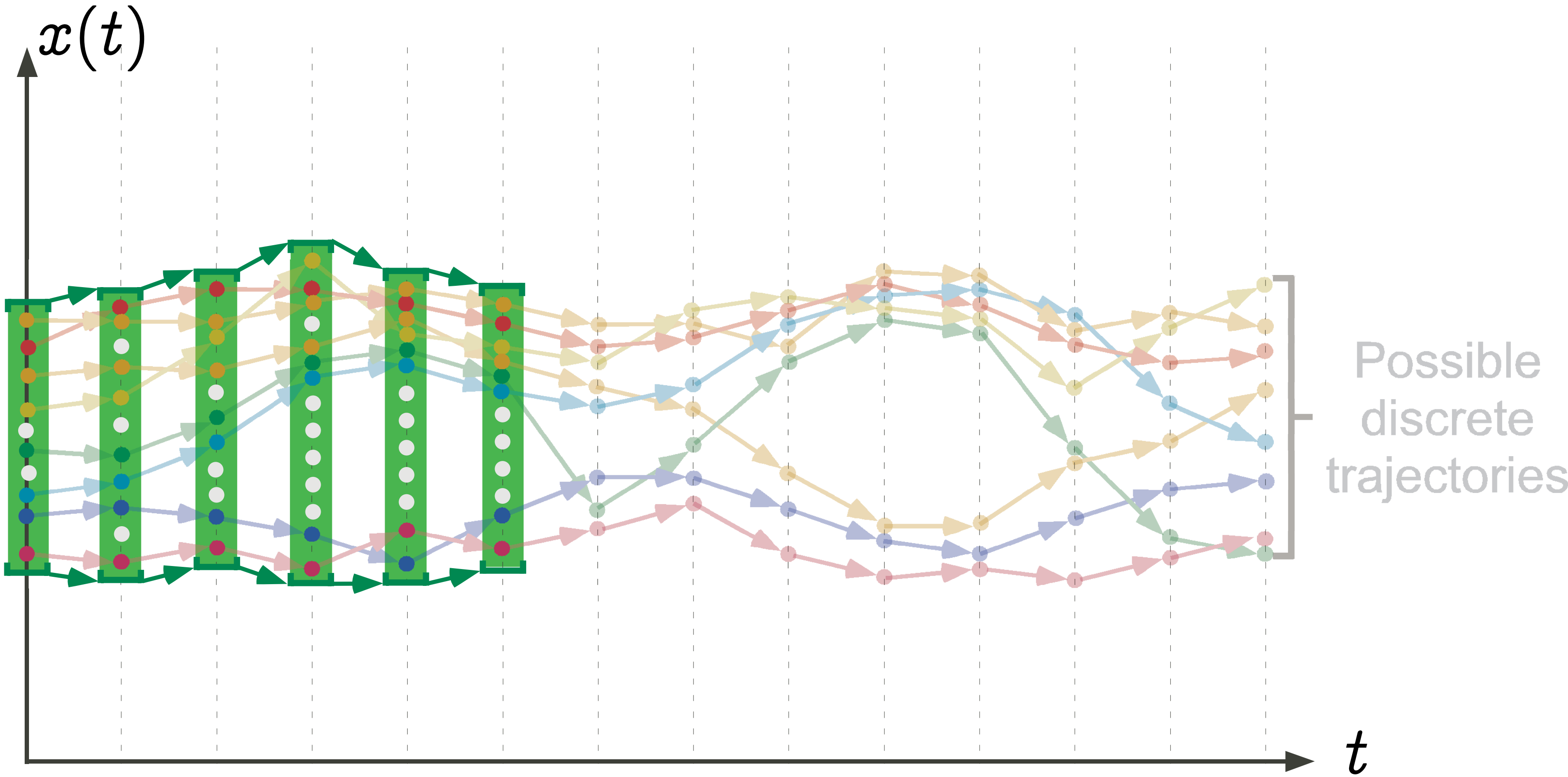
Graphic example: traces of intervals in fixpoint form



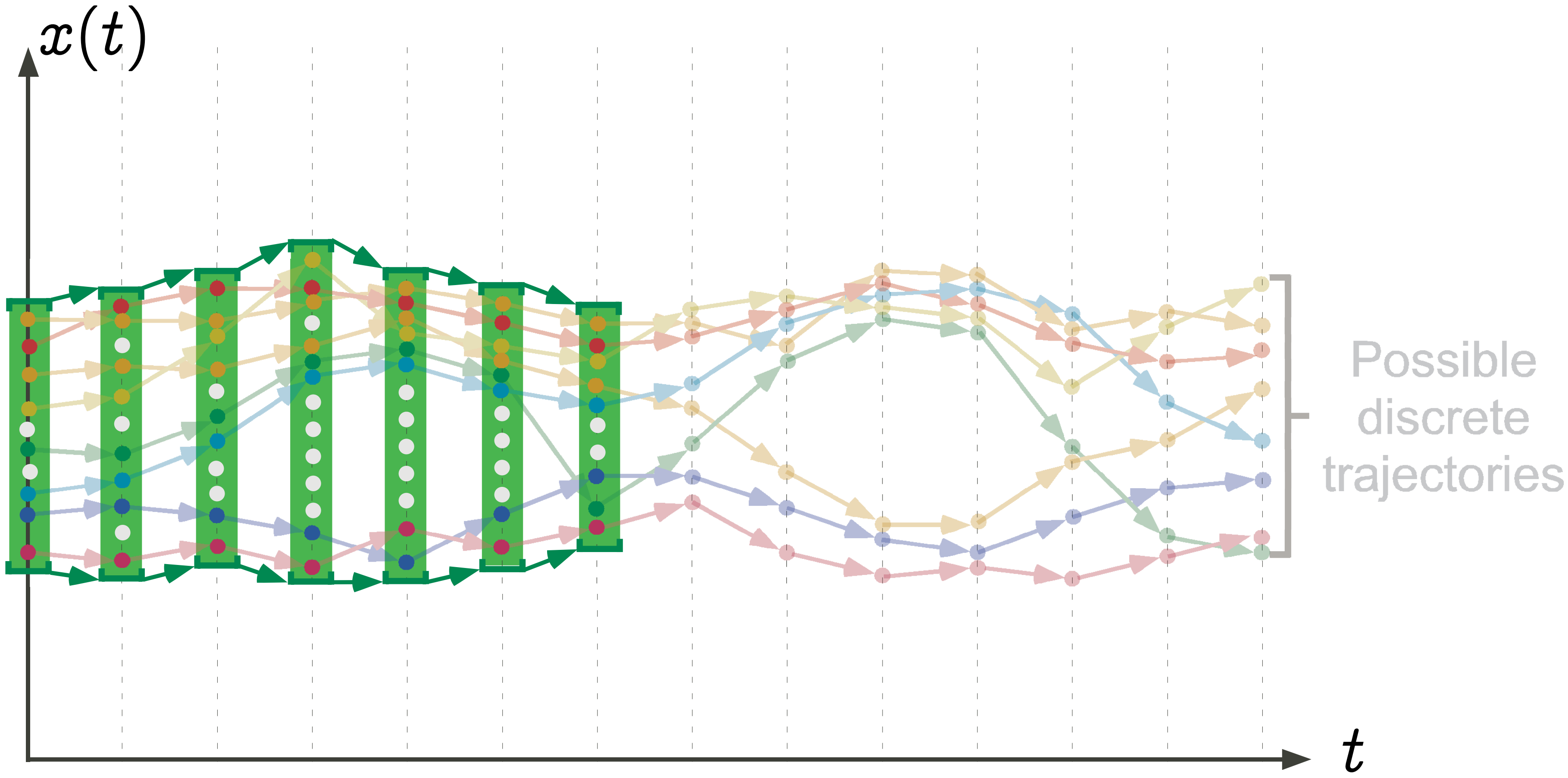
Graphic example: traces of intervals in fixpoint form



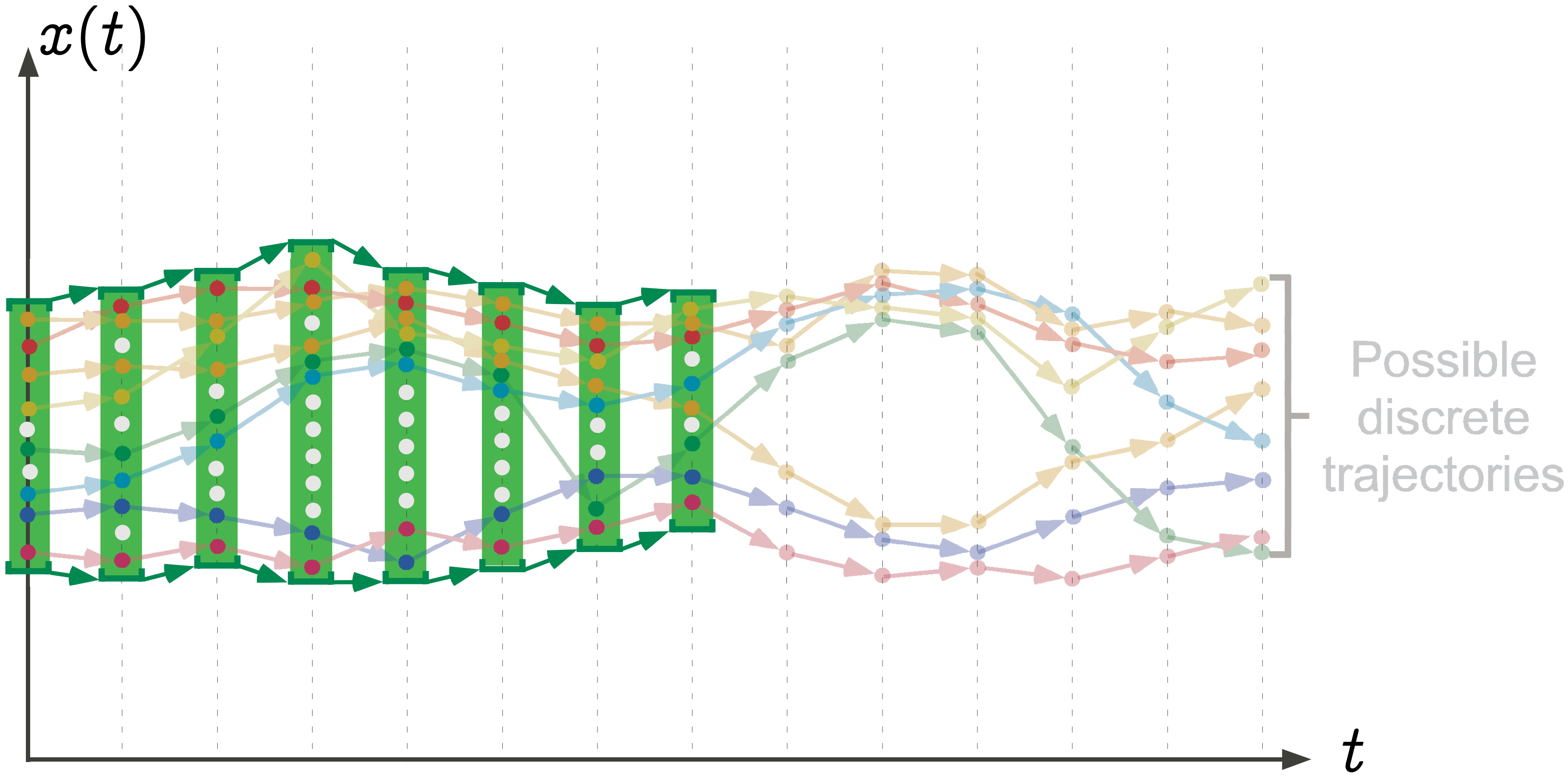
Graphic example: traces of intervals in fixpoint form



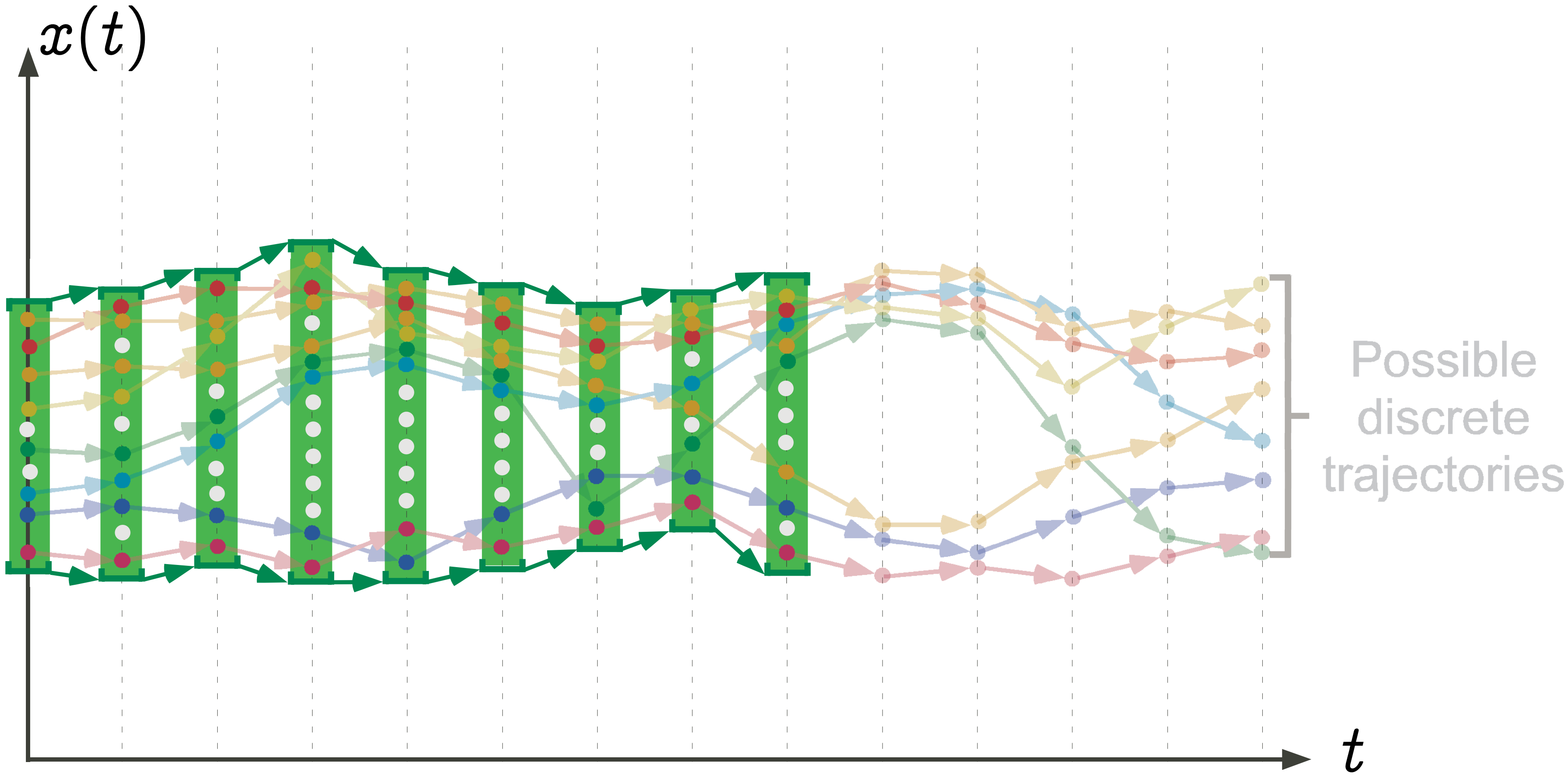
Graphic example: traces of intervals in fixpoint form



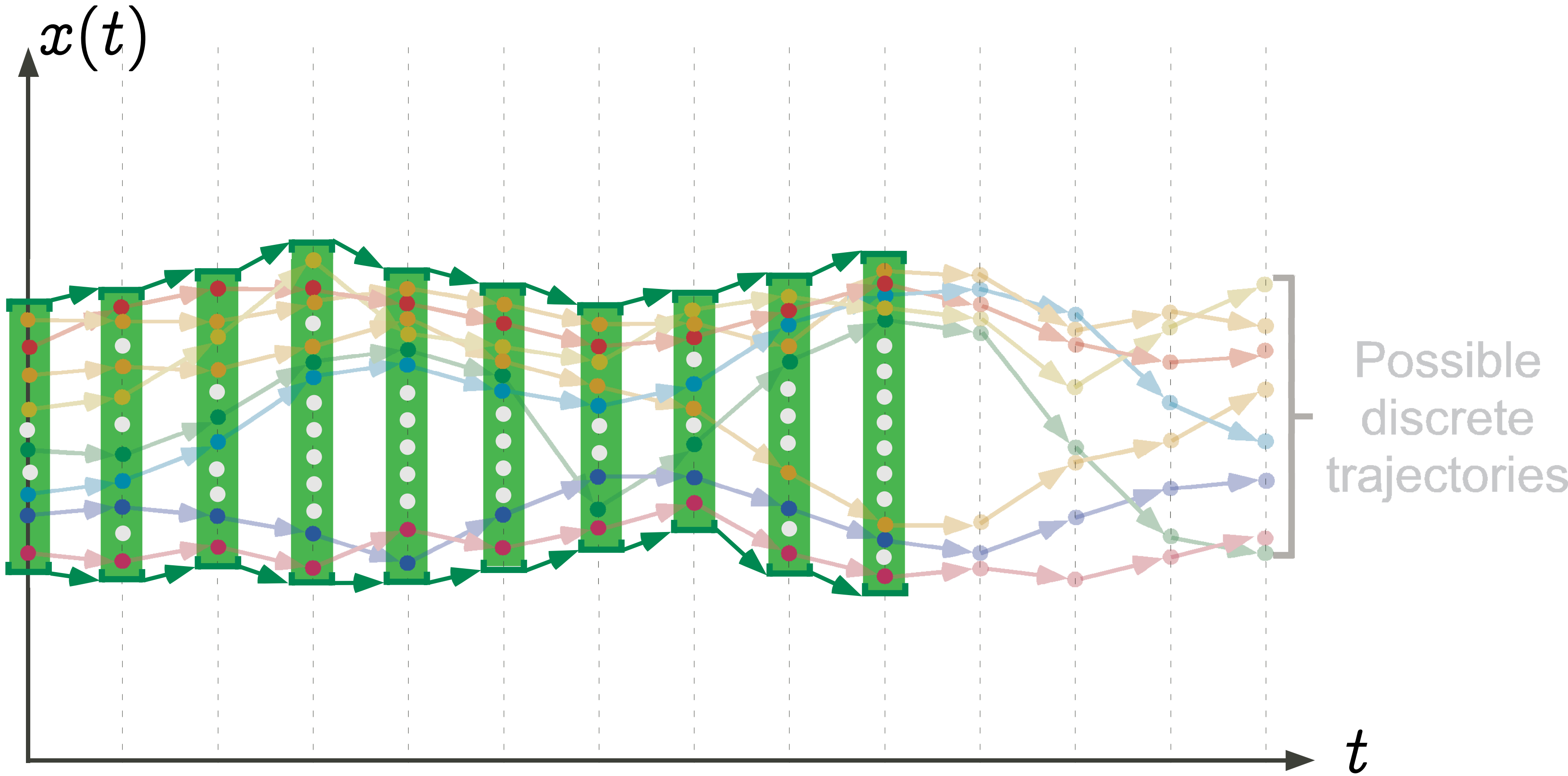
Graphic example: traces of intervals in fixpoint form



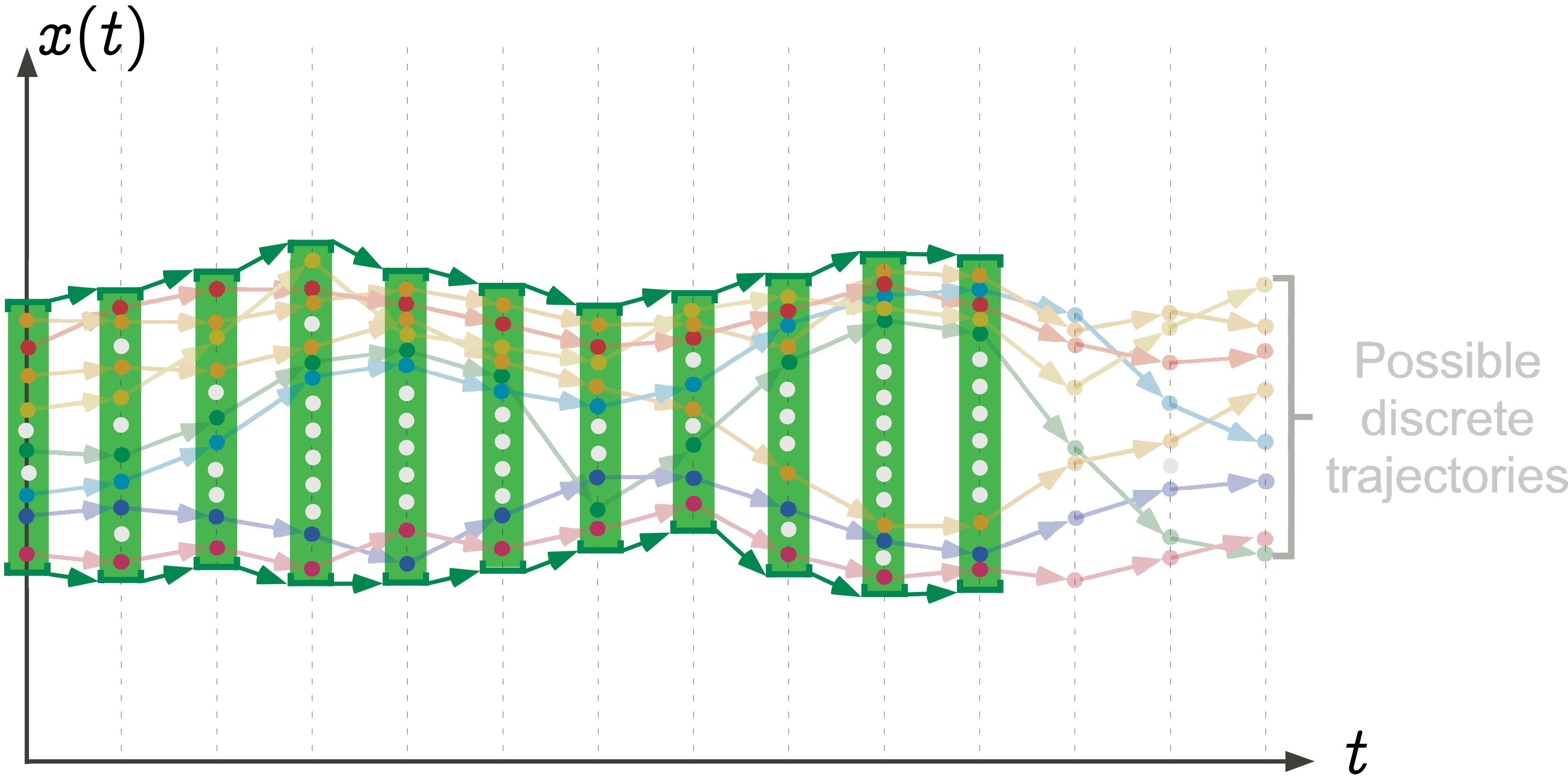
Graphic example: traces of intervals in fixpoint form



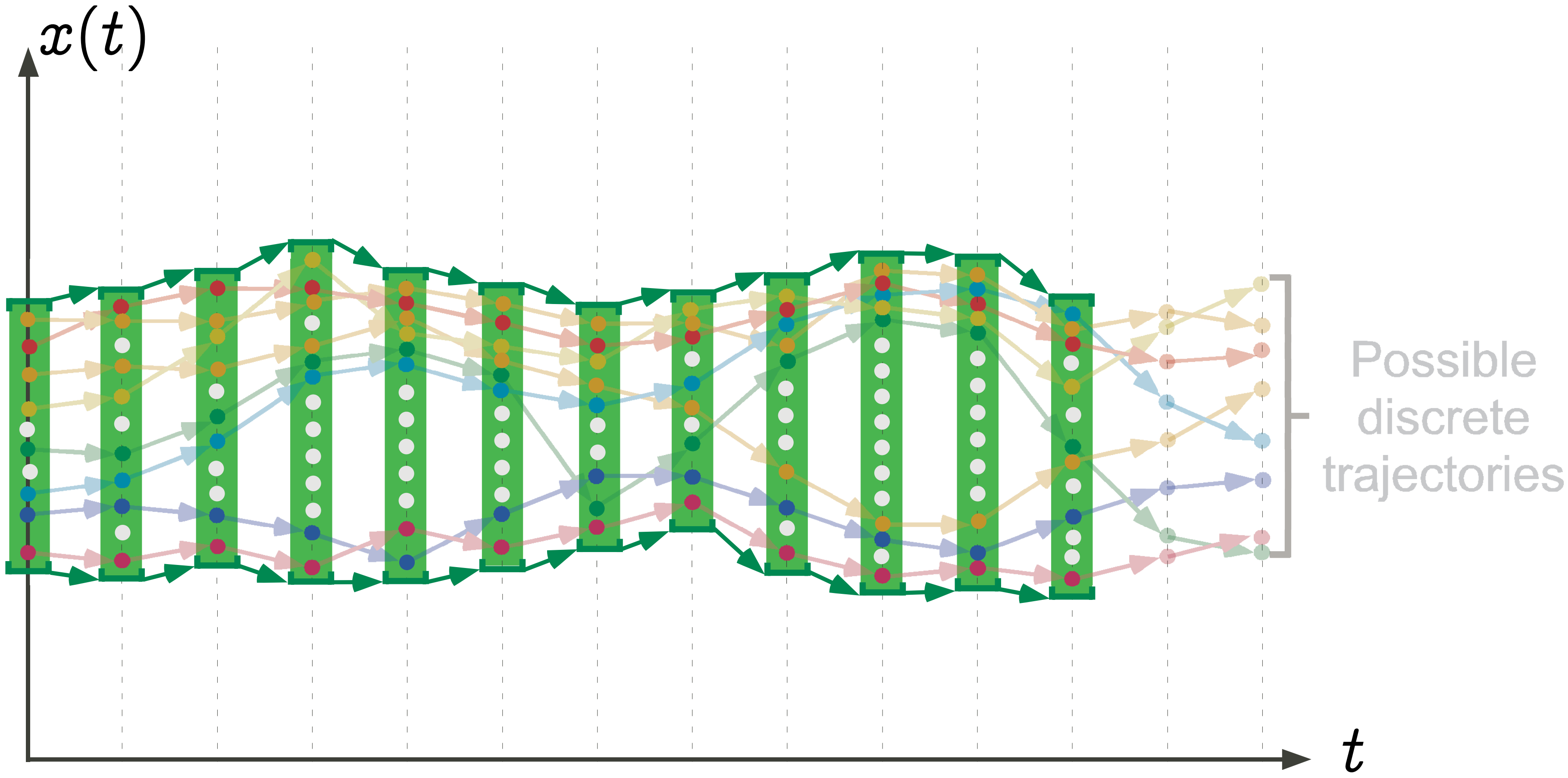
Graphic example: traces of intervals in fixpoint form



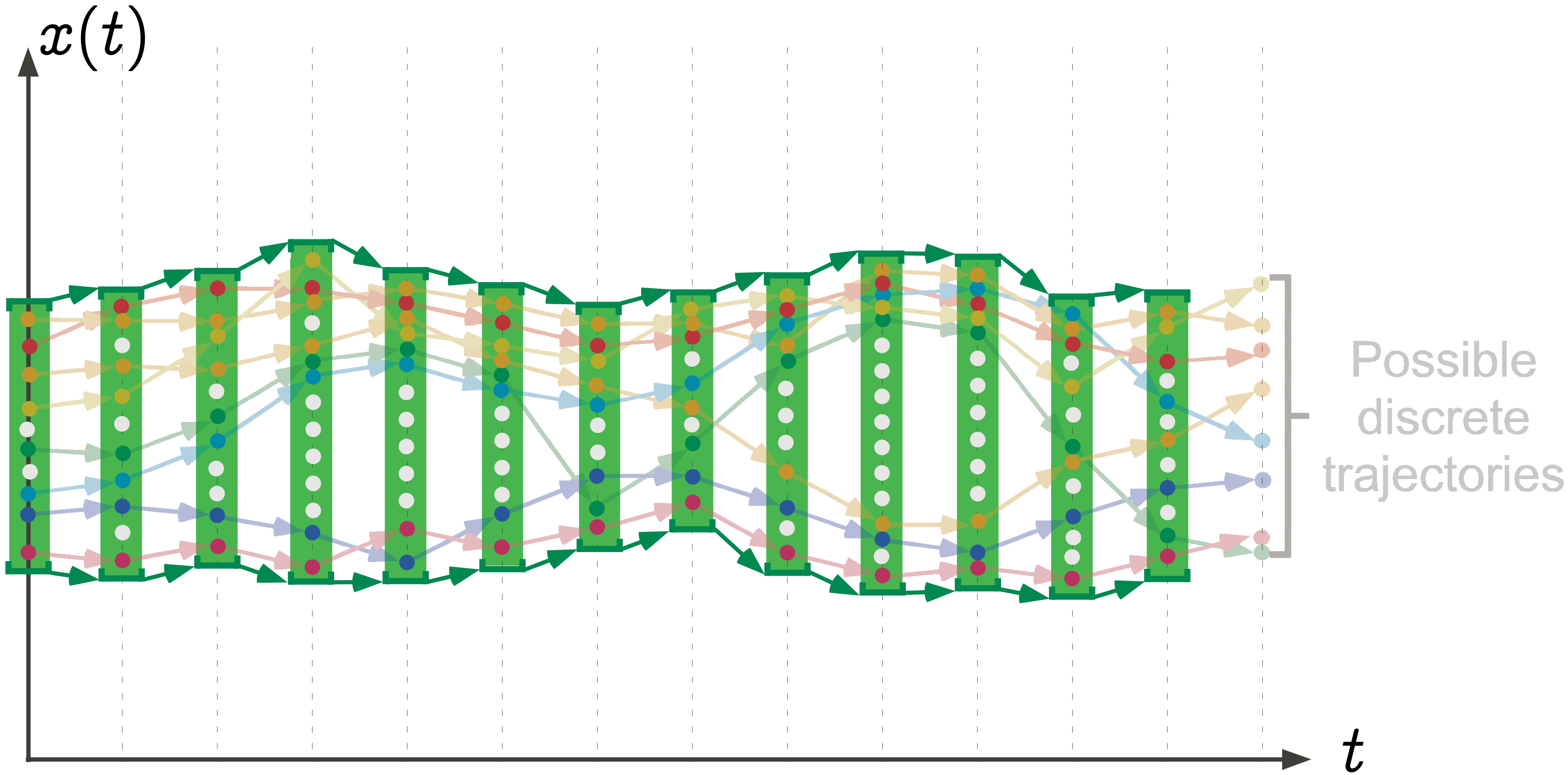
Graphic example: traces of intervals in fixpoint form



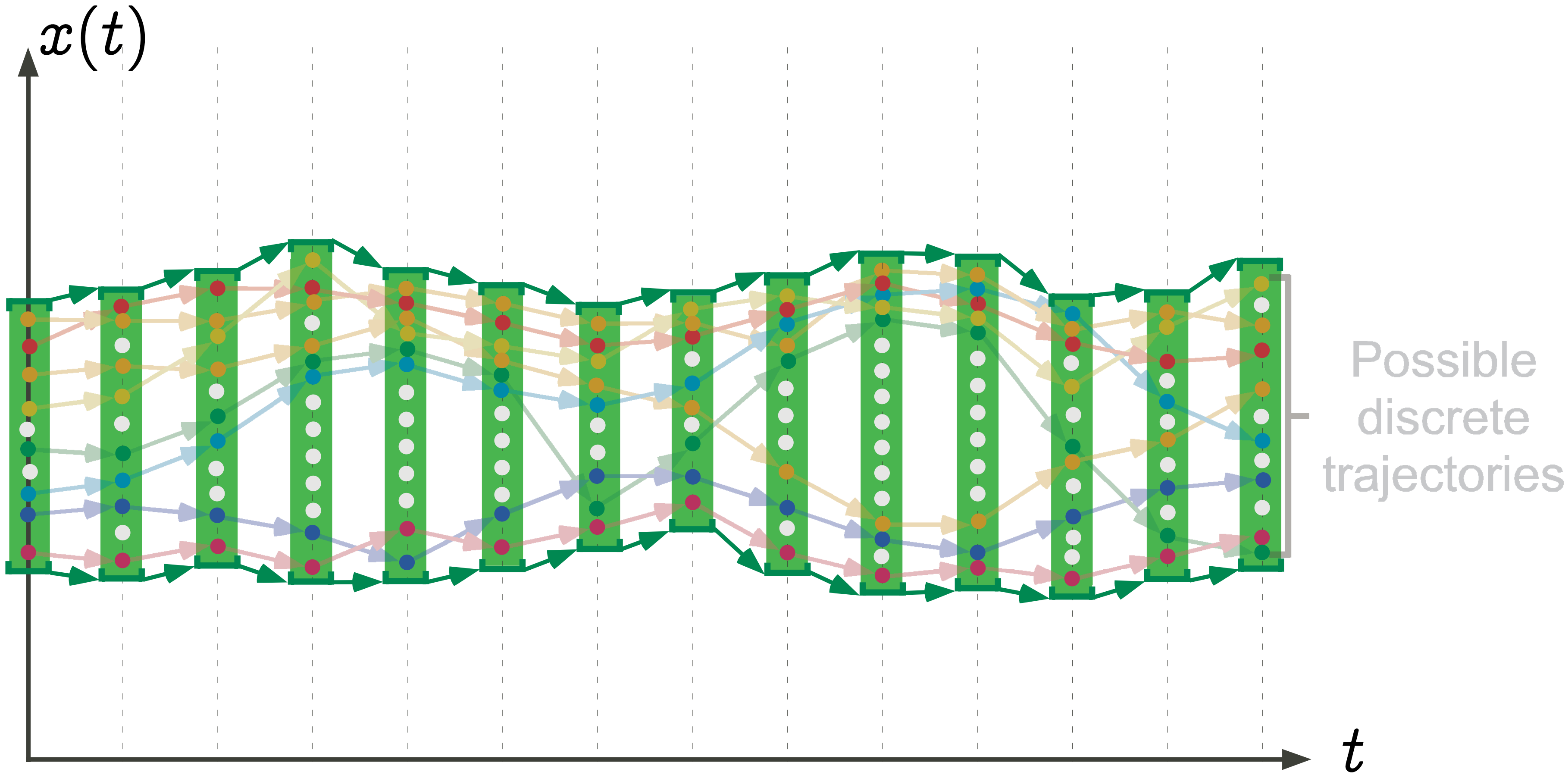
Graphic example: traces of intervals in fixpoint form



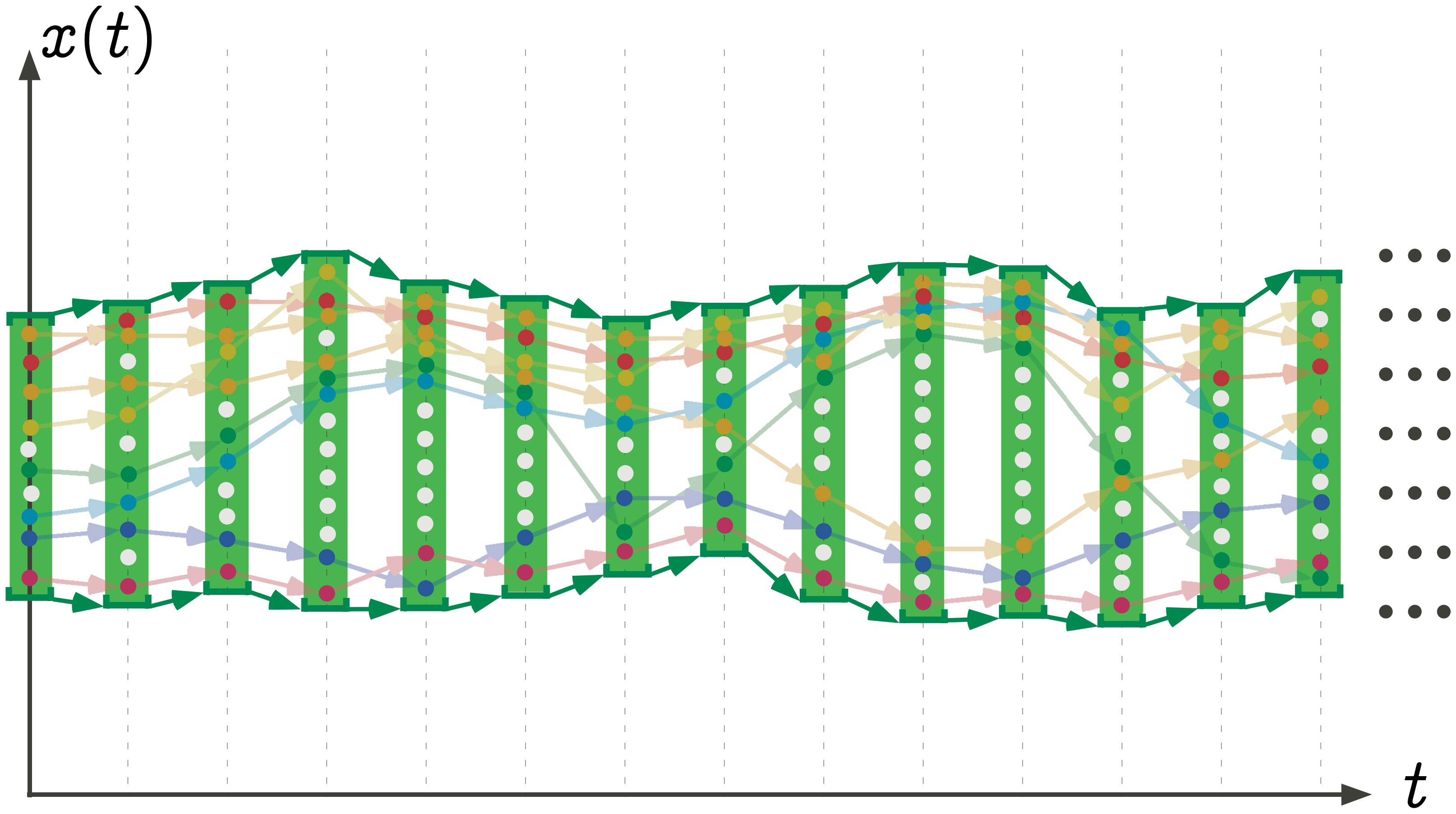
Graphic example: traces of intervals in fixpoint form



Graphic example: traces of intervals in fixpoint form



Graphic example: traces of intervals in fixpoint form

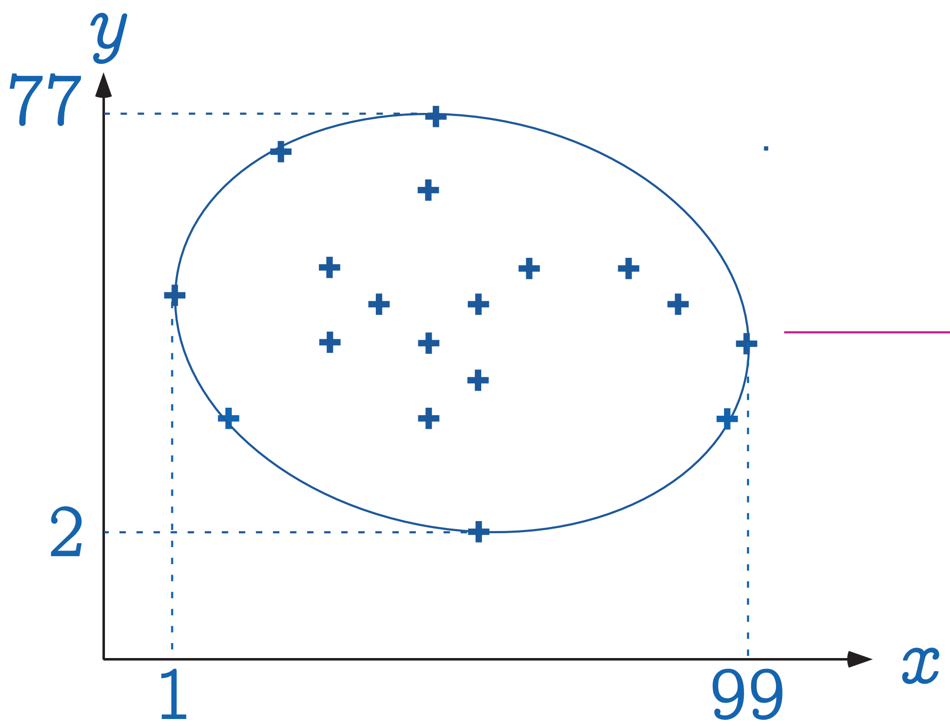


Abstraction by Galois connections

Abstracting sets (i.e. properties)

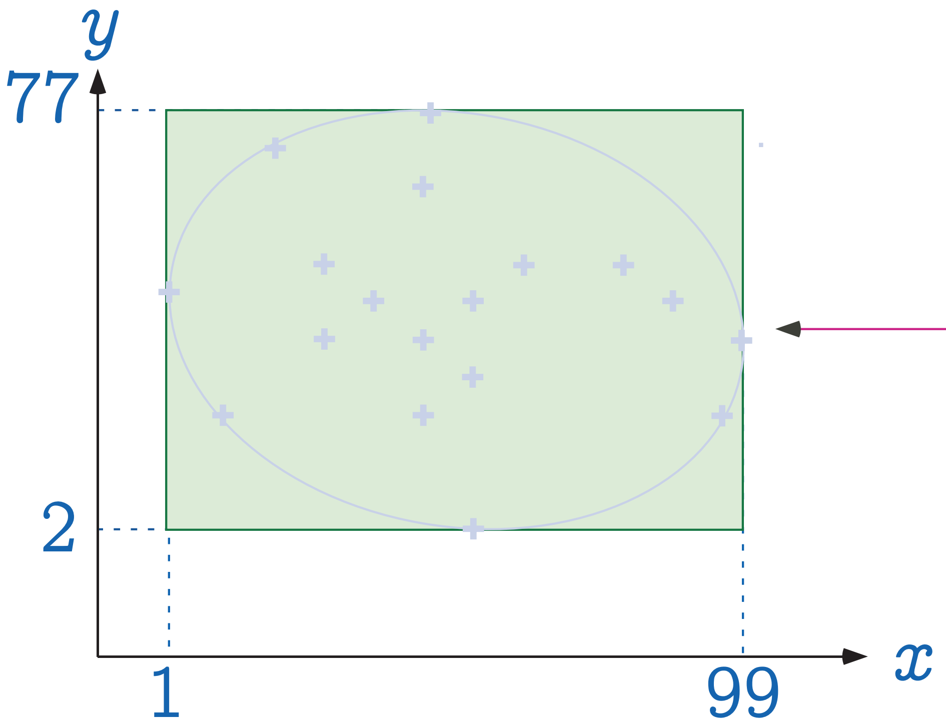
- Choose an abstract domain, replacing sets of objects (states, traces, ...) S by their abstraction $\alpha(S)$
- The abstraction function α maps a set of concrete objects to its abstract interpretation;
- The inverse concretization function γ maps an abstract set of objects to concrete ones;
- Forget no concrete objects: (abstraction from above) $S \subseteq \gamma(\alpha(S))$.

Interval abstraction α



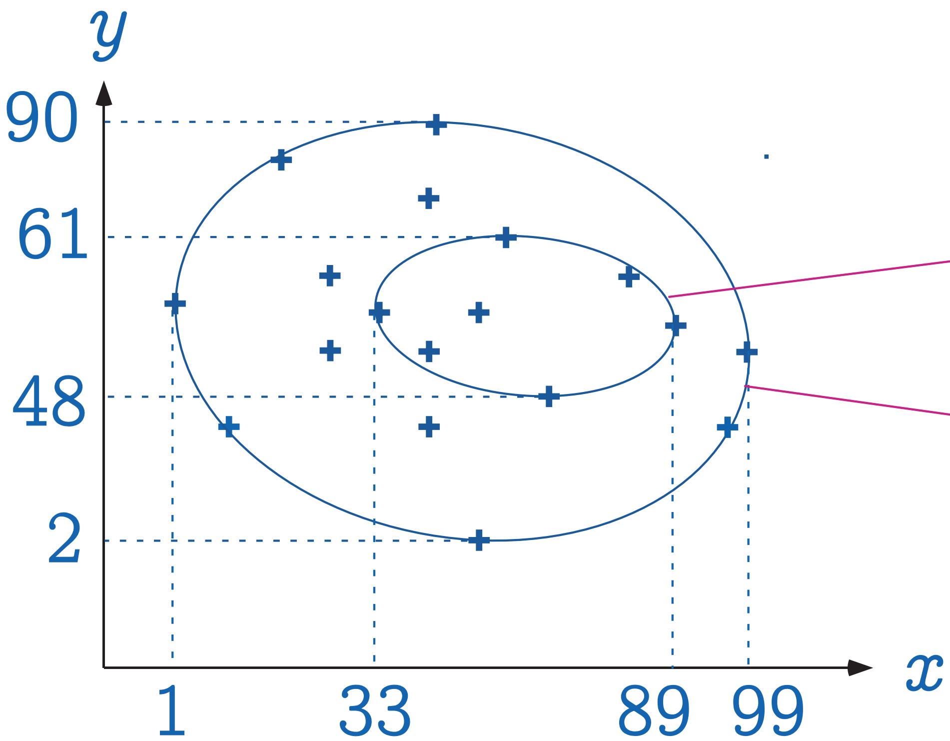
$$\{x : [1, 99], y : [2, 77]\}$$

Interval concretization γ



$$\{x : [1, 99], y : [2, 77]\}$$

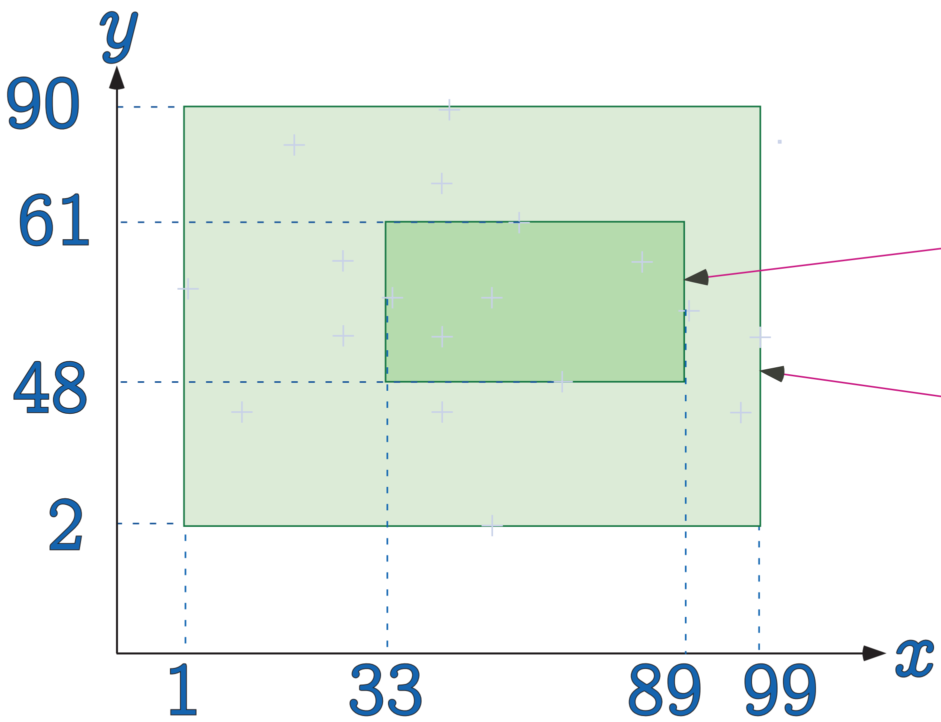
The abstraction α is monotone



$$\{x : [33, 89], y : [48, 61]\}$$
$$\sqsubseteq$$
$$\{x : [1, 99], y : [2, 90]\}$$

$$X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$$

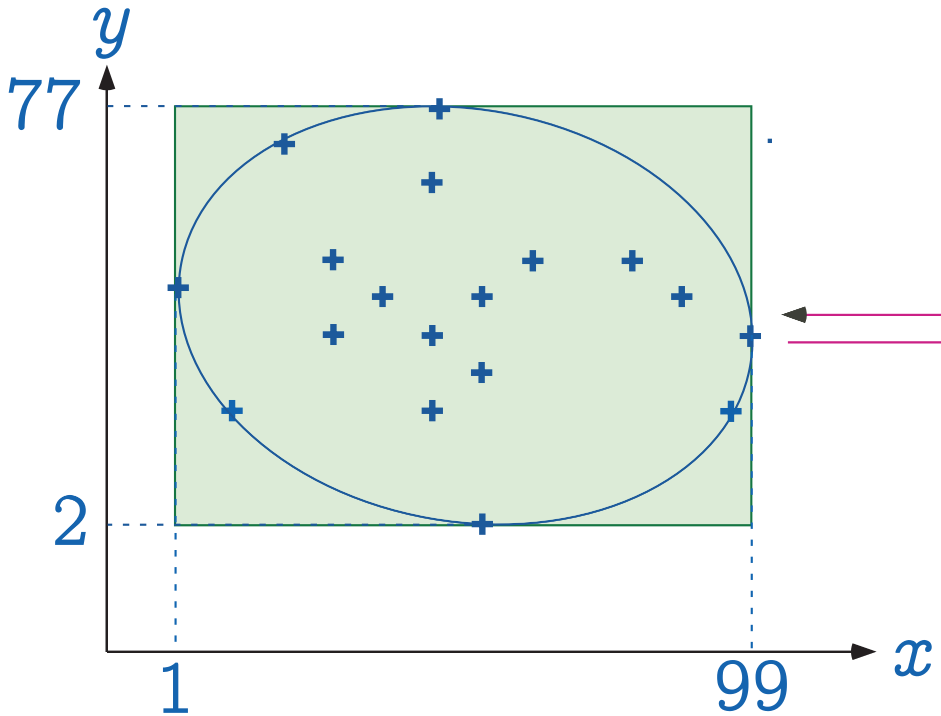
The concretization γ is monotone



$$\{x : [33, 89], y : [48, 61]\}$$
$$\sqsubseteq$$
$$\{x : [1, 99], y : [2, 90]\}$$

$$X \sqsubseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y)$$

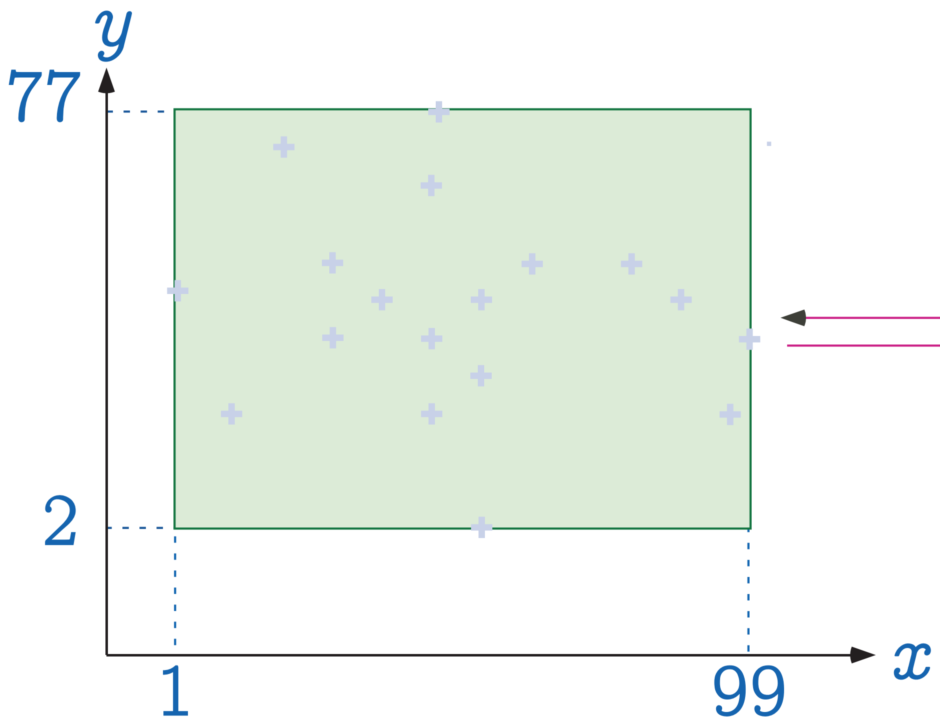
The $\gamma \circ \alpha$ composition is extensive



$\{x : [1, 99], y : [2, 77]\}$

$X \subseteq \gamma \circ \alpha(X)$

The $\alpha \circ \gamma$ composition is reductive



$$\{x : [1, 99], y : [2, 77]\}$$
$$=/\sqsubseteq$$
$$\{x : [1, 99], y : [2, 77]\}$$

$$\alpha \circ \gamma(Y) =/\sqsubseteq Y$$

Correspondance between concrete and abstract properties

– The pair $\langle \alpha, \gamma \rangle$ is a Galois connection:

$$\langle \wp(S), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}, \sqsubseteq \rangle$$

– $\langle \wp(S), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\to}} \langle \mathcal{D}, \sqsubseteq \rangle$ when α is onto (equivalently $\alpha \circ \gamma = 1$ or γ is one-to-one).

Galois connection

$$\langle \mathcal{D}, \subseteq \rangle \xleftarrow[\alpha]{\gamma} \xrightarrow{} \langle \overline{\mathcal{D}}, \sqsubseteq \rangle$$

iff

$$\begin{aligned} & \forall x, y \in \mathcal{D} : x \subseteq y \Longrightarrow \alpha(x) \sqsubseteq \alpha(y) \\ \wedge \; & \forall \overline{x}, \overline{y} \in \overline{\mathcal{D}} : \overline{x} \sqsubseteq \overline{y} \Longrightarrow \gamma(\overline{x}) \subseteq \gamma(\overline{y}) \\ \wedge \; & \forall x \in \mathcal{D} : x \subseteq \gamma(\alpha(x)) \\ \wedge \; & \forall \overline{y} \in \overline{\mathcal{D}} : \alpha(\gamma(\overline{y})) \sqsubseteq \overline{x} \end{aligned}$$

iff

$$\forall x \in \mathcal{D}, \overline{y} \in \overline{\mathcal{D}} : \alpha(x) \sqsubseteq y \Longleftrightarrow x \subseteq \gamma(y)$$

Example: Set of traces to trace of intervals abstraction

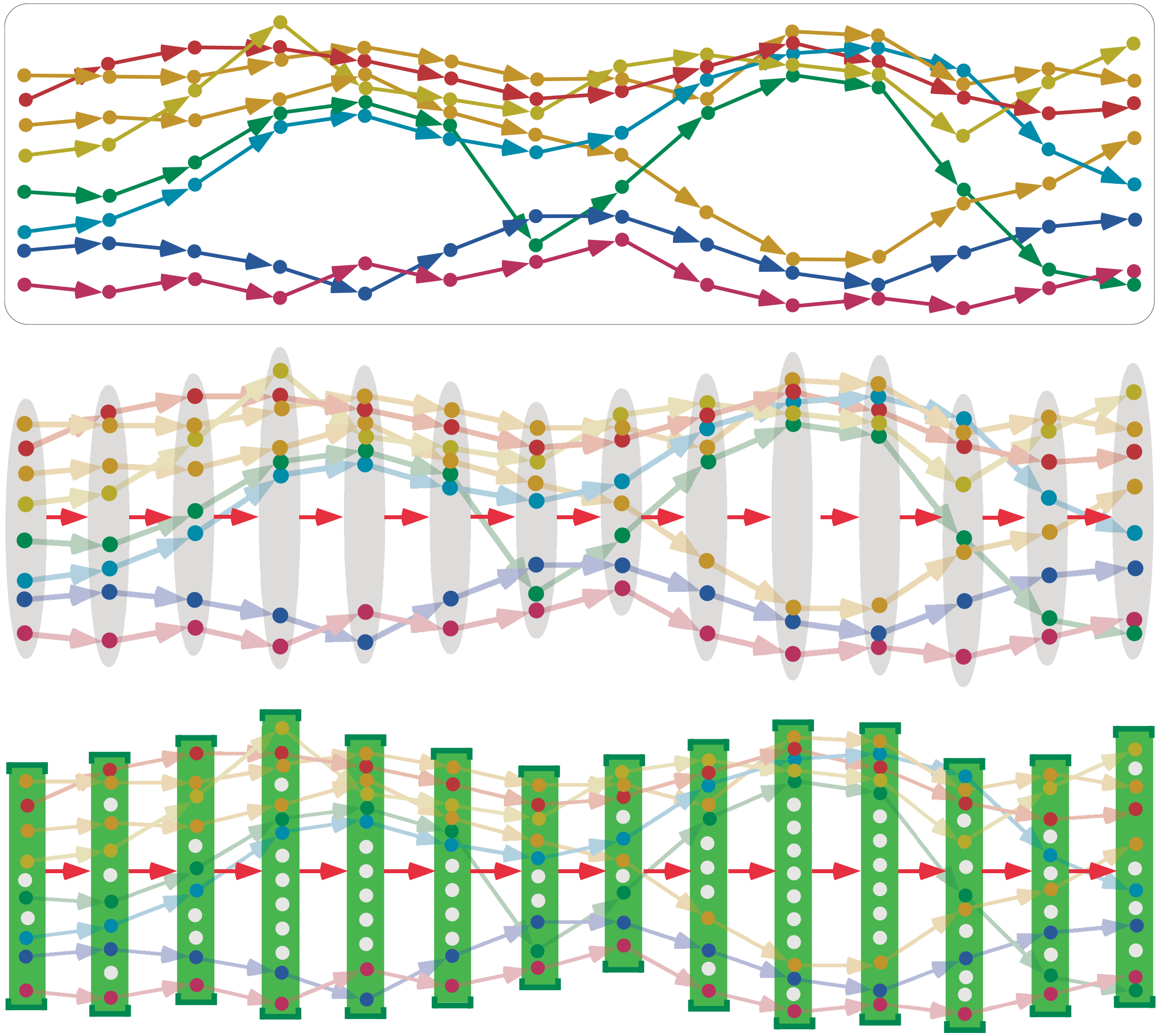
Set of traces:

$\alpha_1 \downarrow$

Trace of sets:

$\alpha_2 \downarrow$

Trace of intervals



Example: Set of traces to reachable states abstraction

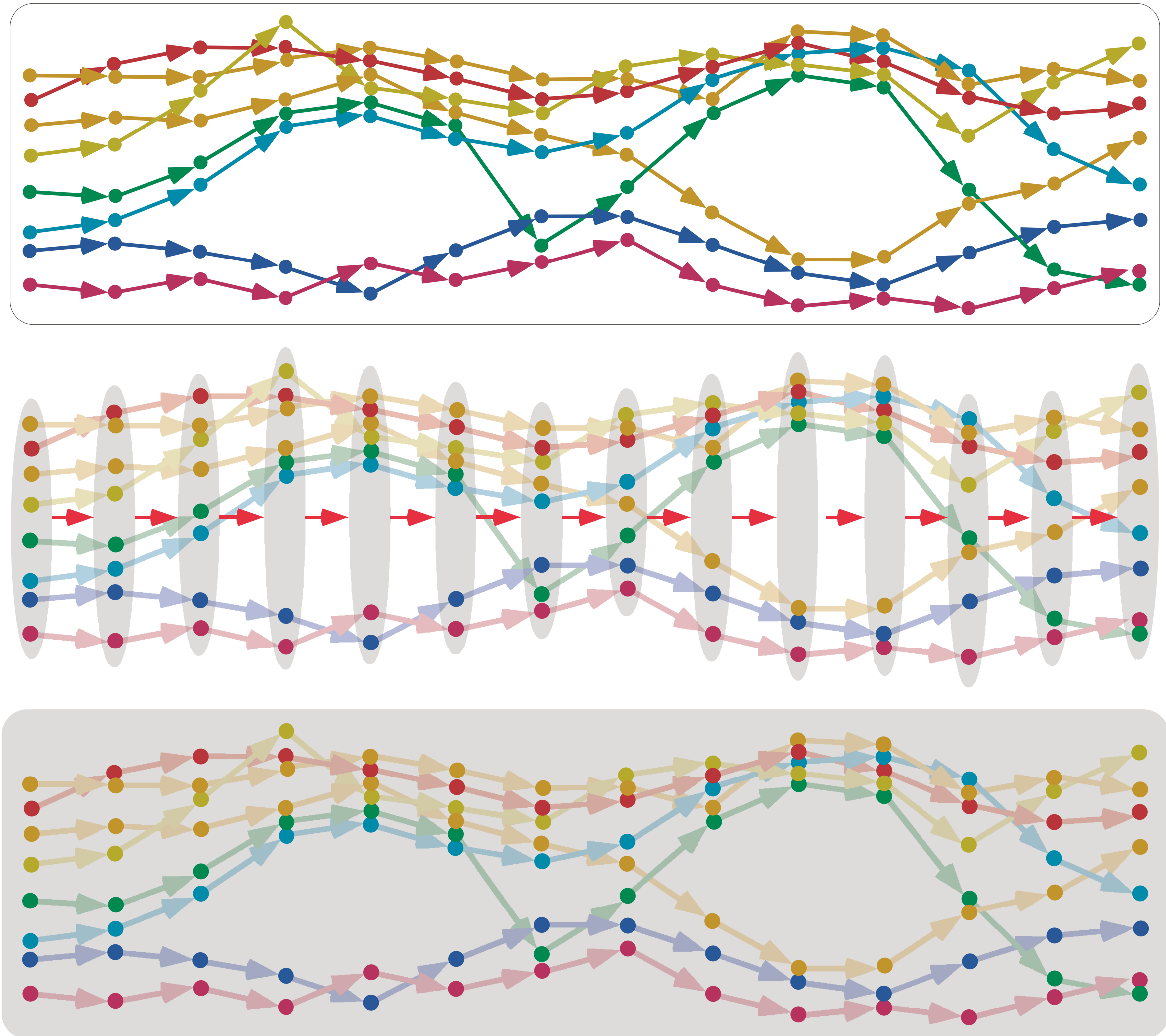
Set of traces:

$\alpha_1 \downarrow$

Trace of sets:

$\alpha_3 \downarrow$

Reachable states



Composition of Galois Connections

The composition of Galois connections:

$$\langle L, \leq \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle M, \sqsubseteq \rangle$$

and:

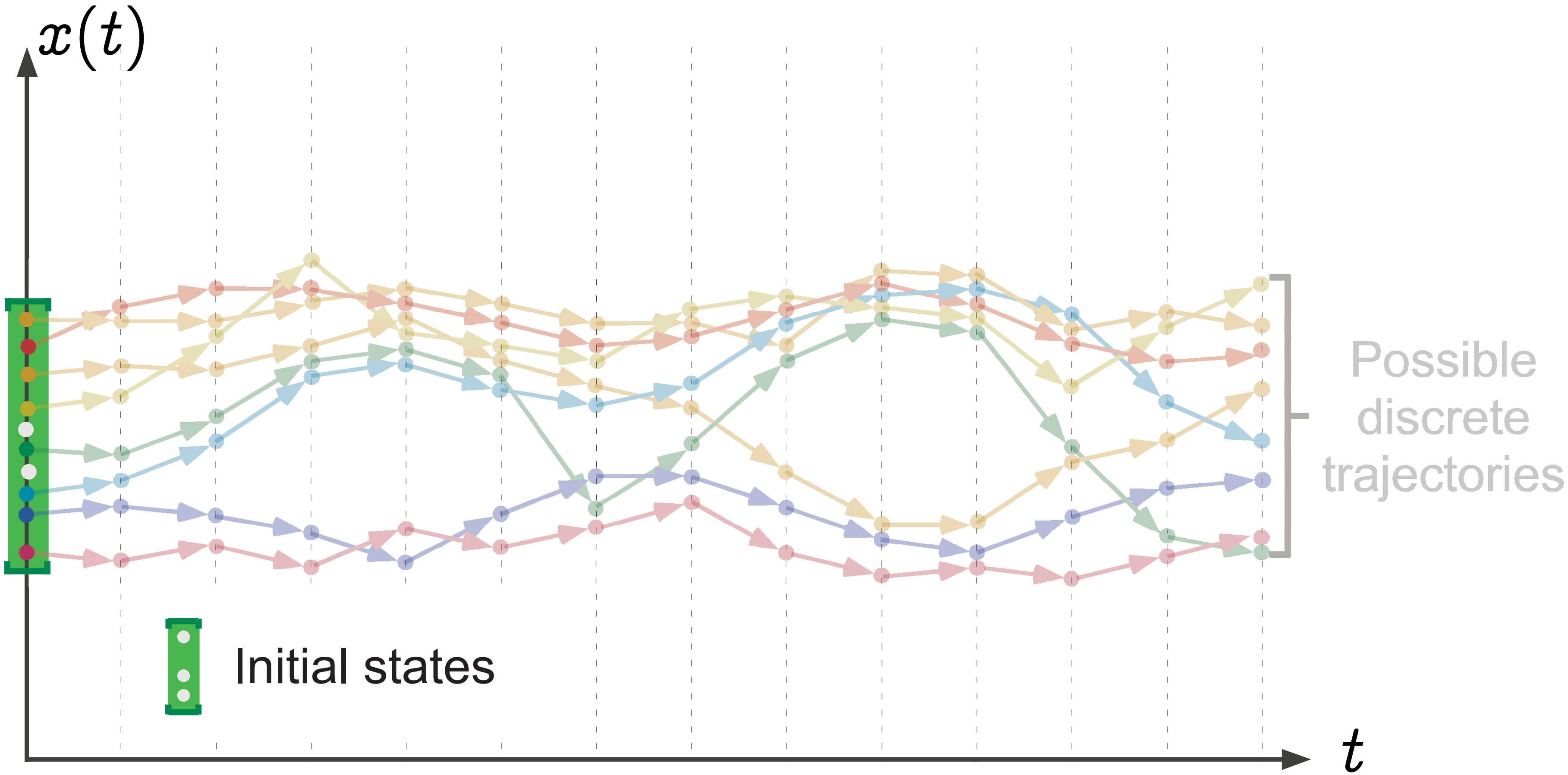
$$\langle M, \sqsubseteq \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle N, \preceq \rangle$$

is a Galois connection:

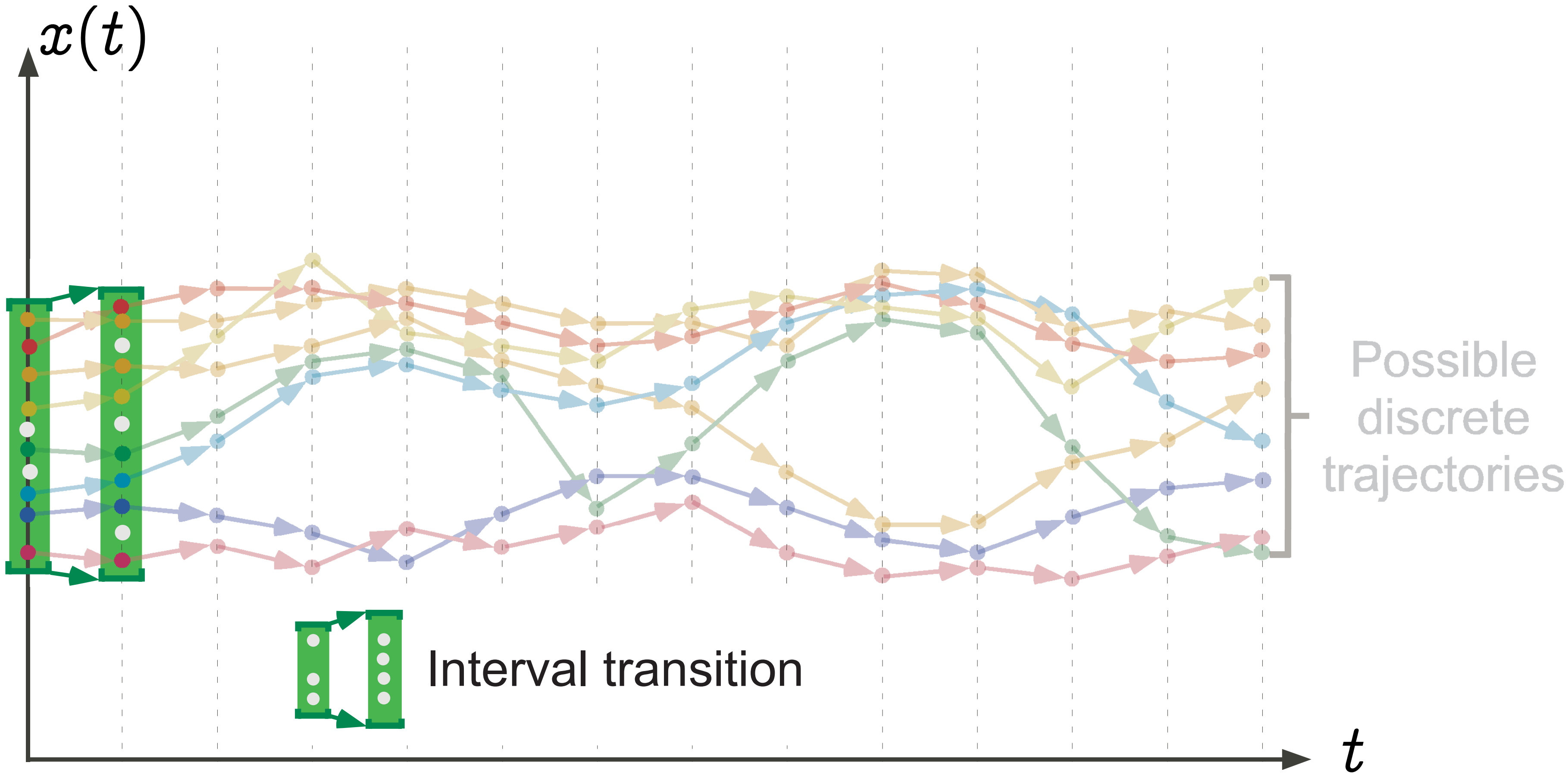
$$\langle L, \leq \rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle N, \preceq \rangle$$

Convergence acceleration by widening/narrowing

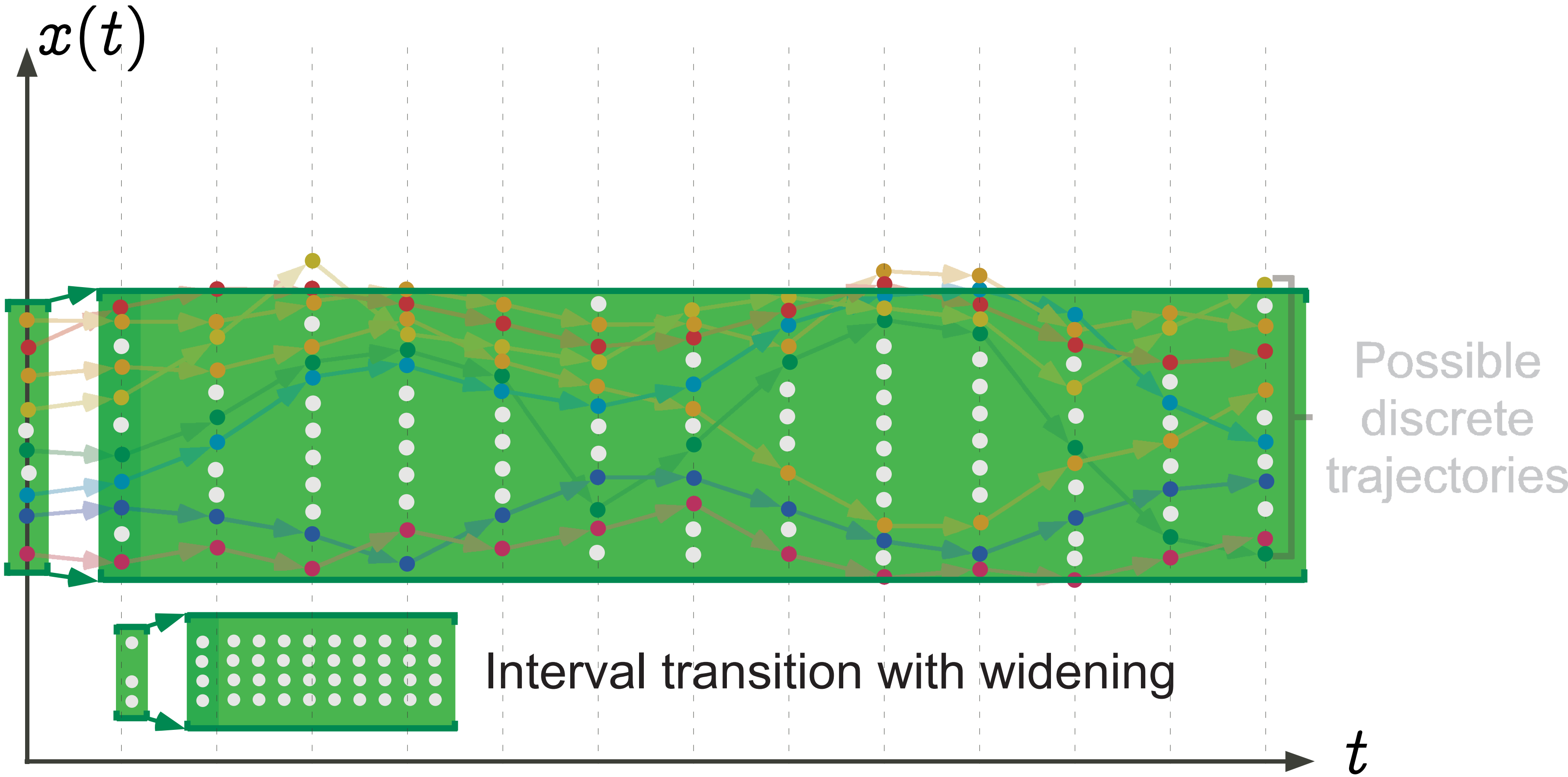
Graphic example: upward iteration with widening



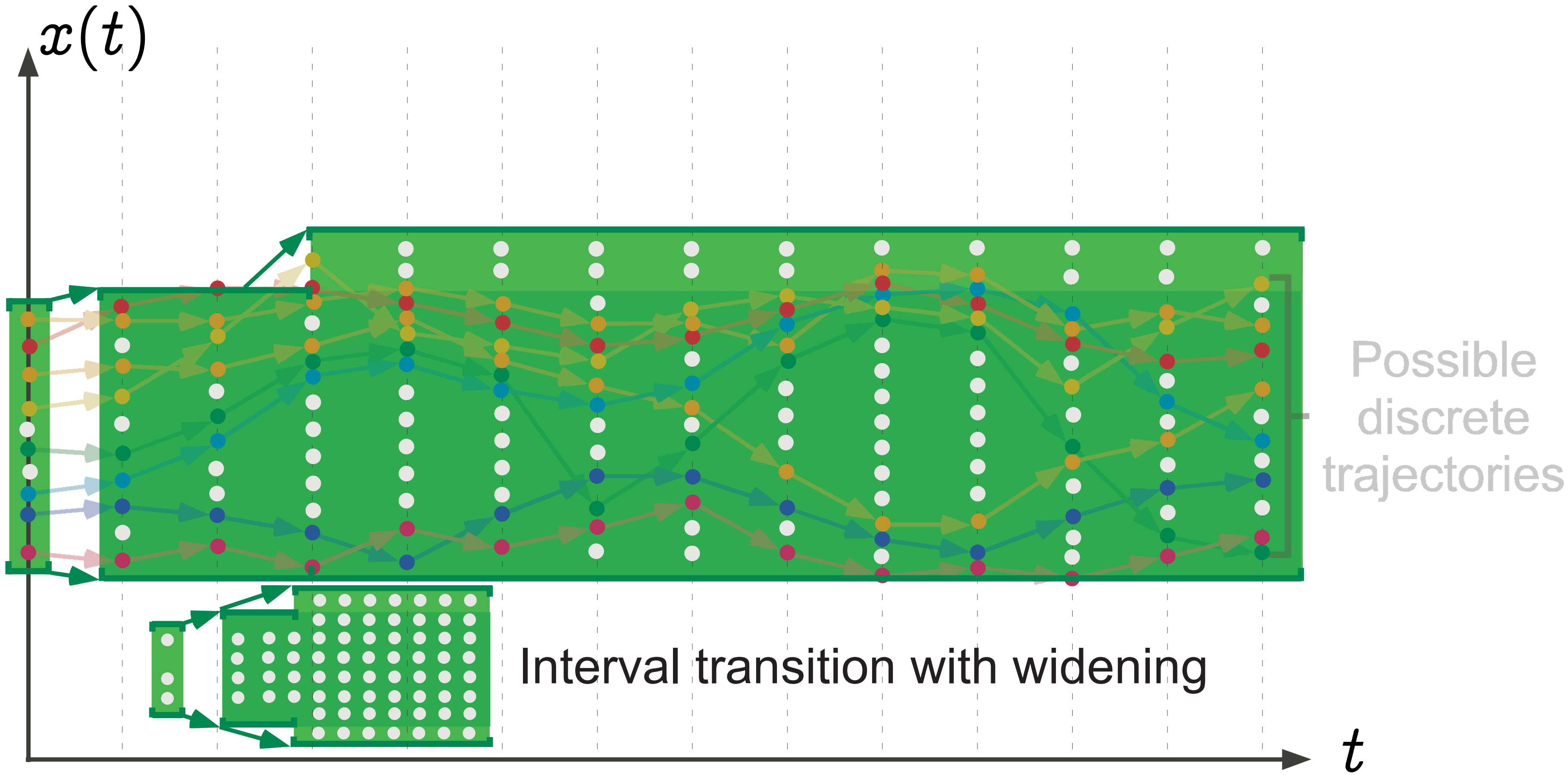
Graphic example: upward iteration with widening



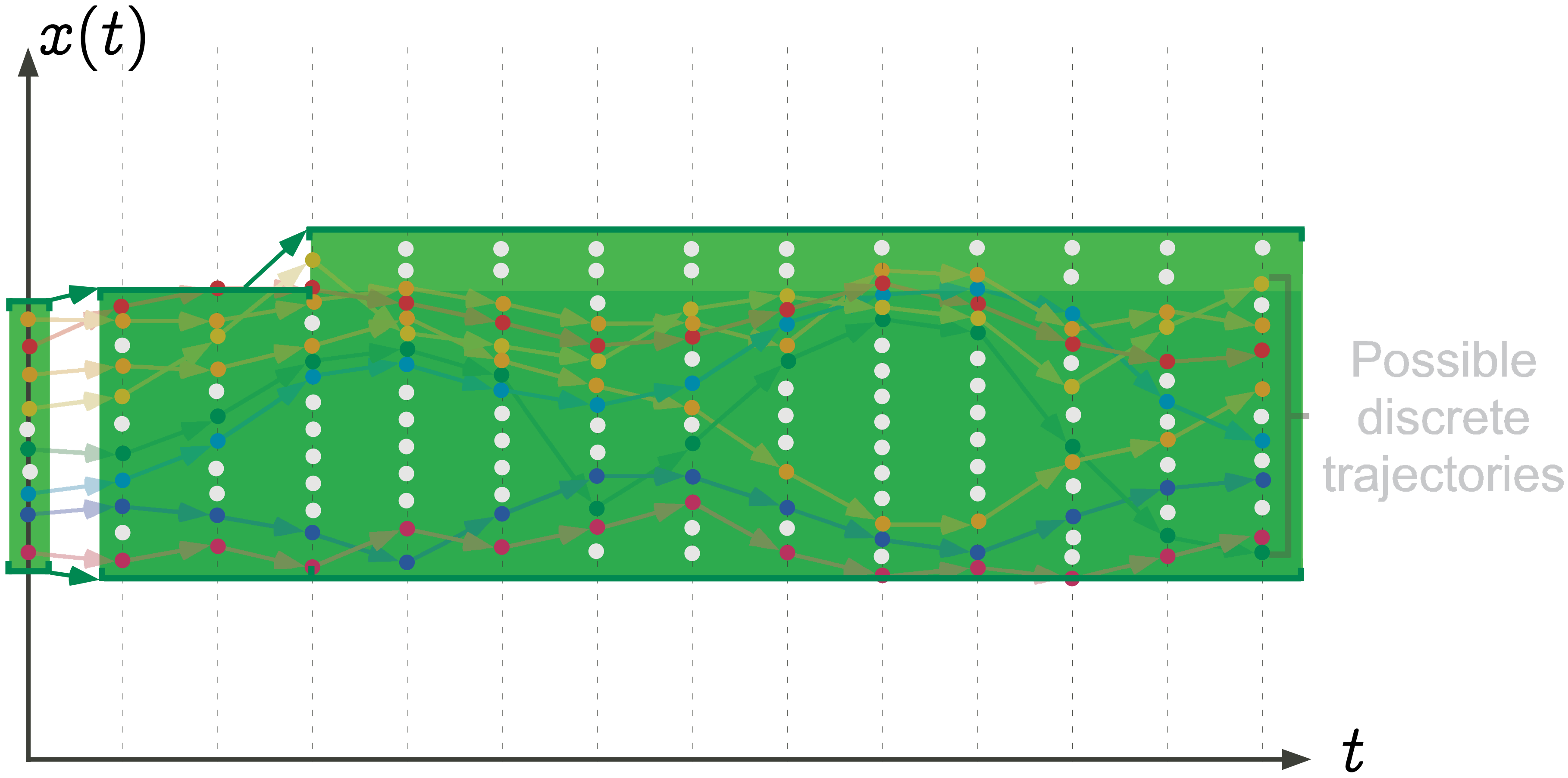
Graphic example: upward iteration with widening



Graphic example: upward iteration with widening



Graphic example: stability of the upward iteration



Interval widening

– $\overline{L} = \{\bot\} \cup \{[\ell, u] \mid \ell, u \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{\} \wedge \ell \leq u\}$

– The widening extrapolates unstable bounds to infinity:

$$\begin{aligned}
\bot \,\nabla\, X &= X \\
X \,\nabla\, \bot &= X \\
[\ell_0,\ u_0] \,\nabla\, [\ell_1,\ u_1] &= [\text{if } \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0, \\
&\qquad \text{if } \mathrm{u}_1 > \mathrm{u}_0 \text{ then } +\infty \text{ else } \mathrm{u}_0]
\end{aligned}$$

Not monotone. For example $[0,\ 1] \sqsubseteq [0,\ 2]$ but $[0,\ 1] \,\nabla\, [0,\ 2] = [0,\ +\infty] \not\sqsubseteq [0,\ 2] = [0,\ 2] \,\nabla\, [0,\ 2]$

Example: Interval analysis (1975)

Program to be analyzed:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

Example: Interval analysis (1975)

Equations (abstract interpretation of the semantics):

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

Example: Interval analysis (1975)

Resolution by chaotic increasing iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = \emptyset \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration:

```
     x := 1;
1:
     while x < 10000 do
2:
         x := x + 1
3:
     od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
        x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,2] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 2] \\ X_3 = [2, 3] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!**

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,3] \\ X_3 = [2,3] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!!**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 3] \\ X_3 = [2, 4] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!!!**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 4] \\ X_3 = [2, 4] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!!!!**

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 4] \\ X_3 = [2, 5] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!!!!!**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,5] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Increasing chaotic iteration: **convergence !!!!!!!**

```
   x := 1;
1:
   while x < 10000 do
2:
        x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 5] \\ X_3 = [2, 6] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Convergence speed-up by widening:

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, +\infty] \quad \Leftarrow \text{widening} \\ X_3 = [2, 6] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
      x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, +\infty] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = \emptyset \end{cases}$$

Example: Interval analysis (1975)

Final solution:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

Example: Interval analysis (1975)

Result of the interval analysis:

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1,9999]}
      x := x + 1
3: {x ∈ [2,+10000]}
   od;
4: {x = 10000}
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

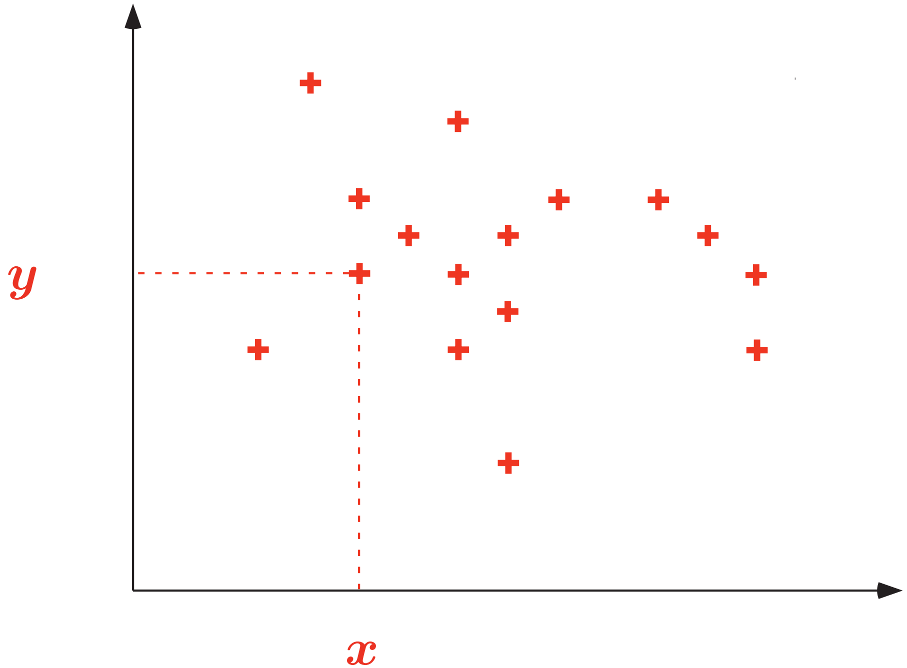
Example: Interval analysis (1975)

Checking absence of runtime errors with interval analysis:

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}
         x := x + 1        ←── no overflow
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

Refinement of abstractions

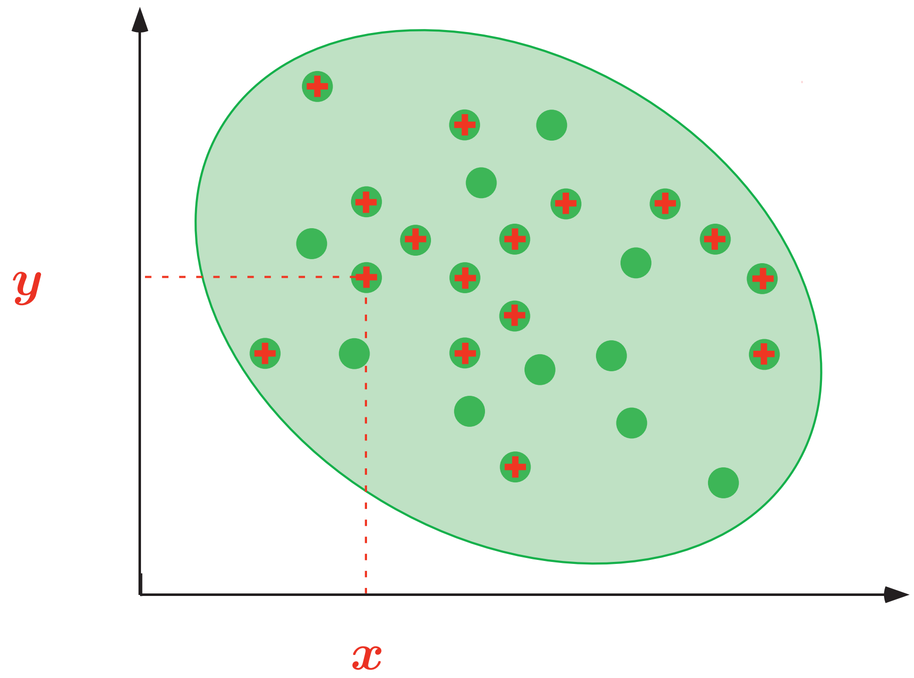
Approximations of an [in]finite set of points:



$$\{\ldots, \langle 19,\ 77\rangle, \ldots,\\ \langle 20,\ 03\rangle, \ldots\}$$

Approximations of an [in]finite set of points:

from above



$$\{\ldots, \langle 19,\ 77\rangle, \ldots,$$

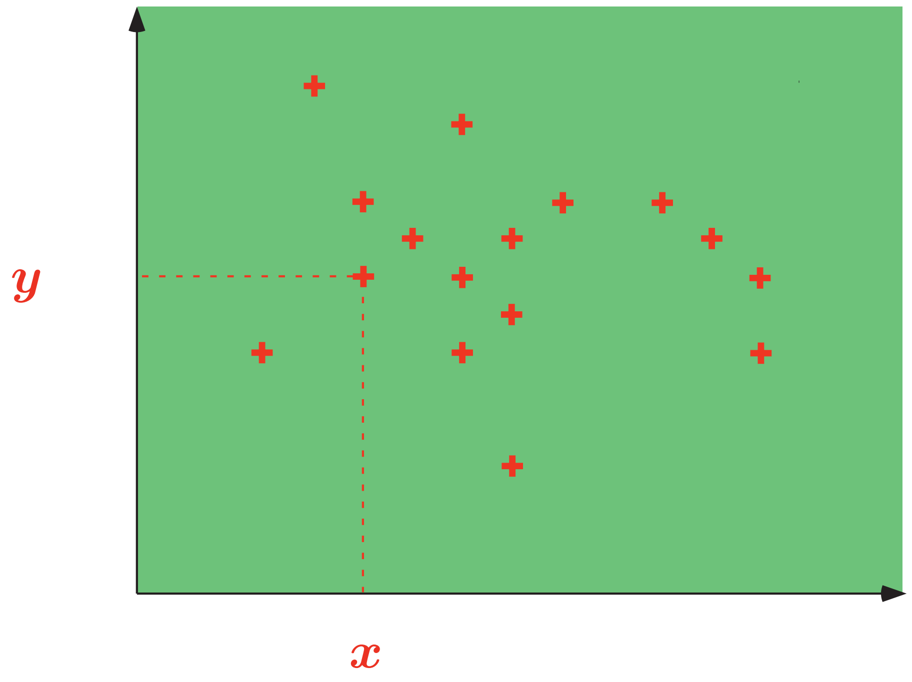
$$\langle 20,\ 03\rangle, \langle ?,\ ?\rangle, \ldots\}$$

From Below: dual [3] + combinations.

---

[3] Trivial for finite states (liveness model-checking), more difficult for infinite states (variant functions).

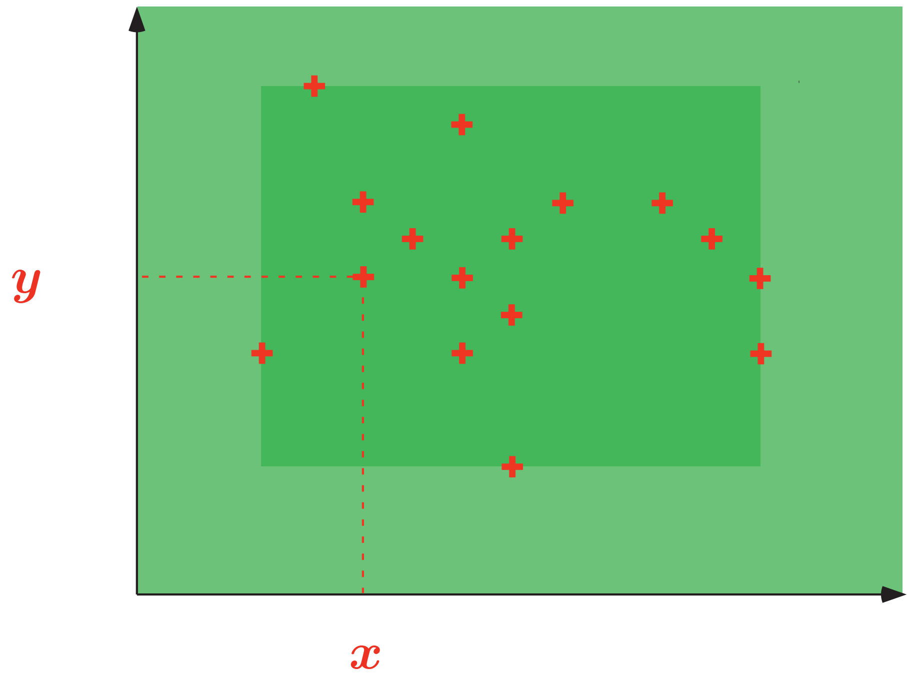
Effective computable approximations of an [in]finite set of points; Signs [4]



$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

[4] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

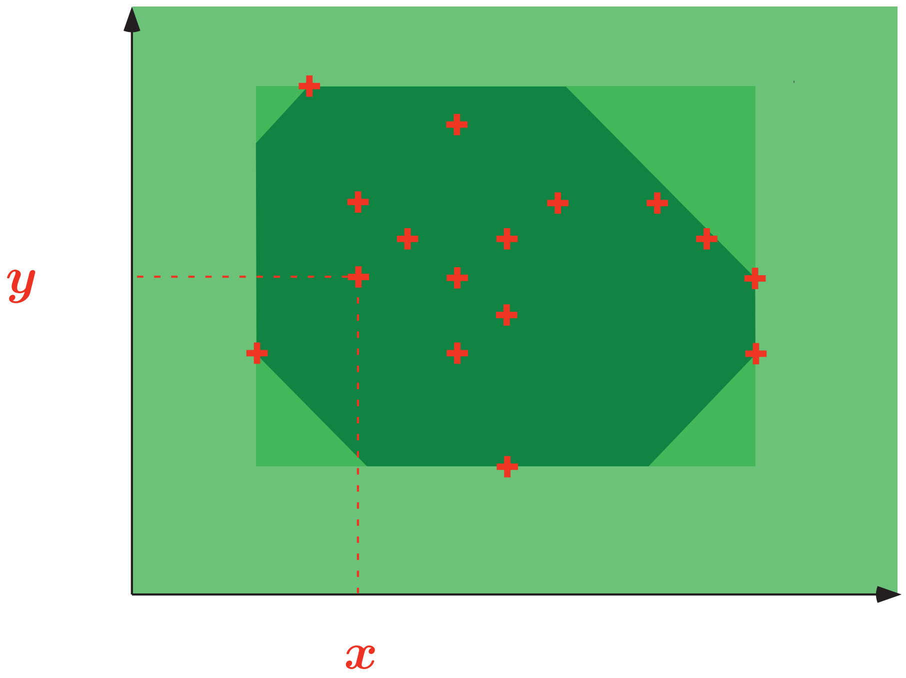
Effective computable approximations of an [in]finite set of points; Intervals [5]



$$\begin{cases} x \in [19, 77] \\ y \in [20, 03] \end{cases}$$

[5] P. Cousot & R. Cousot. *Static determination of dynamic properties of programs*. Proc. 2nd Int. Symp. on Programming, Dunod, 1976.

Effective computable approximations of an [in]finite set of points; Octagons [6]

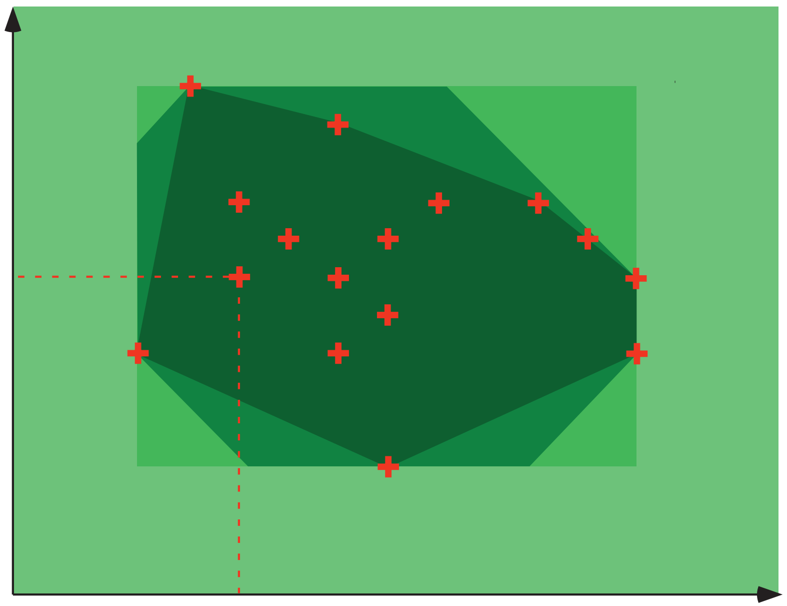


$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

6 A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices*. PADO '2001. LNCS 2053, pp. 155–172. Springer 2001. See the *The Octagon Abstract Domain Library* on `http://www.di.ens.fr/~mine/oct/`

Effective computable approximations of an [in]finite set of points; Polyhedra [7]

y

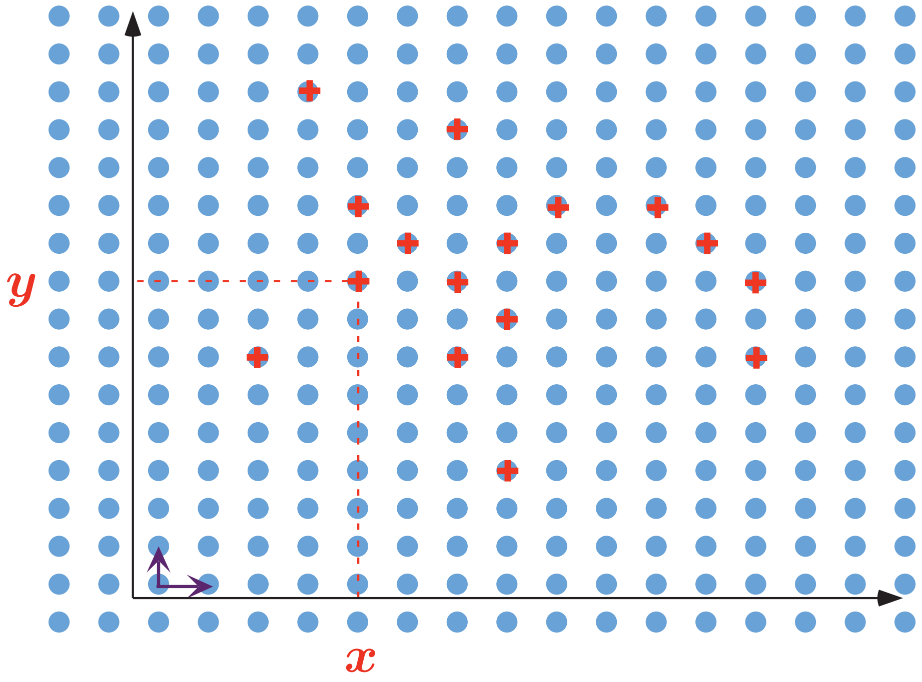


x

$$\begin{cases} 19x + 77y \leq 2004 \\ 20x + 03y \geq 0 \end{cases}$$

[7] P. Cousot & N. Halbwachs. *Automatic discovery of linear restraints among variables of a program*. ACM POPL, 1978, pp. 84–97.

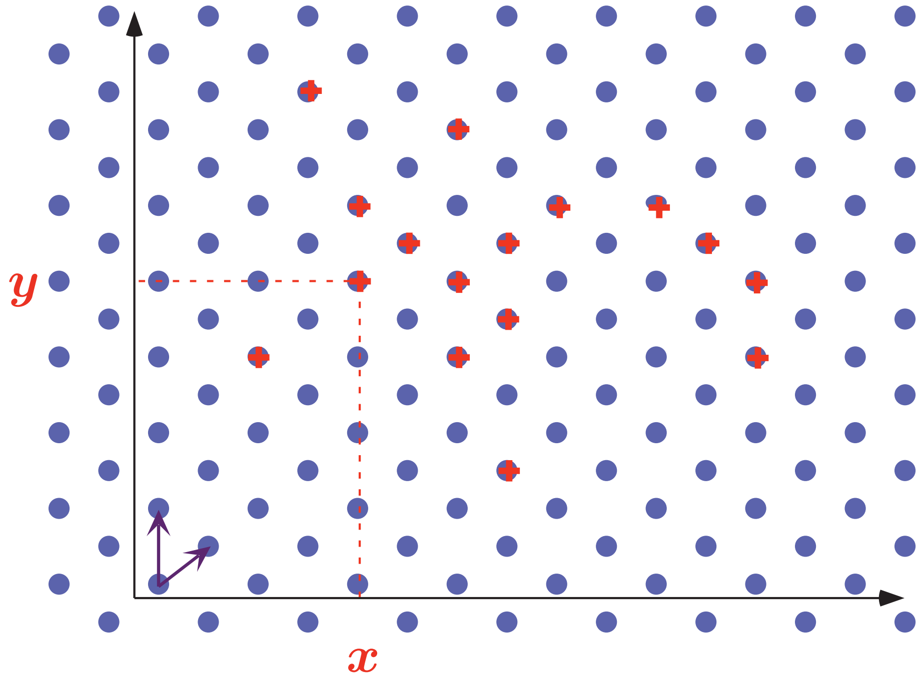
Effective computable approximations of an [in]finite set of points; Simple congruences [8]



$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

8 Ph. Granger. *Static Analysis of Arithmetical Congruences*. Int. J. Comput. Math. 30, 1989, pp. 165–190.

Effective computable approximations of an [in]finite set of points; Linear congruences [9]

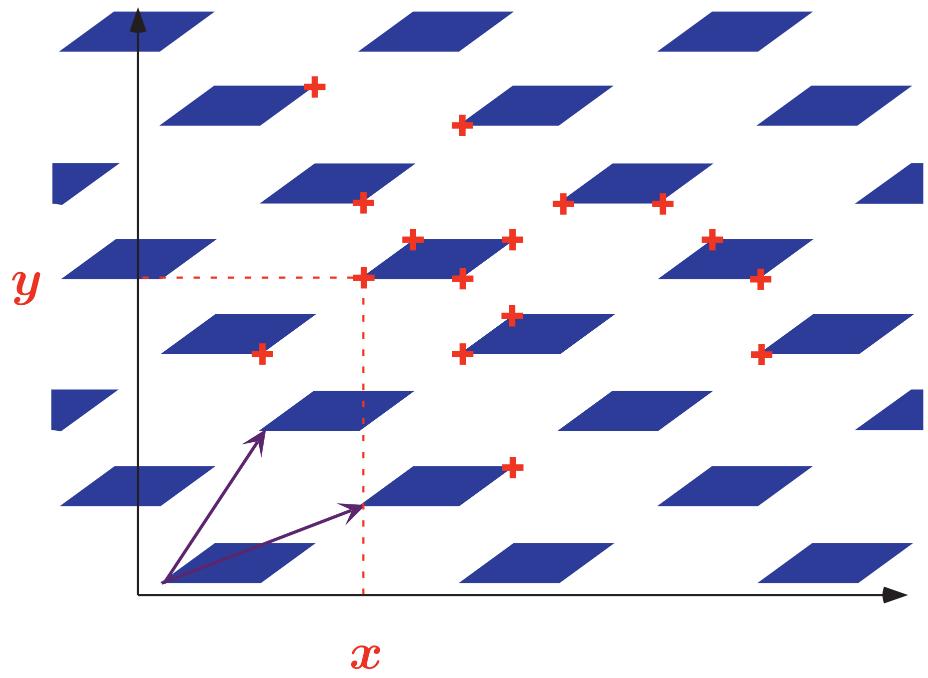


$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

---

[9] Ph. Granger. *Static Analysis of Linear Congruence Equalities among Variables of a Program.* TAPSOFT '91, pp. 169–192. LNCS 493, Springer, 1991.

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences [10]

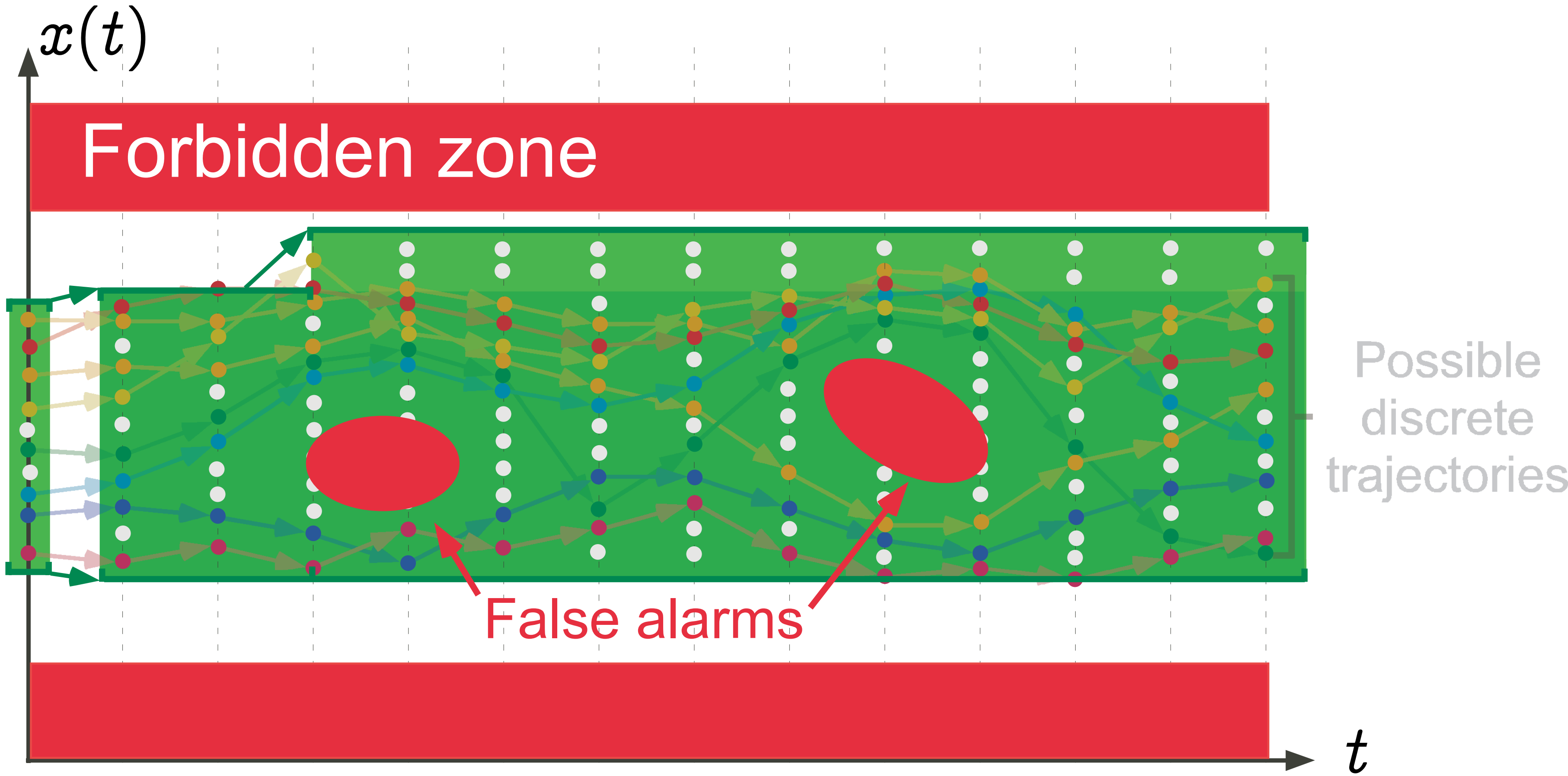


$$\begin{cases} 1x + 9y \in [0,77] \bmod 10 \\ 2x - 1y \in [0,99] \bmod 11 \end{cases}$$

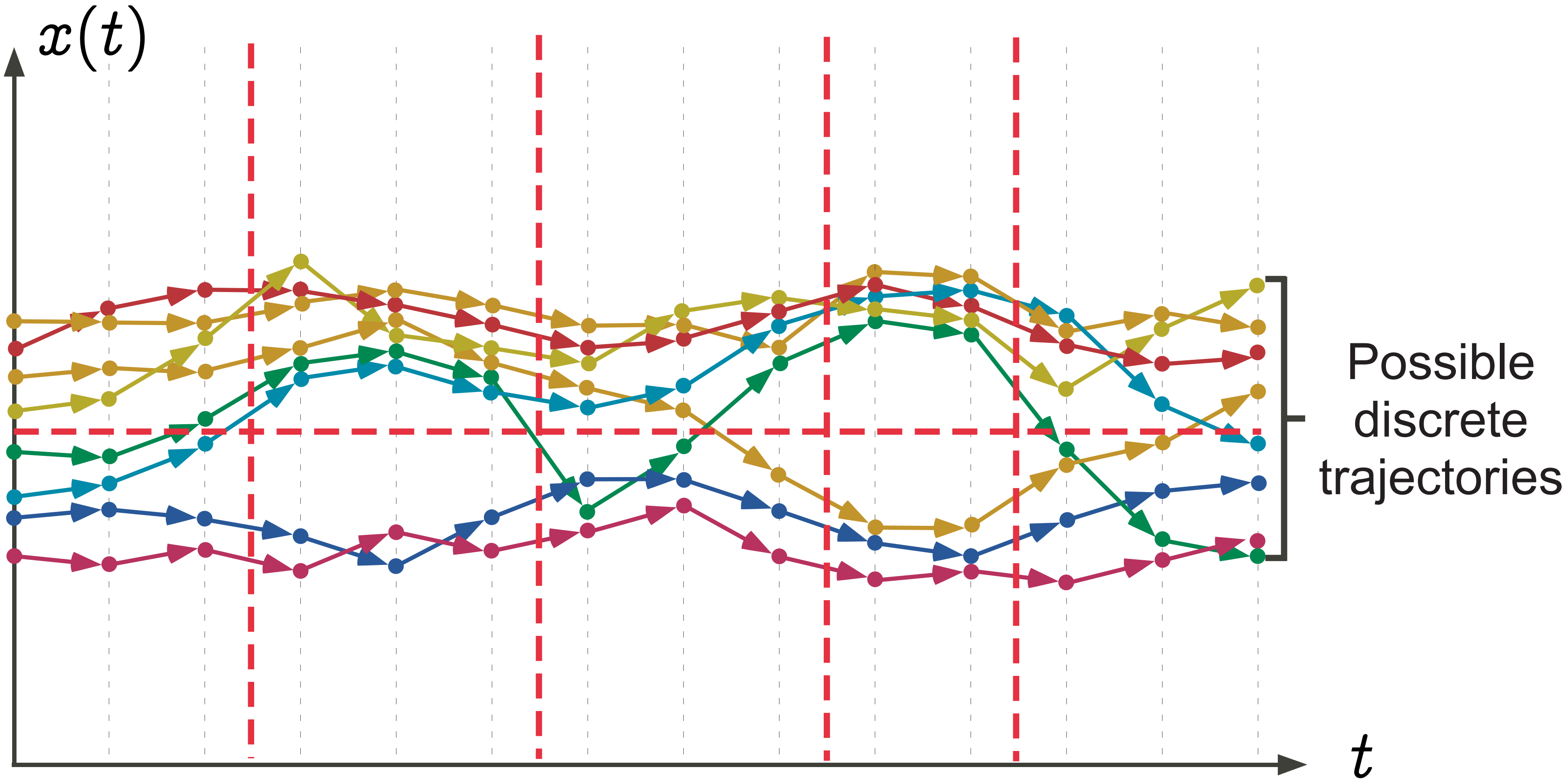
10 F. Masdupuy. *Array Operations Abstraction Using Semantic Analysis of Trapezoid Congruences.* ACM ICS '92.

Refinement of iterates

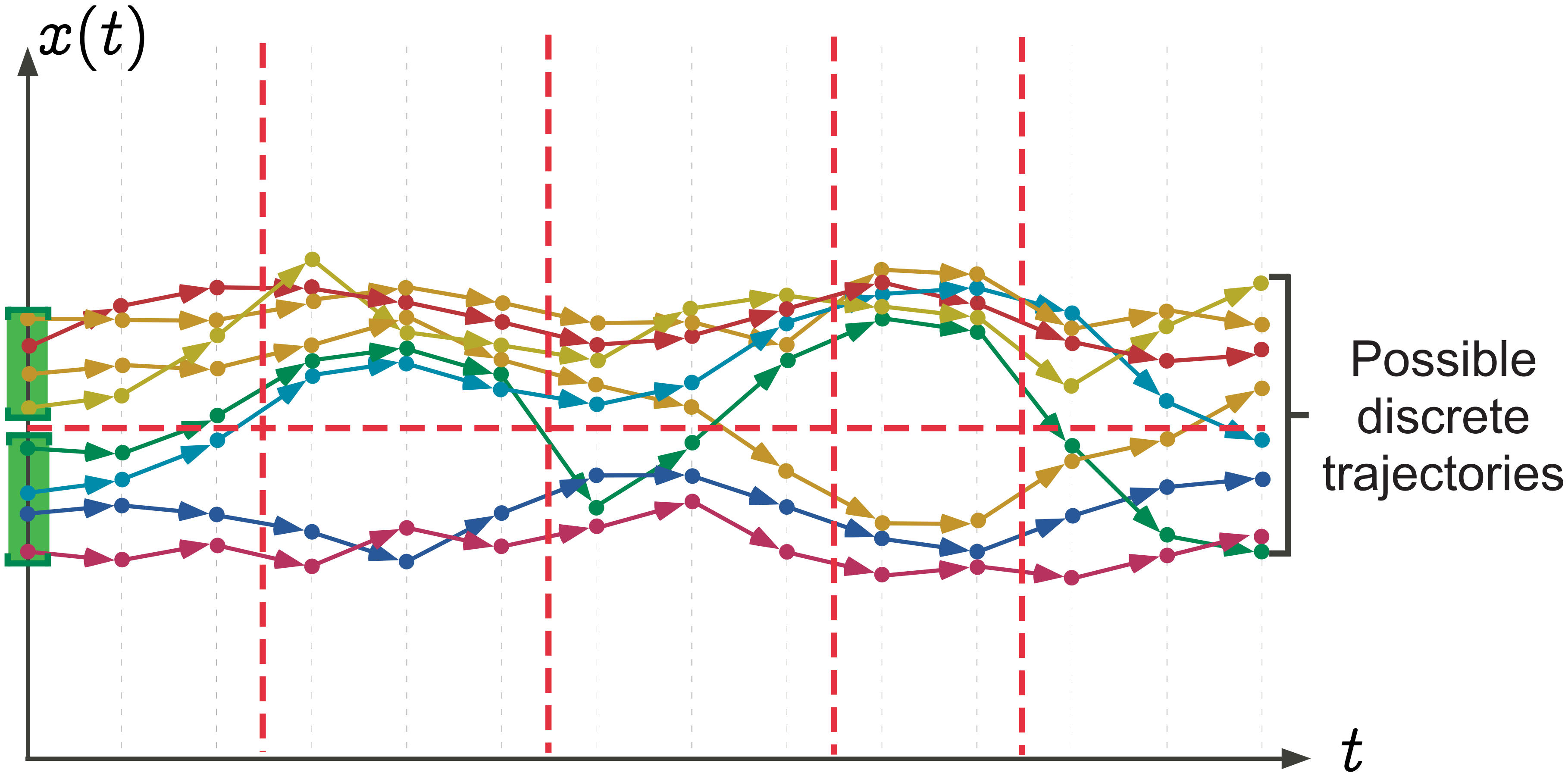
Graphic example: Refinement required by false alarms



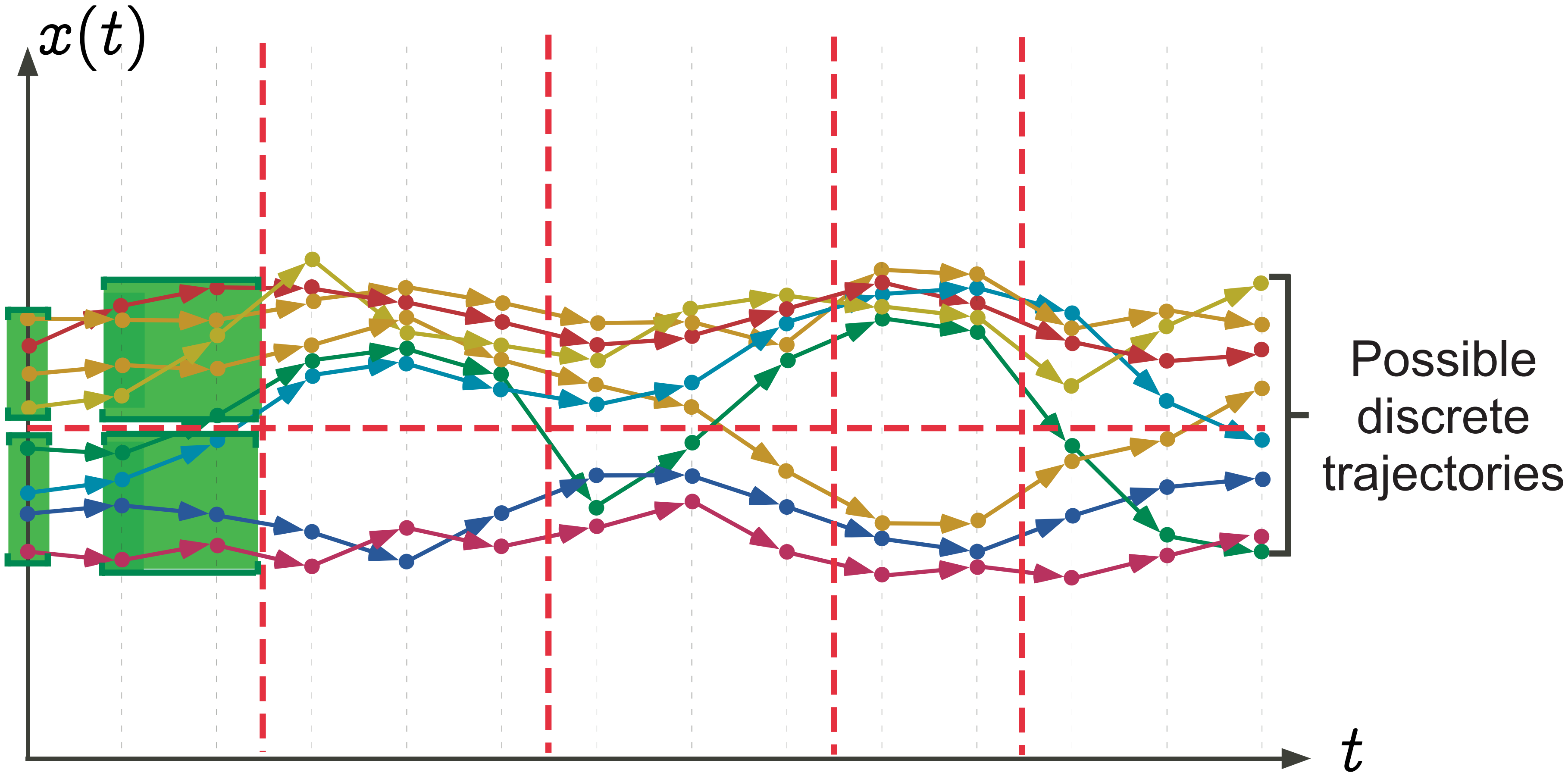
Graphic example: Partitionning



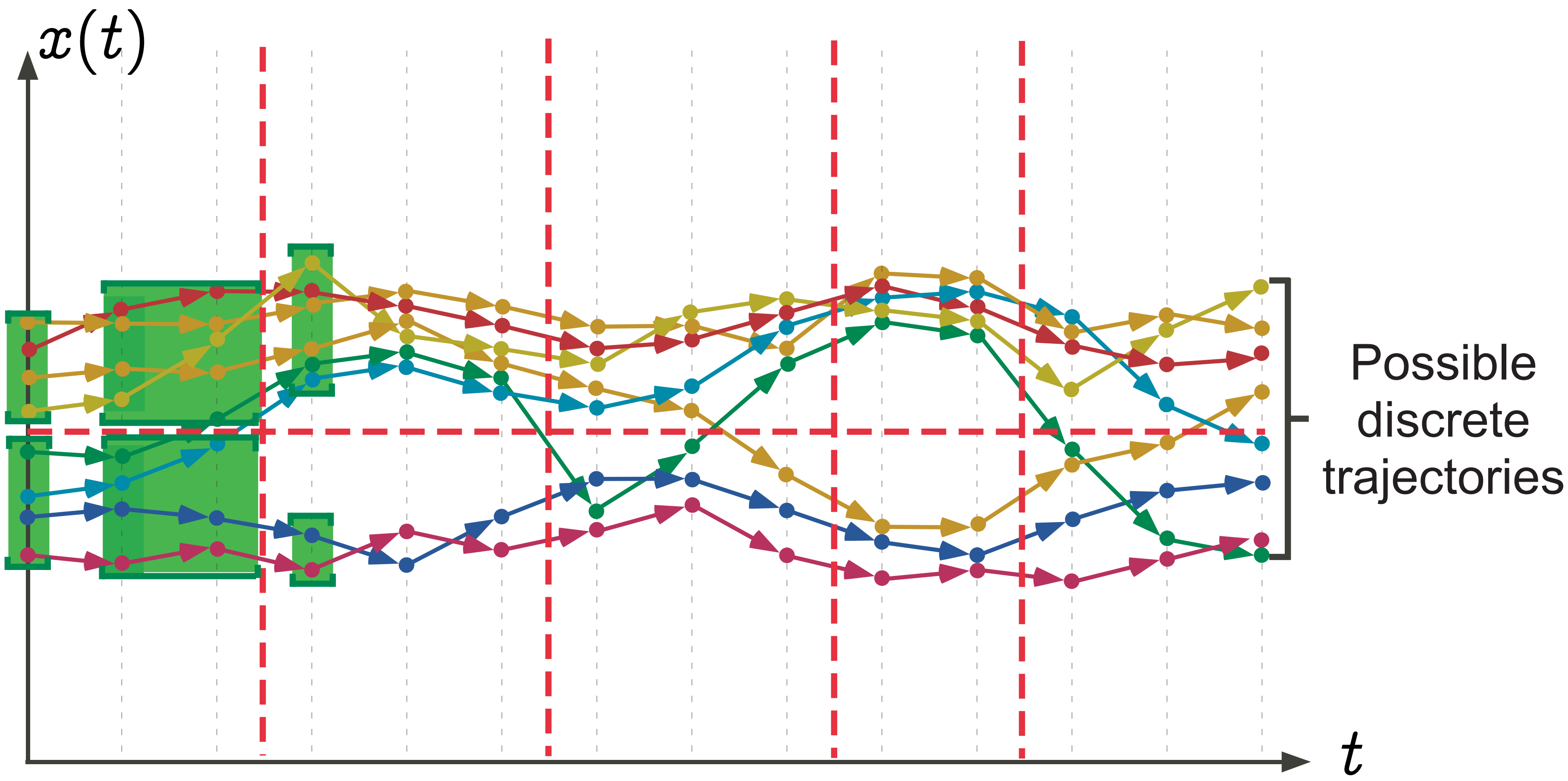
Graphic example: partitionned upward iteration with widening



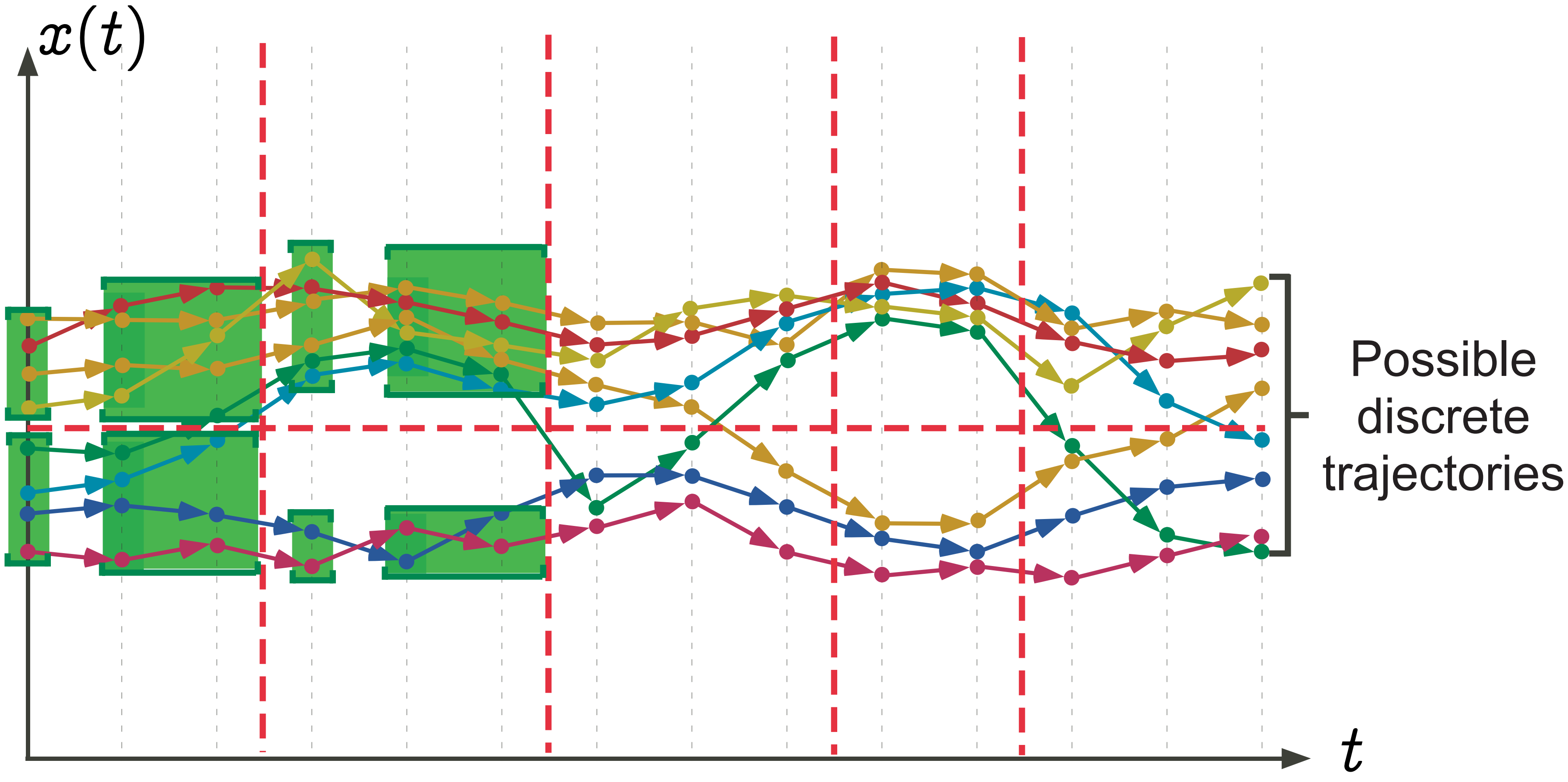
Graphic example: partitionned upward iteration with widening



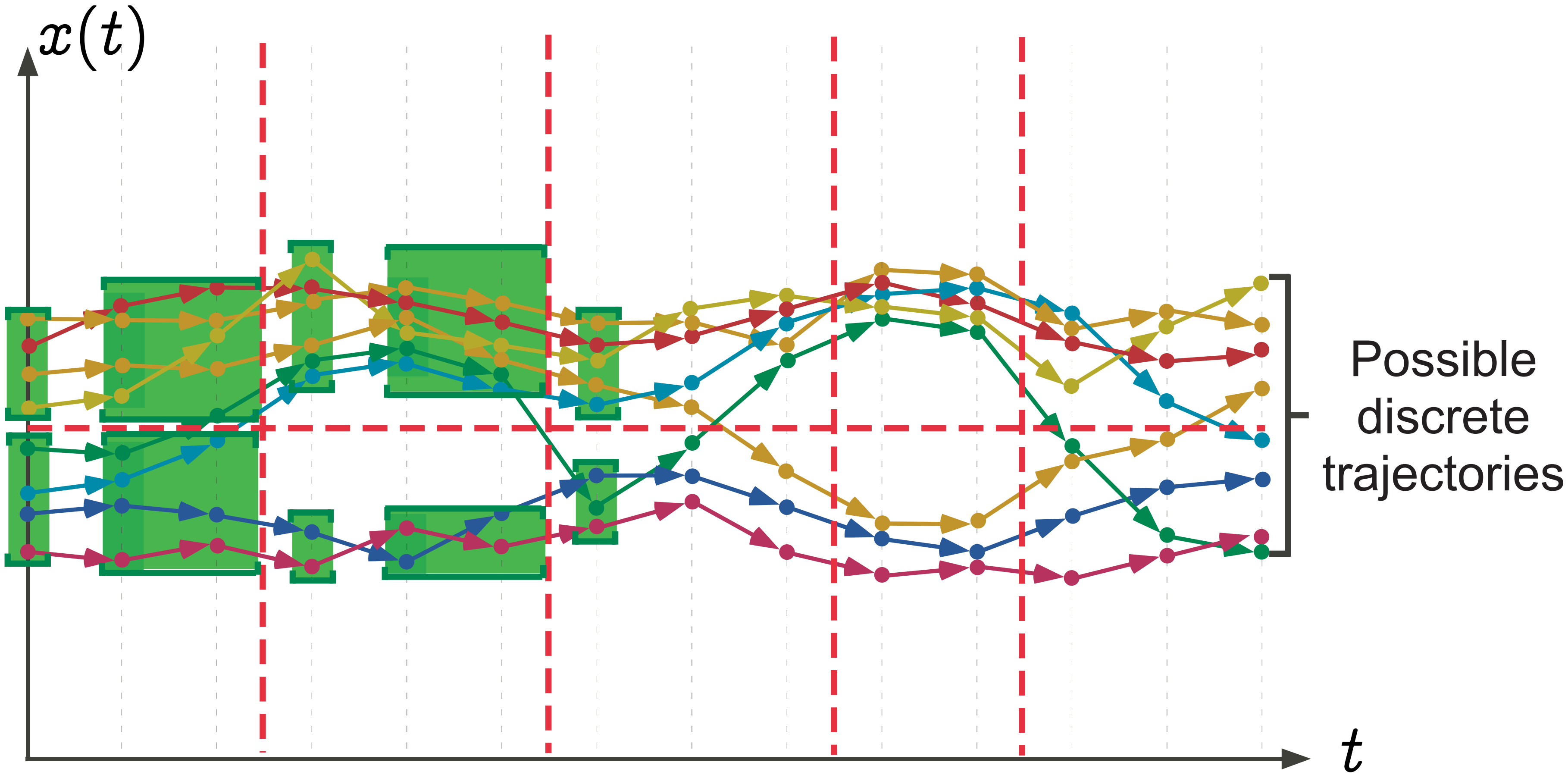
Graphic example: partitionned upward iteration with widening



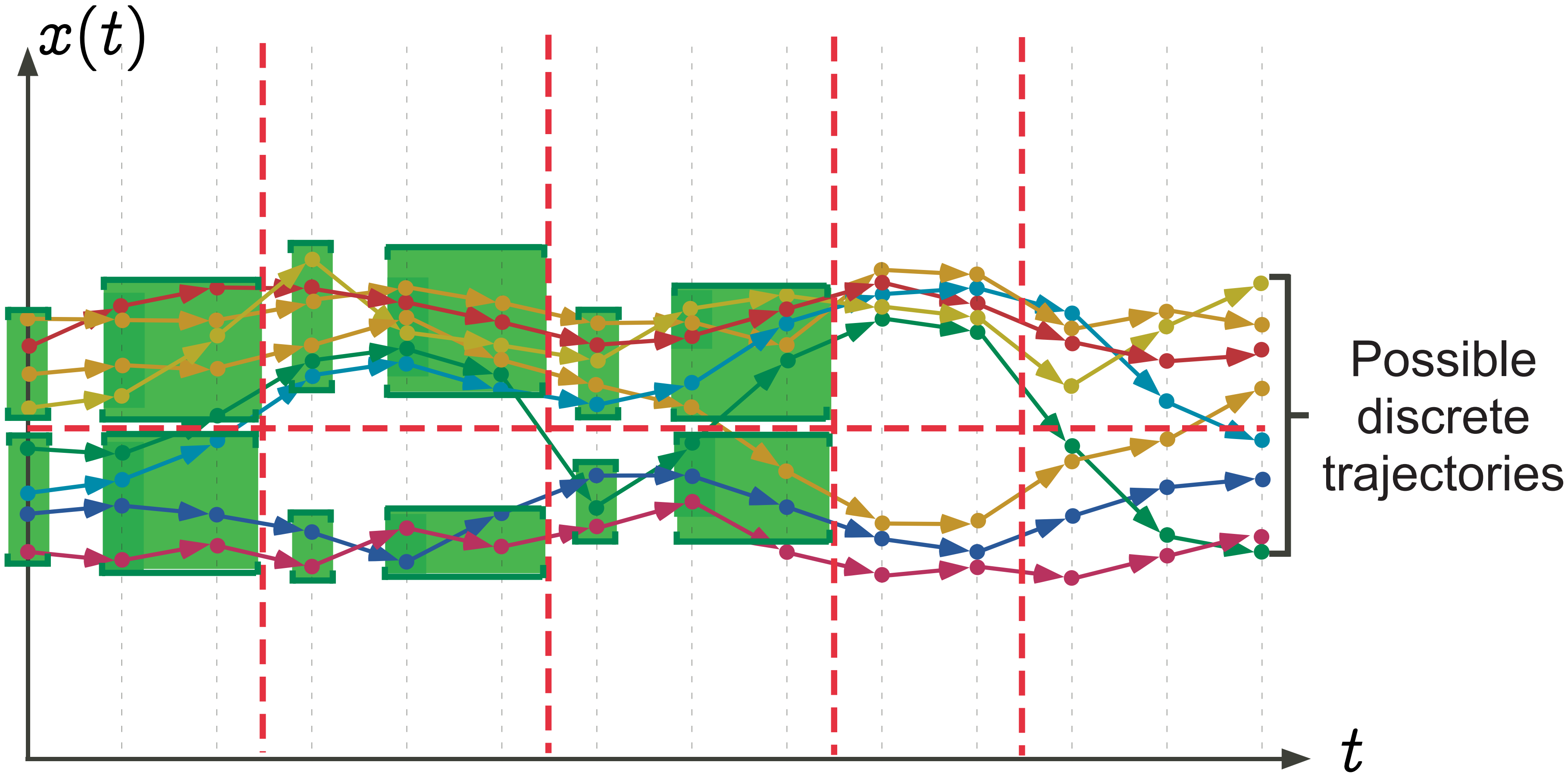
Graphic example: partitionned upward iteration with widening



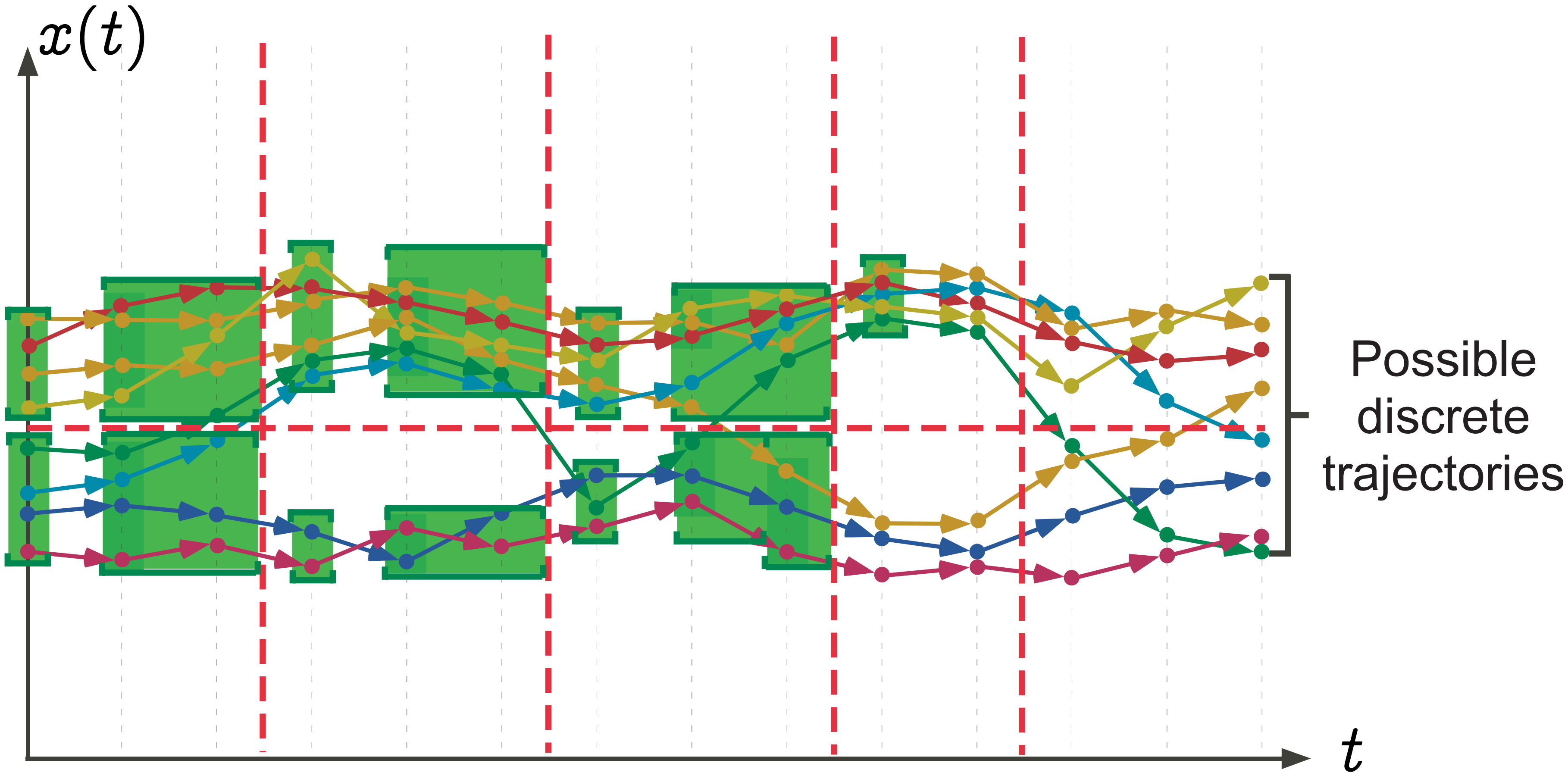
Graphic example: partitionned upward iteration with widening



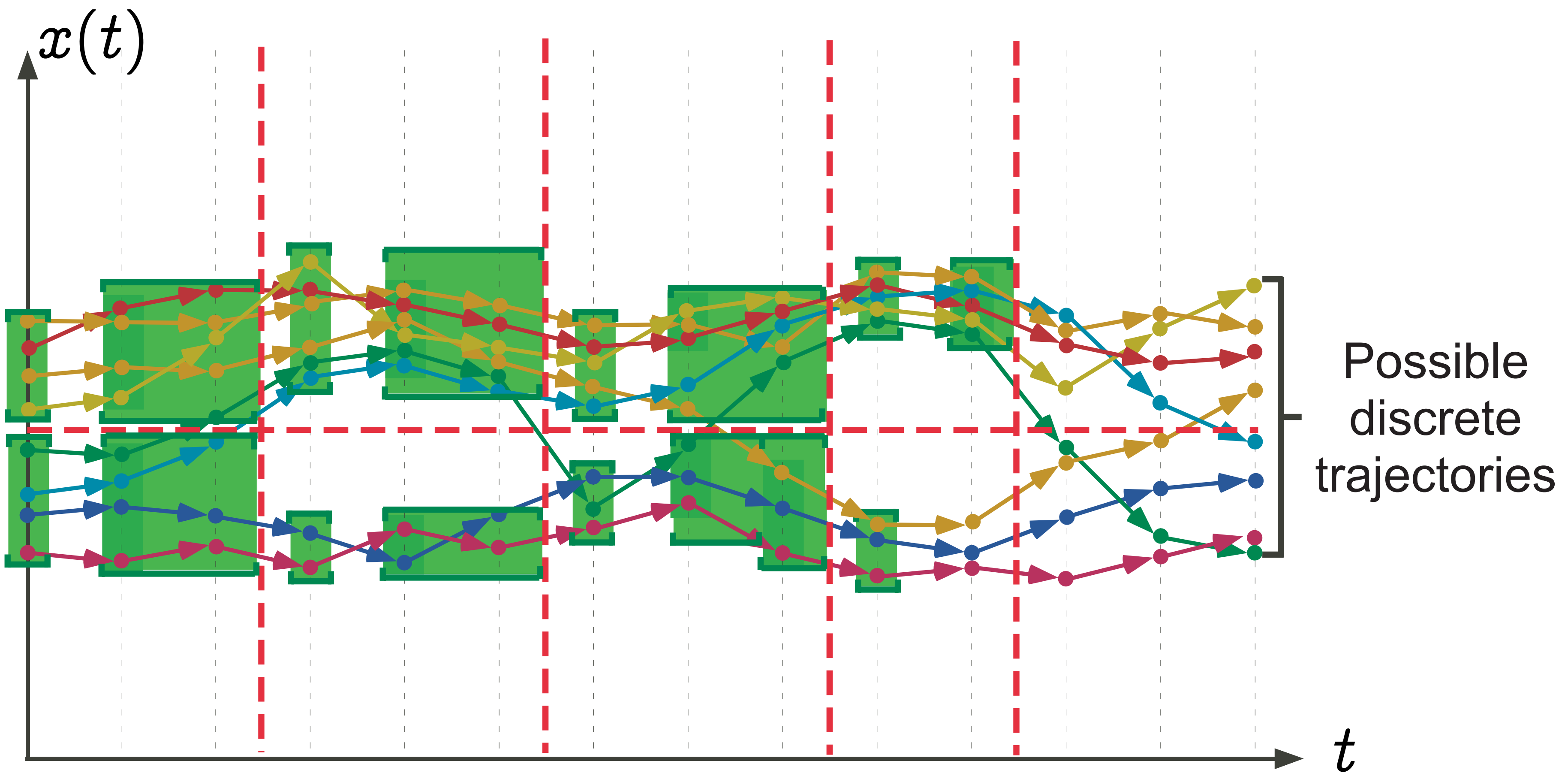
Graphic example: partitionned upward iteration with widening



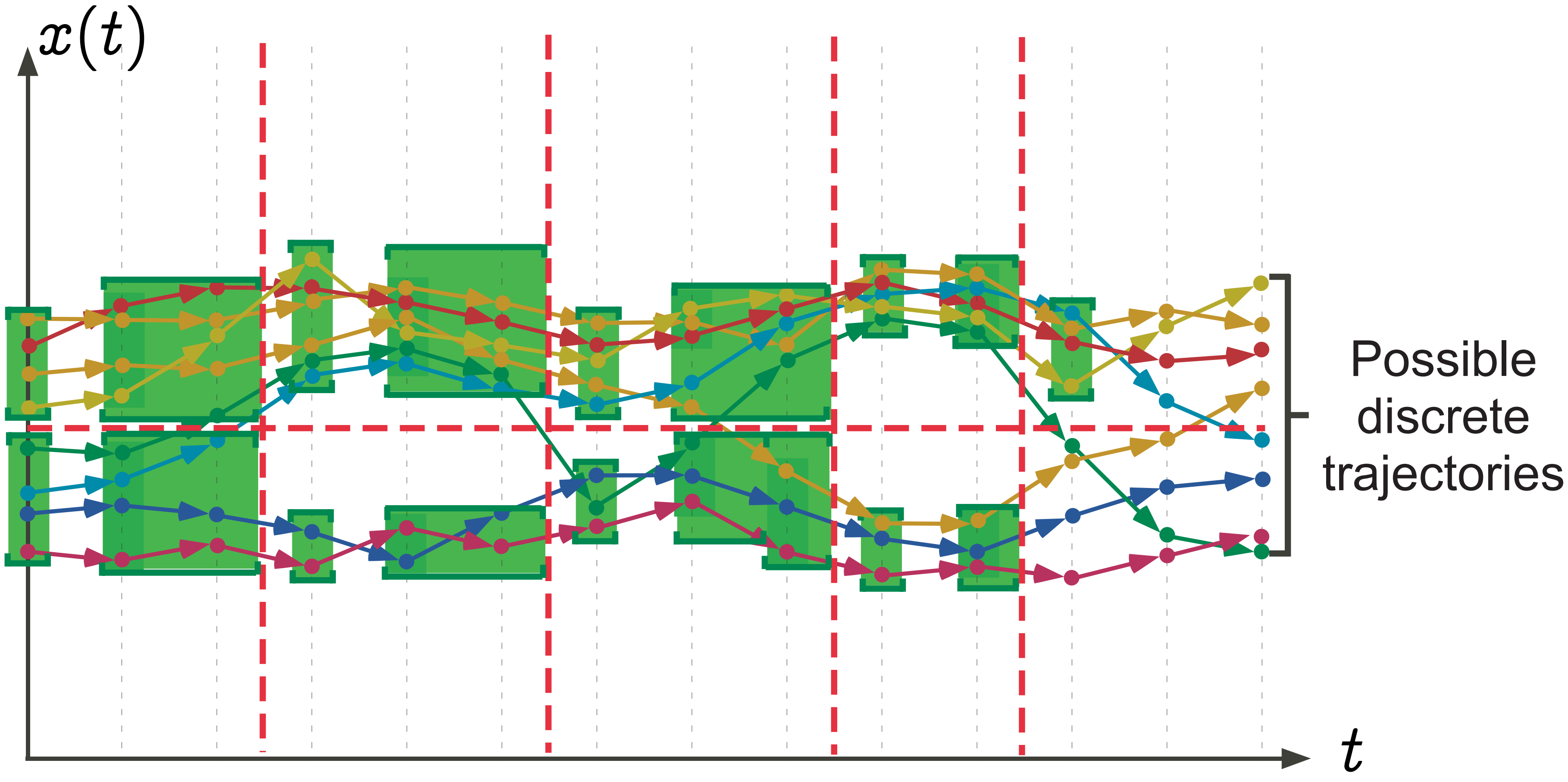
Graphic example: partitionned upward iteration with widening



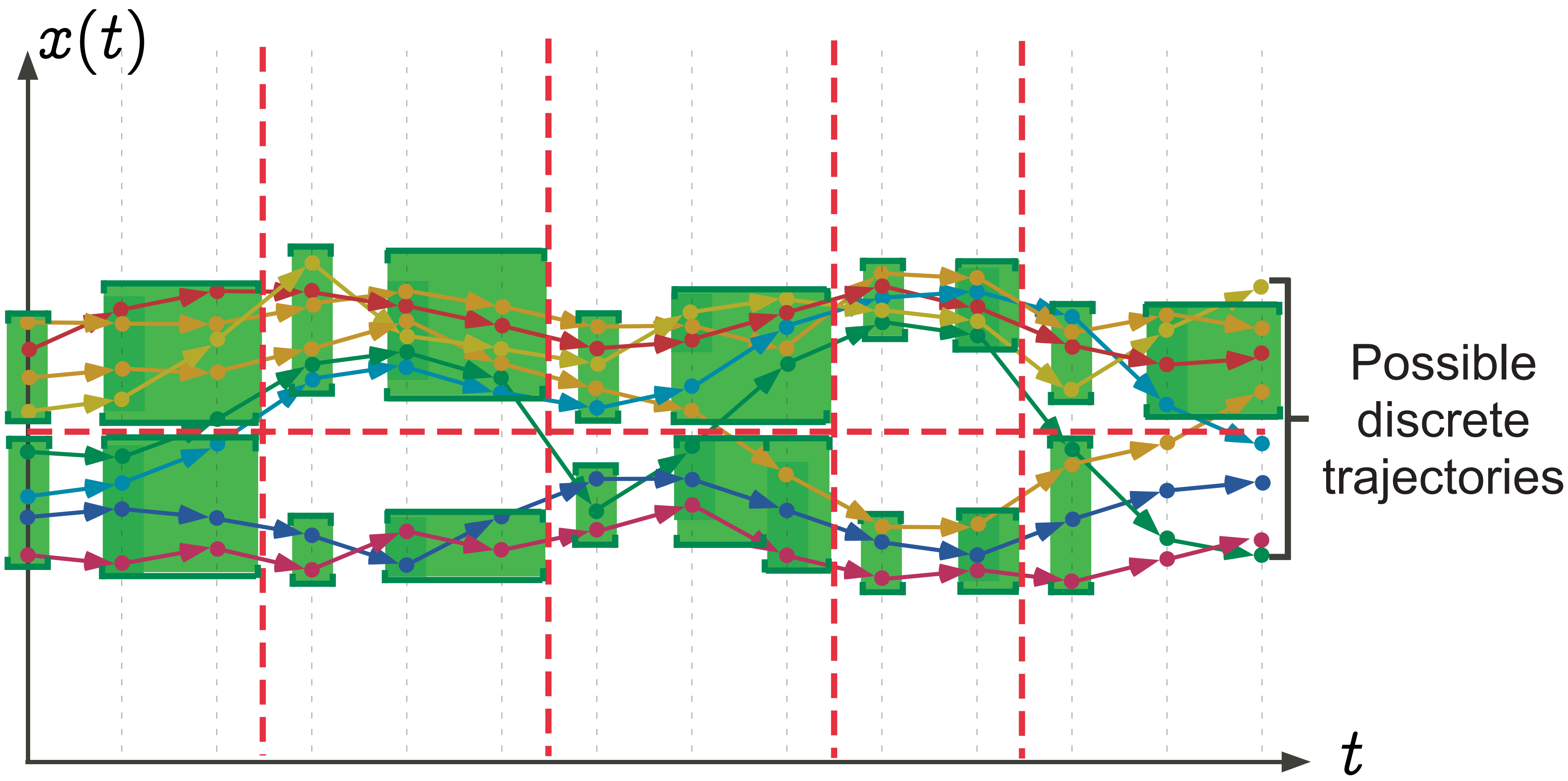
Graphic example: partitionned upward iteration with widening



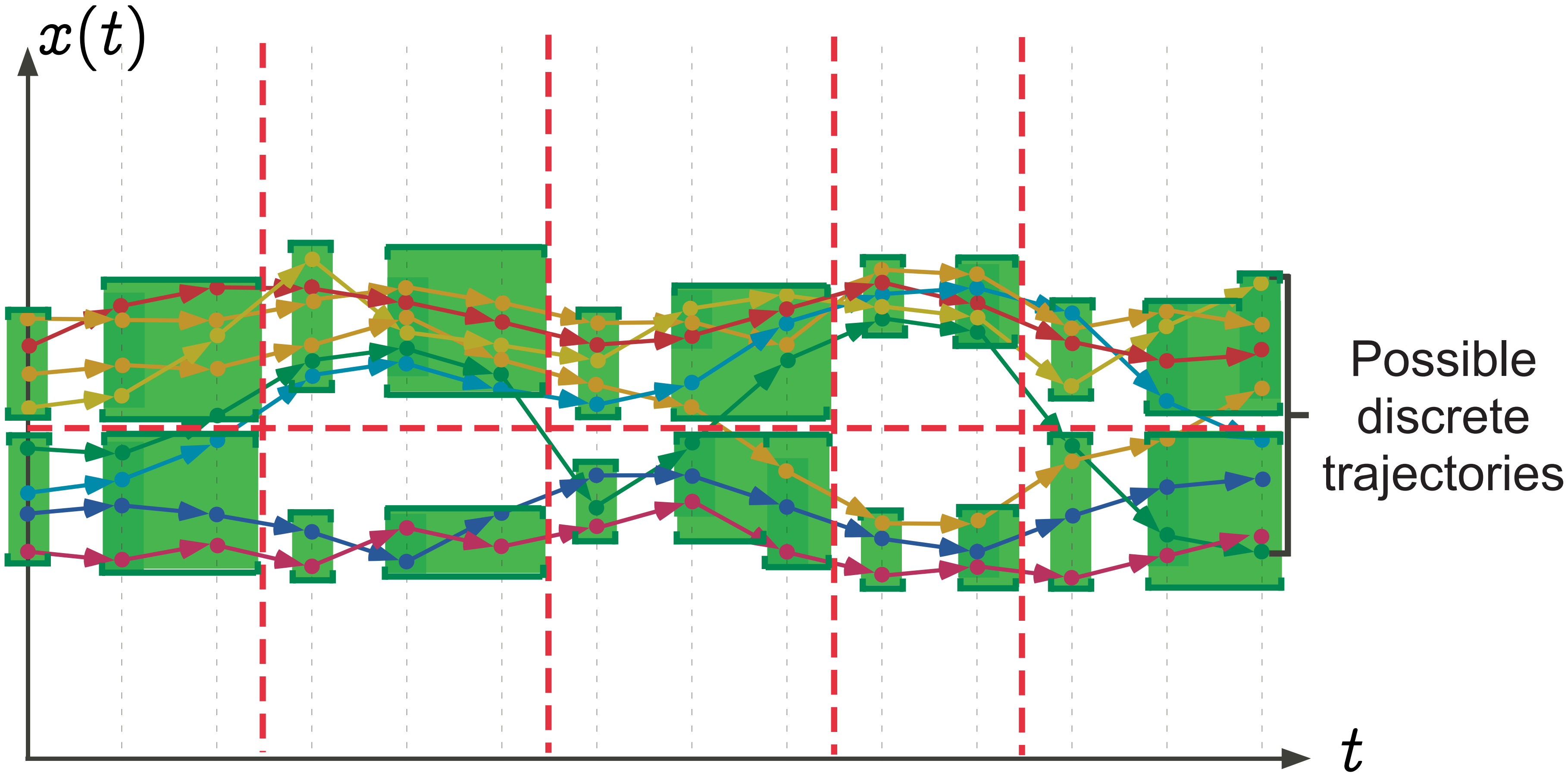
Graphic example: partitionned upward iteration with widening



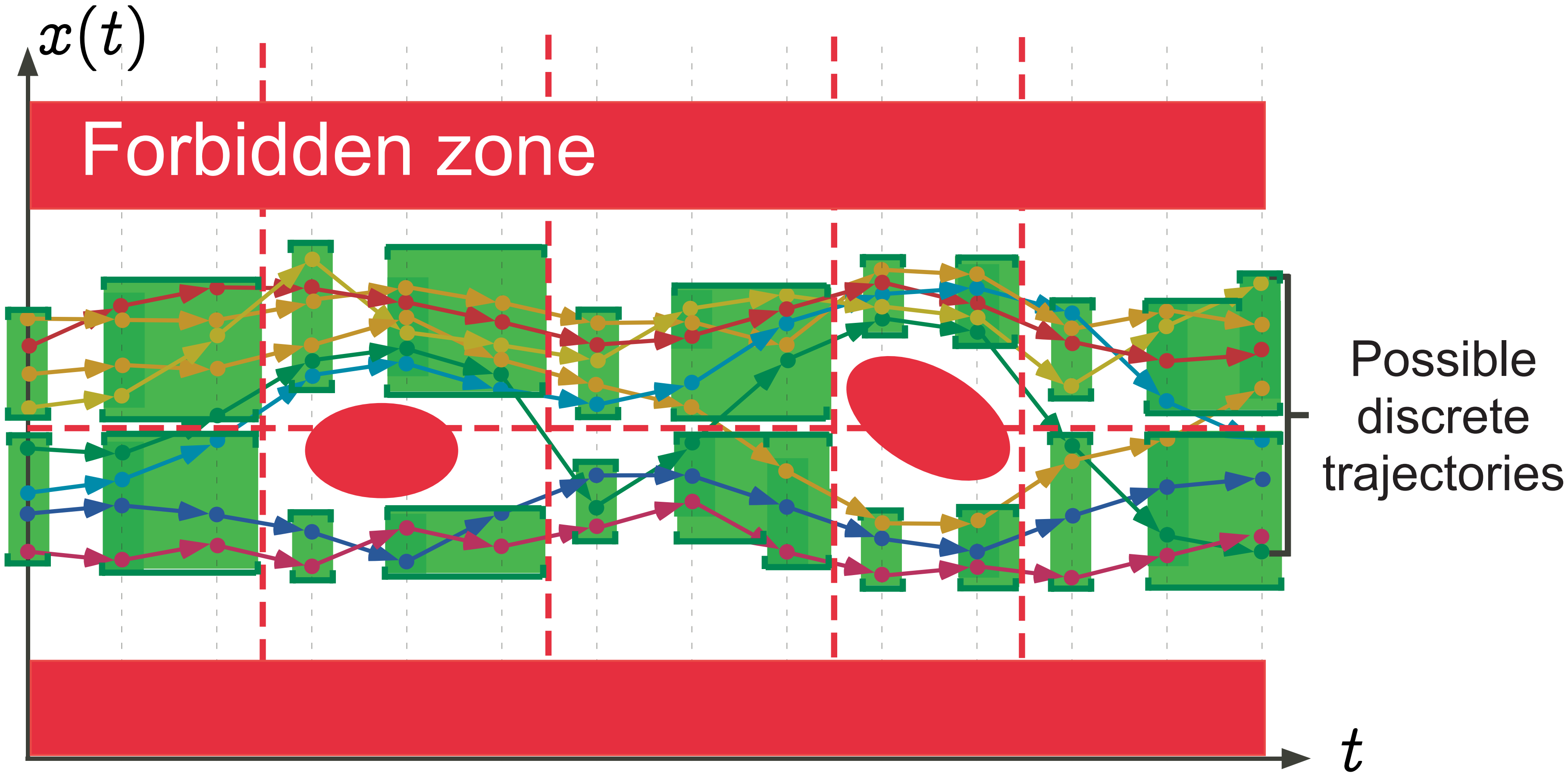
Graphic example: partitionned upward iteration with widening



Graphic example: partitionned upward iteration with widening



Graphic example: safety verification



Interval widening with threshold set

– The threshold set T is a finite set of numbers (plus $+\infty$ and $-\infty$),

– $$[a, b] \nabla_T [a', b'] = [\begin{array}{l} \mathit{if}\ a' < a\ \mathit{then}\ \max\{\ell \in T \mid \ell \leq a'\} \\ \quad \mathit{else}\ a, \\ \mathit{if}\ b' > b\ \mathit{then}\ \min\{h \in T \mid h \geq b'\} \\ \quad \mathit{else}\ b] \end{array}.$$

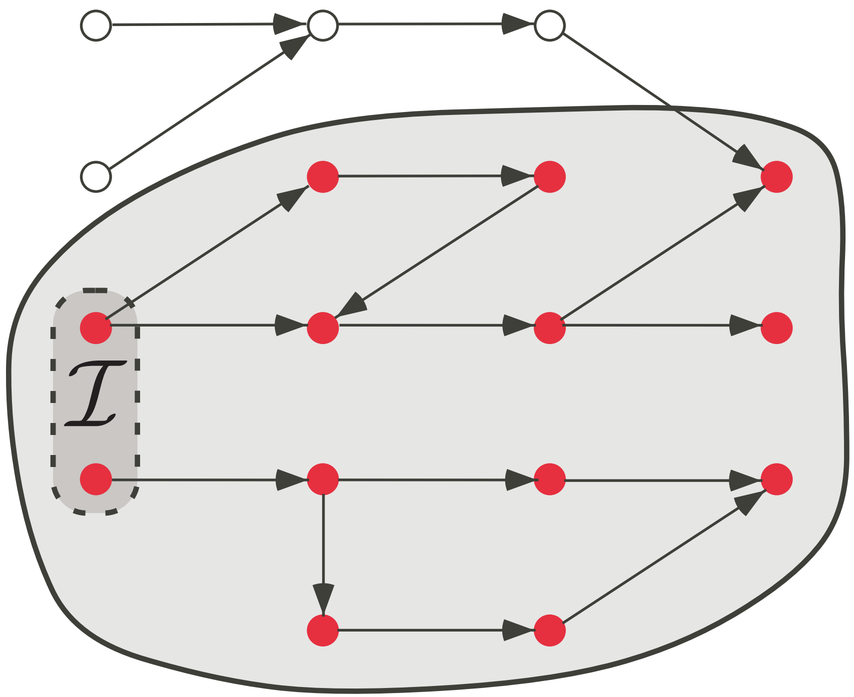
– Examples (intervals):

- sign analysis: $T = \{-\infty, 0, +\infty\}$;
- strict sign analysis: $T = \{-\infty, -1, 0, +1, +\infty\}$;

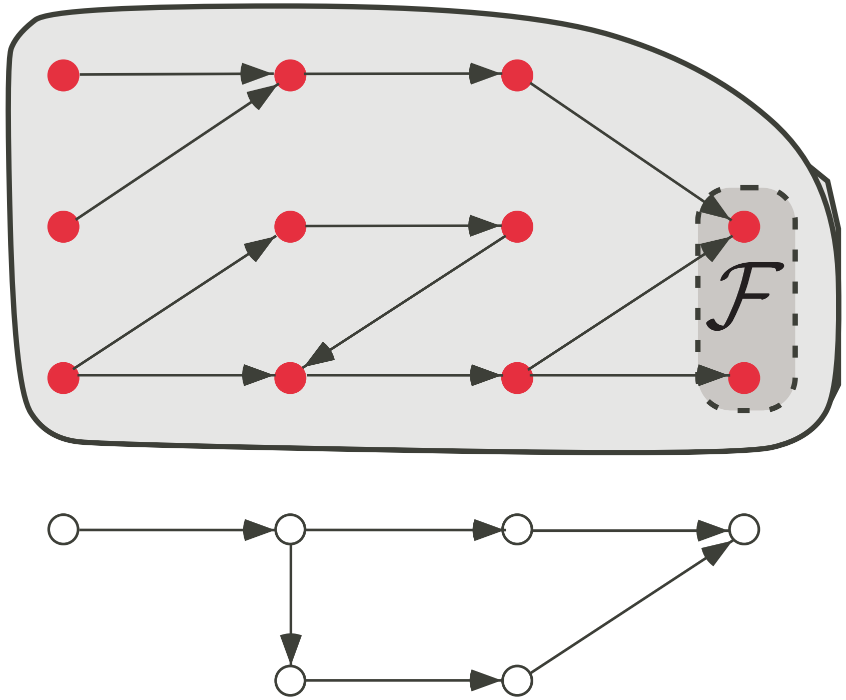
– T is a parameter of the analysis.

Combinations of abstractions

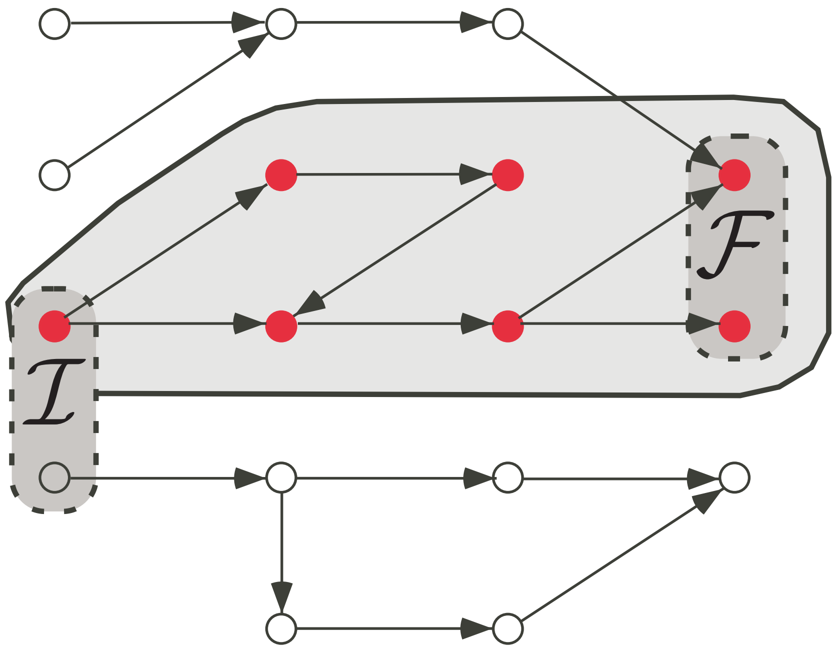
Forward/reachability analysis



Backward/ancestry analysis



Iterated forward/backward analysis



Example of iterated forward/backward analysis

Arithmetical mean of two integers x and y:

```
{x>=y}
  while (x <> y) do
    {x>=y+2}
      x := x - 1;
    {x>=y+1}
      y := y + 1
    {x>=y}
  od
{x=y}
```

Necessarily $x \geq y$ for proper termination

Example of iterated forward/backward analysis

Adding an auxiliary counter `k` decremented in the loop body and asserted to be null on loop exit:

```
{x=y+2k,x>=y}
  while (x <> y) do
    {x=y+2k,x>=y+2}
      k := k - 1;
    {x=y+2k+2,x>=y+2}
      x := x - 1;
    {x=y+2k+1,x>=y+1}
      y := y + 1
    {x=y+2k,x>=y}
  od
{x=y,k=0}
  assume (k = 0)
{x=y,k=0}
```

Moreover the difference of `x` and `y` must be even for proper termination

Applications of abstract interpretation

Theoretical applications of abstract interpretation

- **Static Program Analysis** [POPL '77,78,79] inluding **Data-flow Analysis** [POPL '79,00], **Set-based Analysis** [FPCA '95], etc
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92, TCS 277(1–2) 2002]
- **Typing** [POPL '97]
- **Model Checking** [POPL '00]
- **Program Transformation** [POPL '02]
- **Software watermarking** [POPL '04]

Industrial applications of abstract interpretation

– **Program analysis and manipulation**: a small rate of false alarms is acceptable
  - AiT: worst case execution time[11]
  - StackAnalyzer: stack usage analysis[11]
– **Program verification**: no false alarms is acceptable
  - TVLA: A system for generating abstract interpreters
  - Astrée: verification of absence of run-time errors[11]

[11] applied to the primary flight control software of the Airbus A340/600 and A380 fly-by-wire systems

Bibliography

Seminal papers

– Patrick Cousot & Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4th Symp. on Principles of Programming Languages, pages 238—252. ACM Press, 1977.

– Patrick Cousot & Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In 5th Symp. on Principles of Programming Languages, pages 84—97. ACM Press, 1978.

– Patrick Cousot & Radhia Cousot. Systematic design of program analysis frameworks. In 6th Symp. on Principles of Programming Languages pages 269—282. ACM Press, 1979.

Recent surveys

– Patrick Cousot. Interprétation abstraite. Technique et Science Informatique, Vol. 19, Nb 1-2-3. Janvier 2000, Hermès, Paris, France. pp. 155-164.

– Patrick Cousot. Abstract Interpretation Based Formal Methods and Future Challenges. In Informatics, 10 Years Back — 10 Years Ahead, R. Wilhelm (Ed.), LNCS 2000, pp. 138-156, 2001.

– Patrick Cousot & Radhia Cousot. Abstract Interpretation Based Verification of Embedded Software: Problems and Perspectives. In Proc. 1st Int. Workshop on Embedded Software, EMSOFT 2001, T.A. Henzinger & C.M. Kirsch (Eds.), LNCS 2211, pp. 97–113. Springer, 2001.

Anticipated Content of Course 16.399: Abstract Interpretation

– Today : an informal overview of abstract interpretation;

– The software verification problem (undecidability, complexity, test, simulation, specification, formal methods (deductive methods, model-checking, static analysis) and their limitations, intuitive notion of approximation, false alarms);

– Mathematical foundations (naive set theory, first order classical logic, lattice theory, fixpoints);

– Semantics of programming languages (abstract syntax, operational semantics, inductive definitions, example of a simple imperative language, grammar and interpretor of the language, trace semantics);
– Program specification and manual proofs (safety properties, Hoare logic, predicate transformers, liveness properties, linear-time temporal logic (LTL));
– Order-theoretic approximation (abstraction, closures, Galois connections, fixpoint abstraction, widening, narrowing, reduced product, absence of best approximation, refinement);

- Principle of static analysis by abstract interpretation (reachability analysis of a transition system, finite approximation, model-checking, infinite approximation, static analysis, program-based versus language-based analysis, limitations of finite approximations);
- Design of a generic structural abstract interpreter (collecting semantics, non-relational and relational analysis, convergence acceleration by wideing/narrowing);
- Static analysis (forward reachability analysis, backward analysis, iterated forward/backward analysis, inevitability analysis, termination)