

Diseño y Verificación de Programas Concurrentes

Escuela de Informática - CACIC 2021

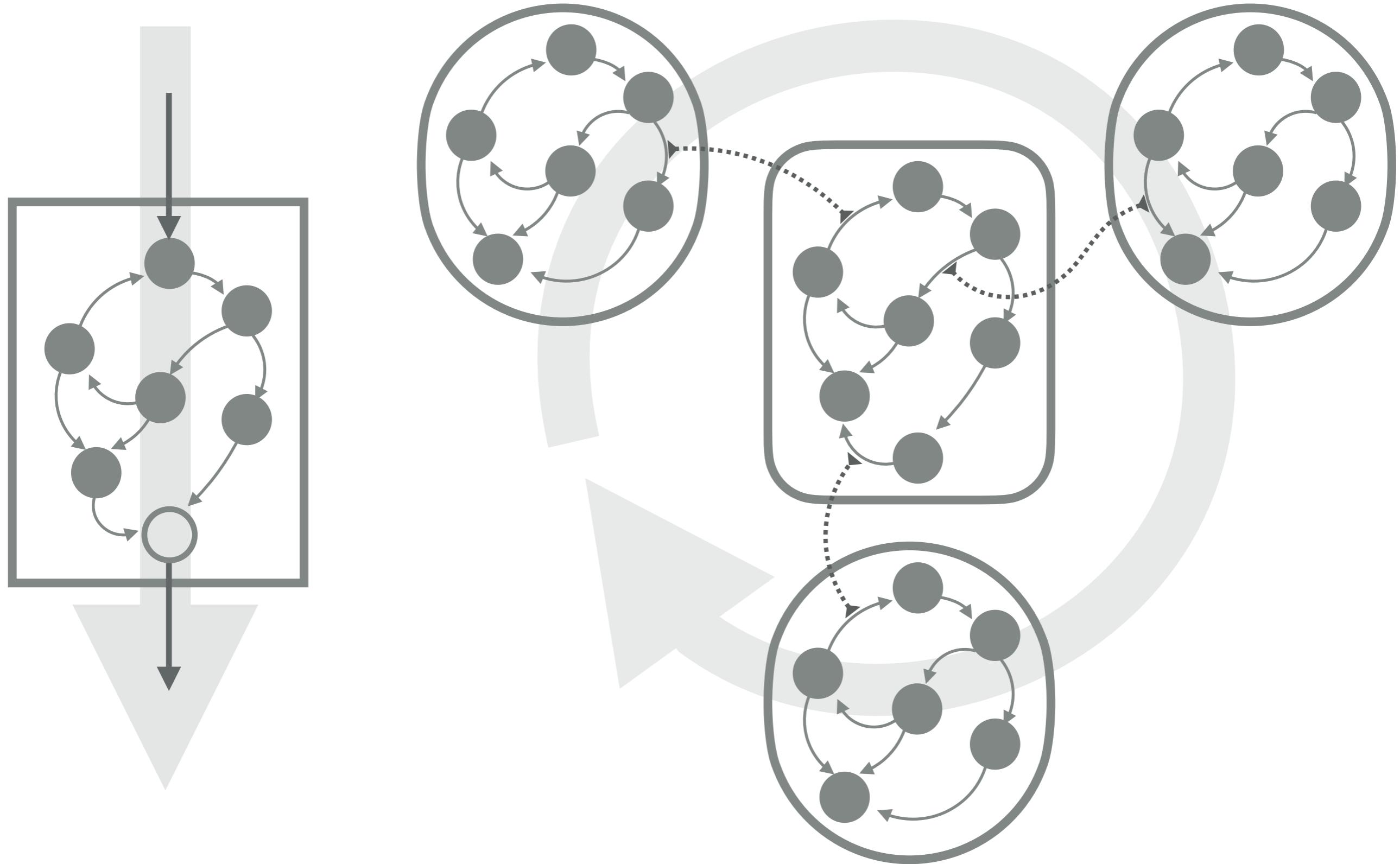
ANALIZANDO MODELOS DE SISTEMAS REACTIVOS

INFORMACIÓN Y MODELOS

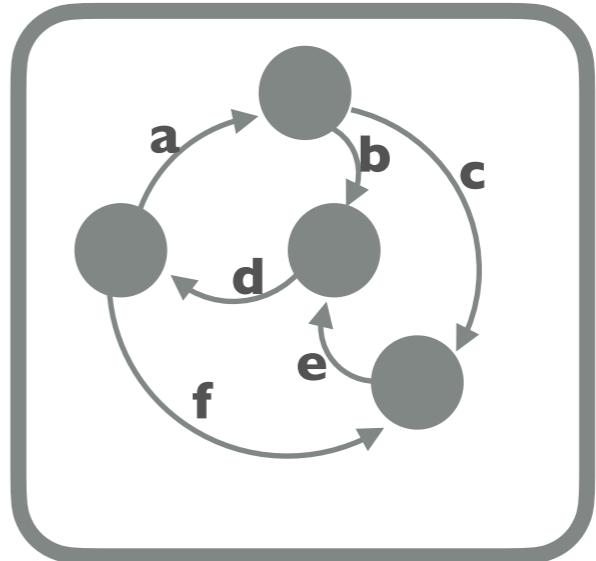
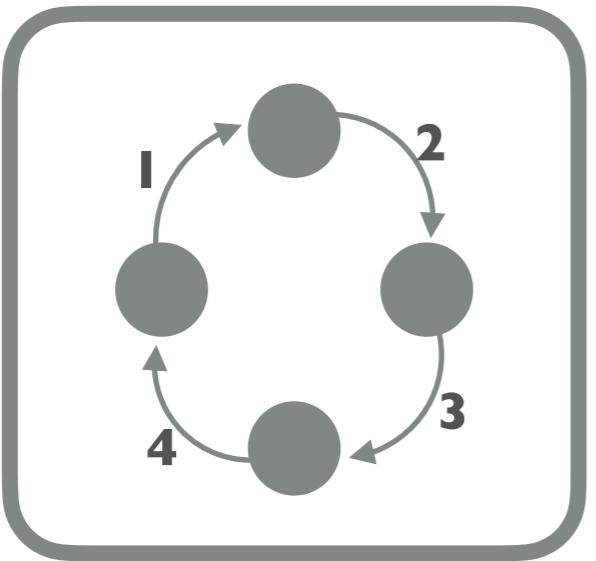


Estancia la Guitarra - Córdoba

COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS

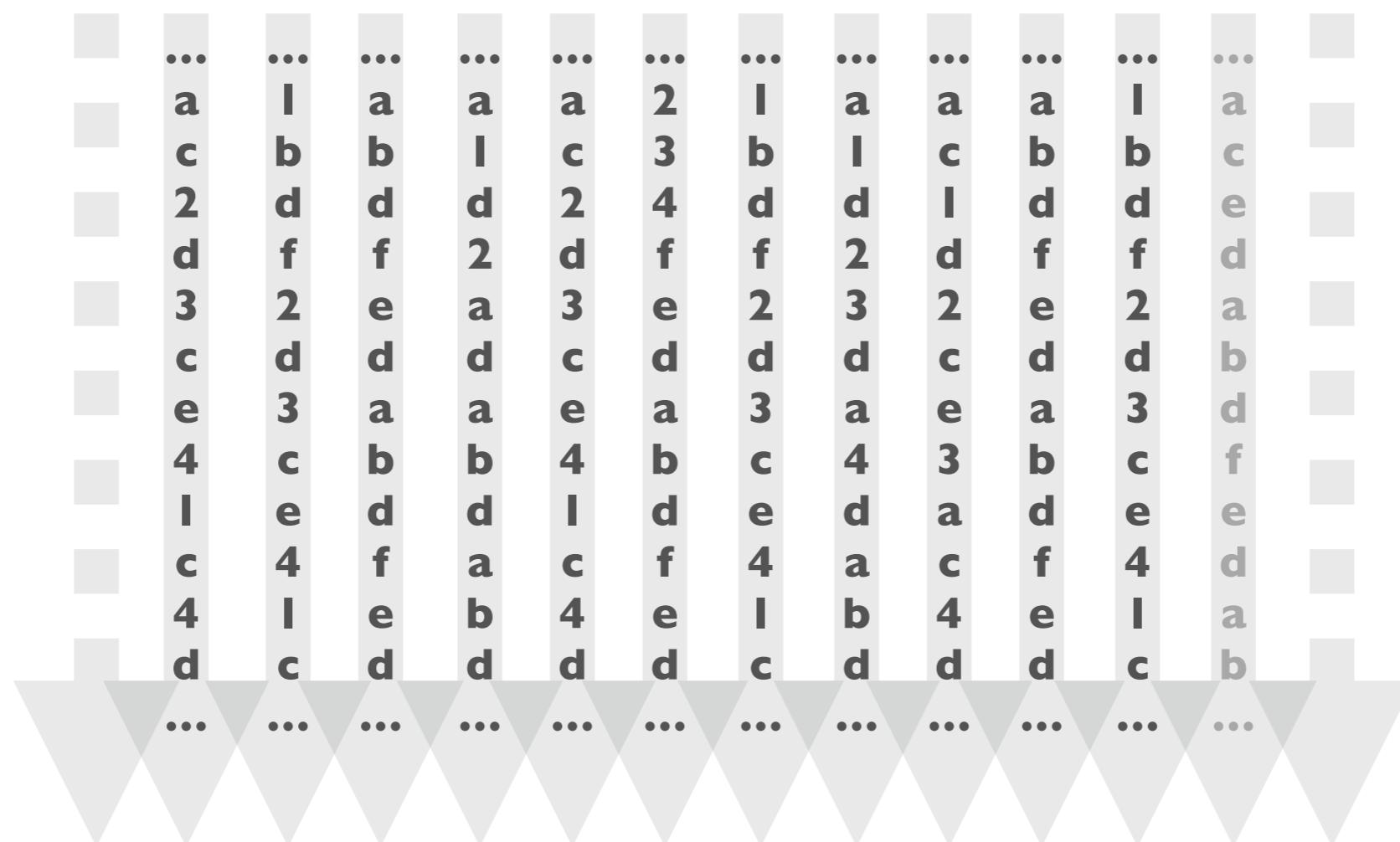
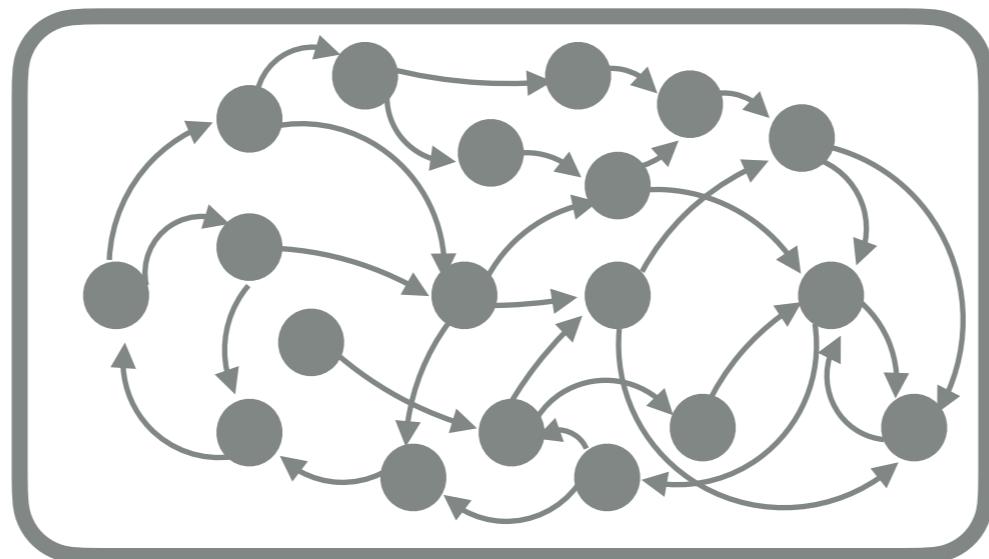


COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS

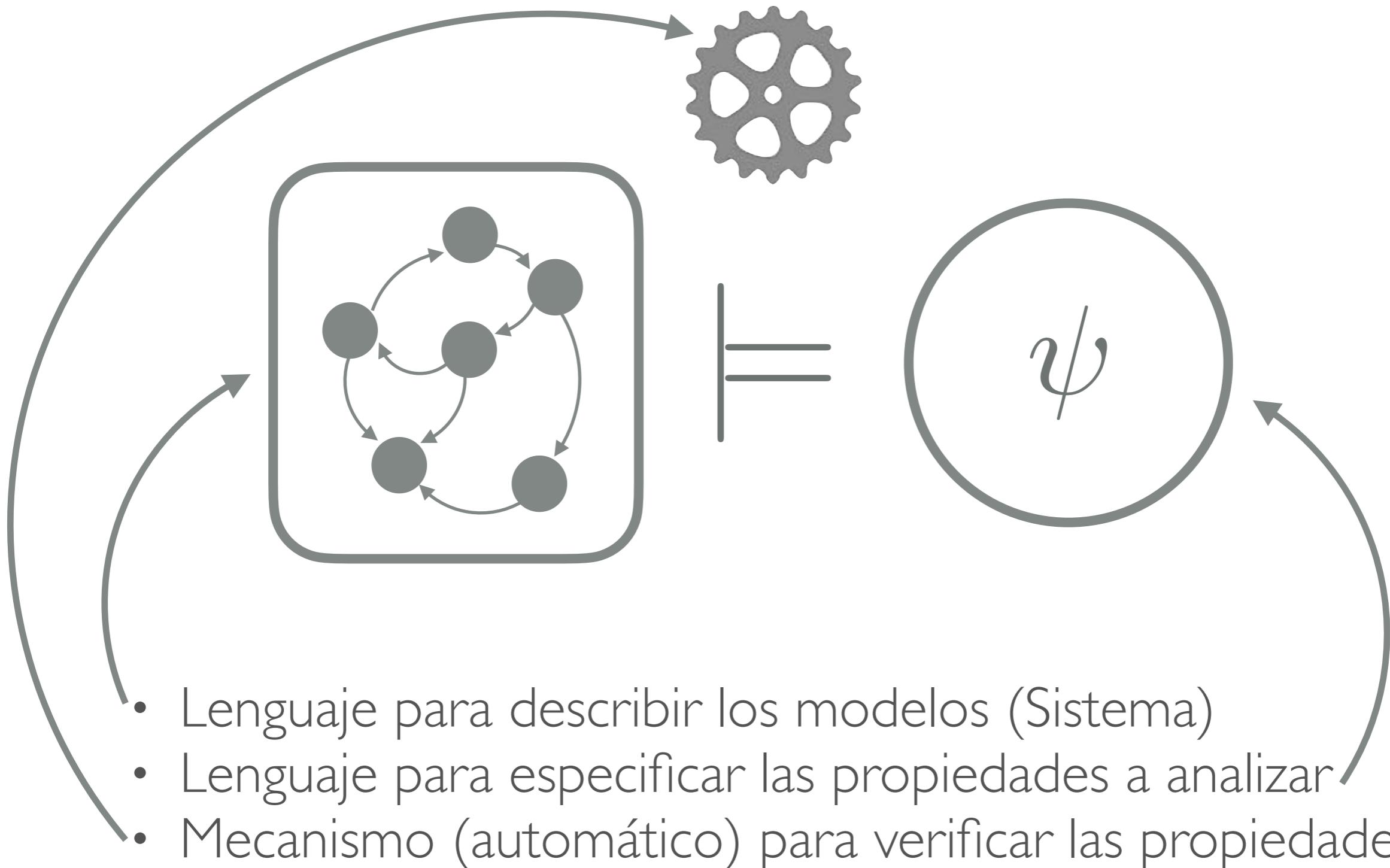


... I 2 3 4 I 2 3 4 I 2 3 4 ...

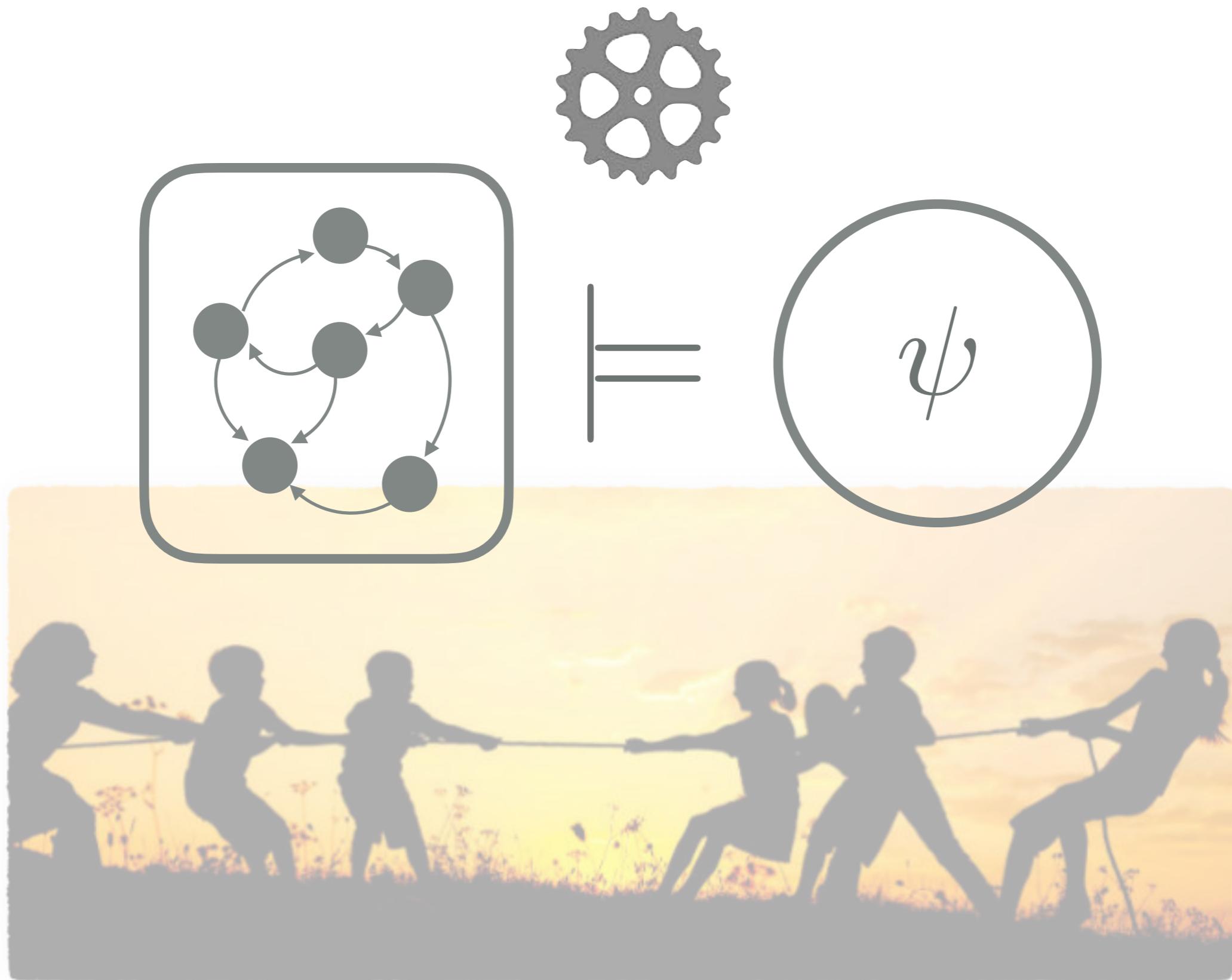
COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS



COMO ANALIZAR (VERIFICAR PROPIEDADES) DE SISTEMAS REACTIVOS



EXPRESIVIDAD VS. EFICIENCIA (AUTOMATIZACIÓN)



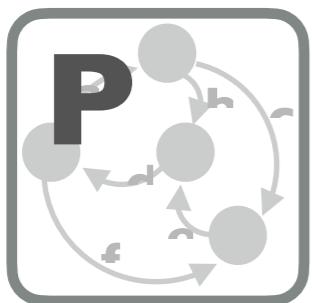
LENGUAJES OPERACIONALES VS. DECLARATIVOS

En general preferimos notaciones **operacionales** para la descripción de sistemas.

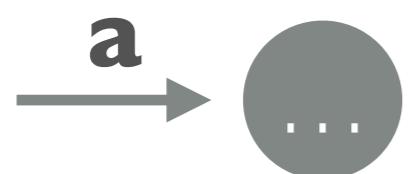
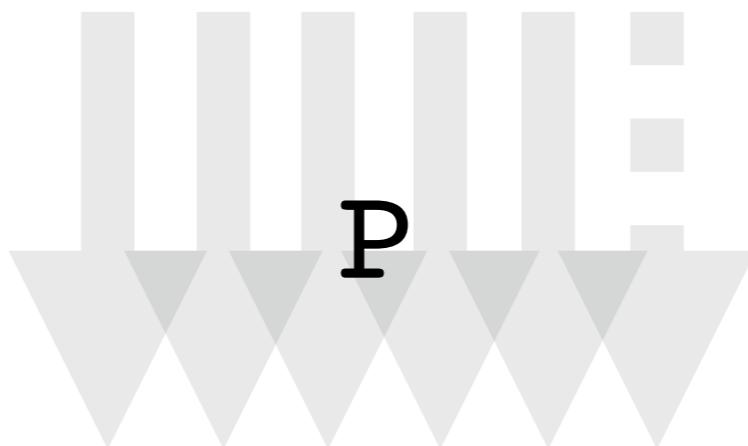


En general preferimos lenguajes **declarativos** para la especificación de propiedades de sistemas.

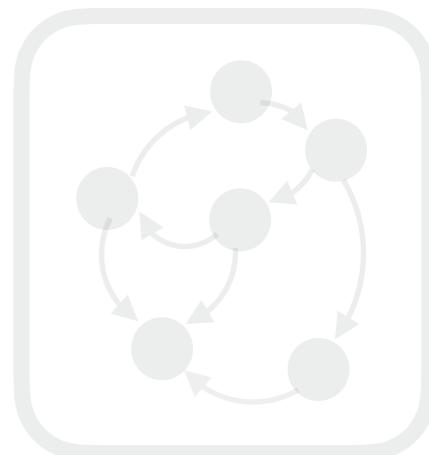
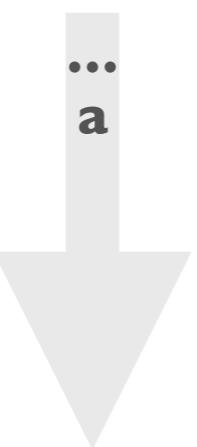
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



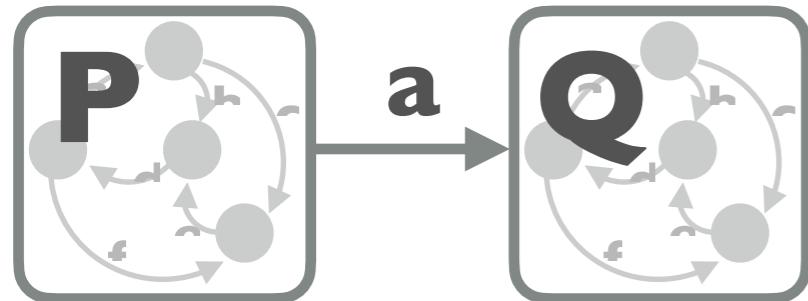
$P = (\dots).$



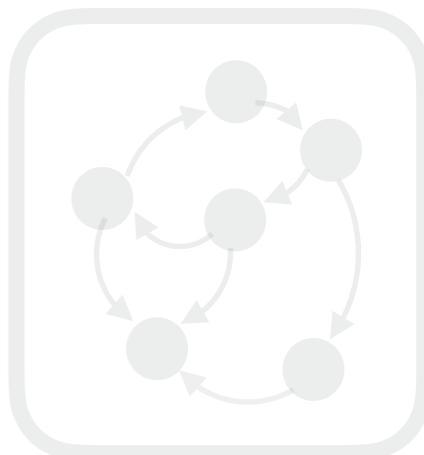
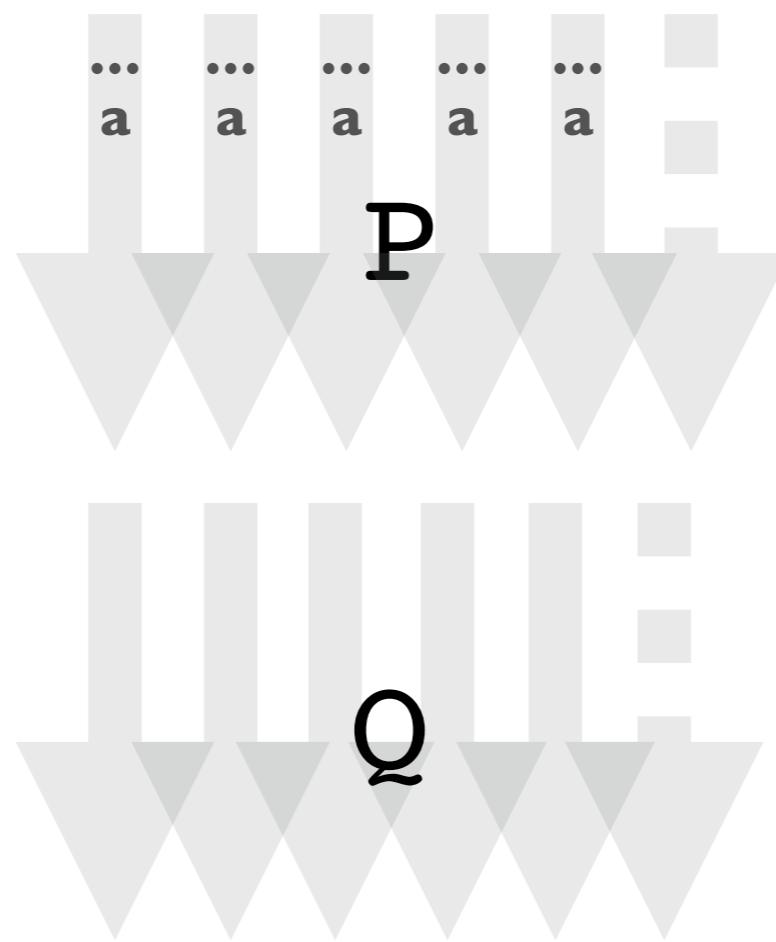
$a \rightarrow \dots$



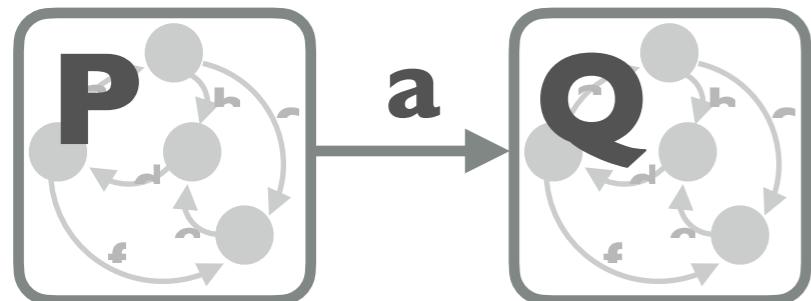
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



$P = (\dots a \rightarrow Q), Q = (\dots)$



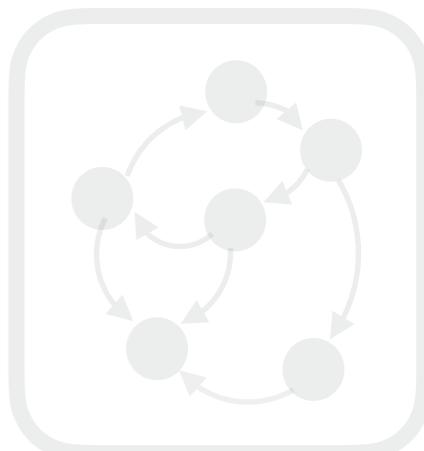
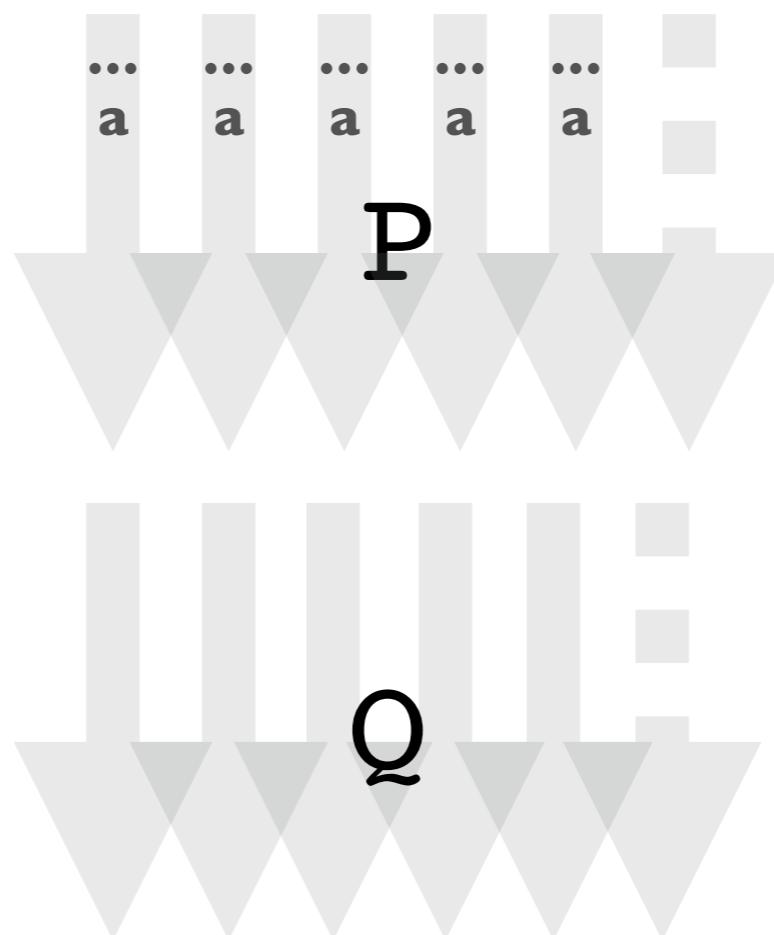
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



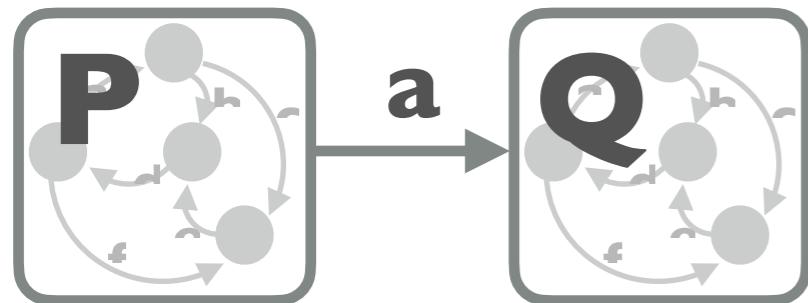
$P = (\dots a \rightarrow Q), Q = (\dots)$

Recursión

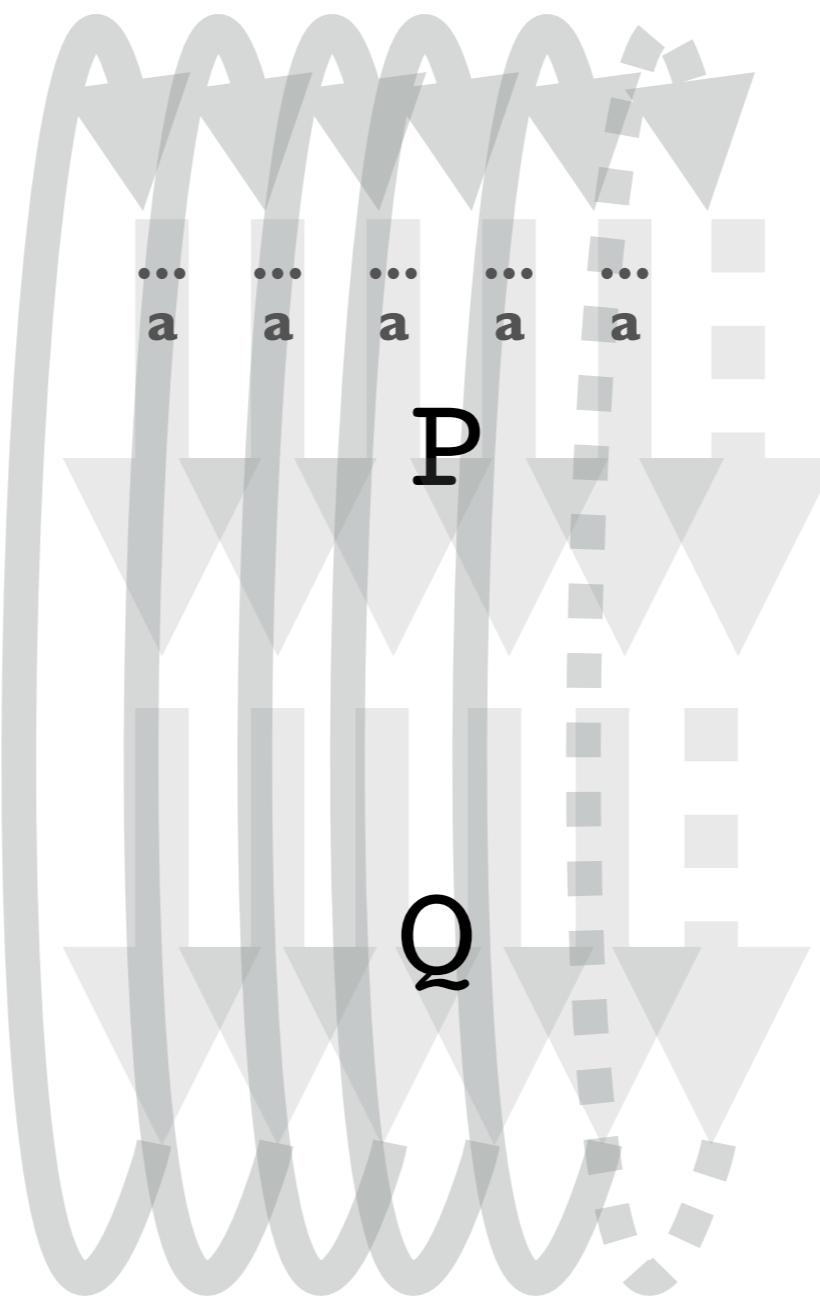
$Q = (\dots \rightarrow P).$



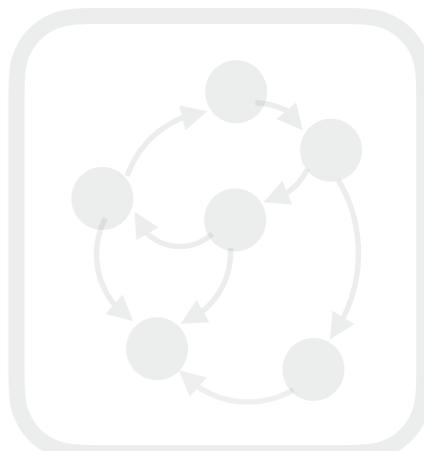
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



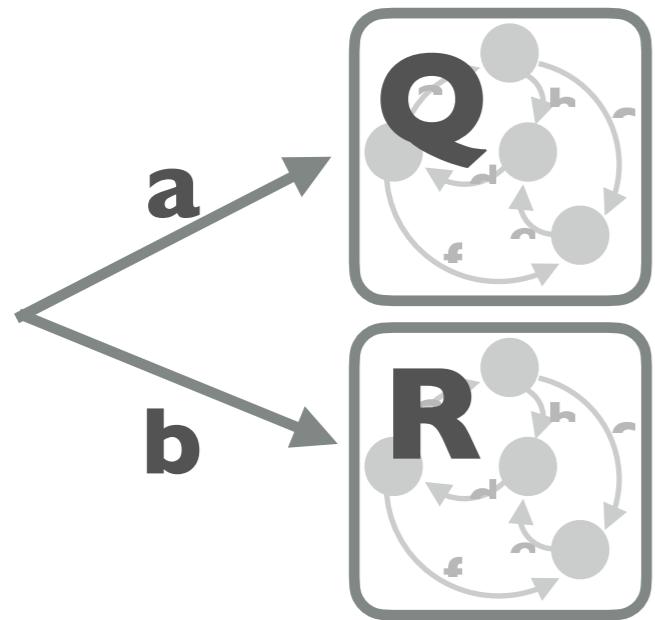
$P = (\dots a \rightarrow Q), Q = (\dots)$



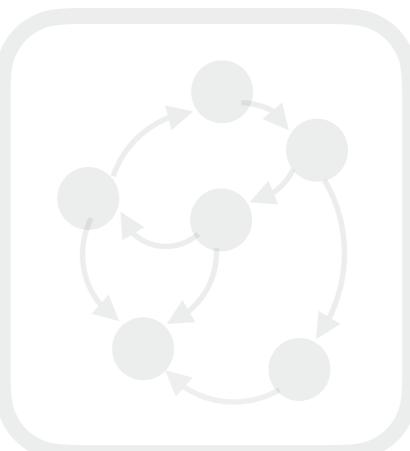
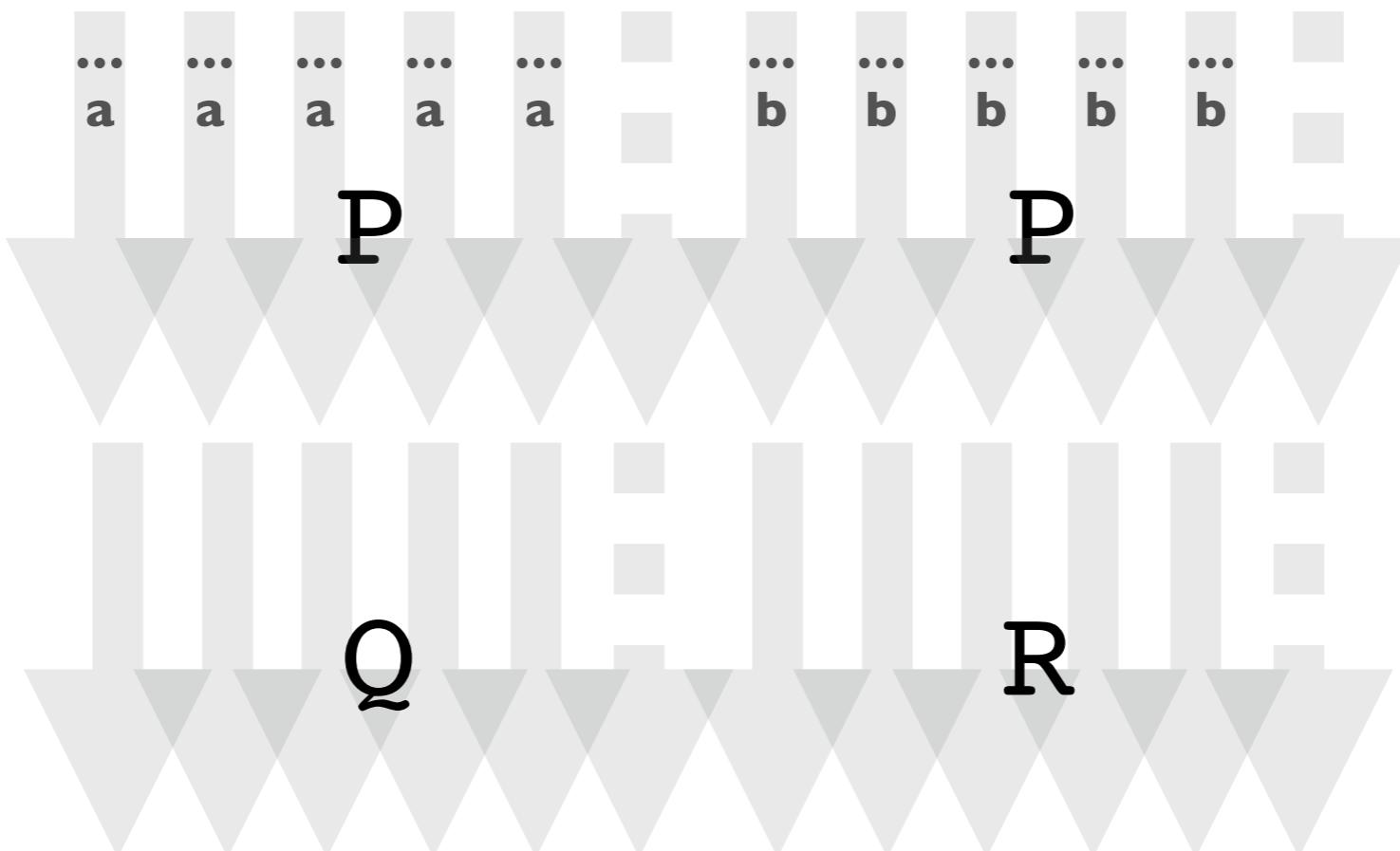
Recursión
 $Q = (\dots \rightarrow P).$



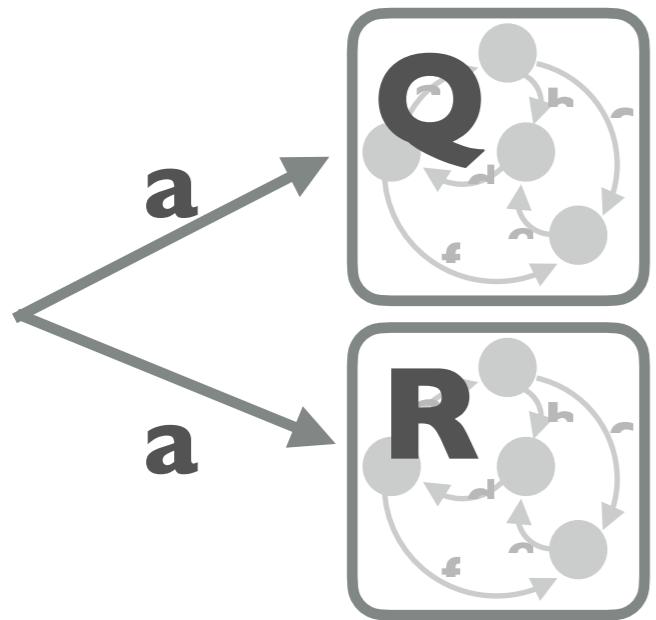
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



... ($a \rightarrow Q$ | $b \rightarrow R$)

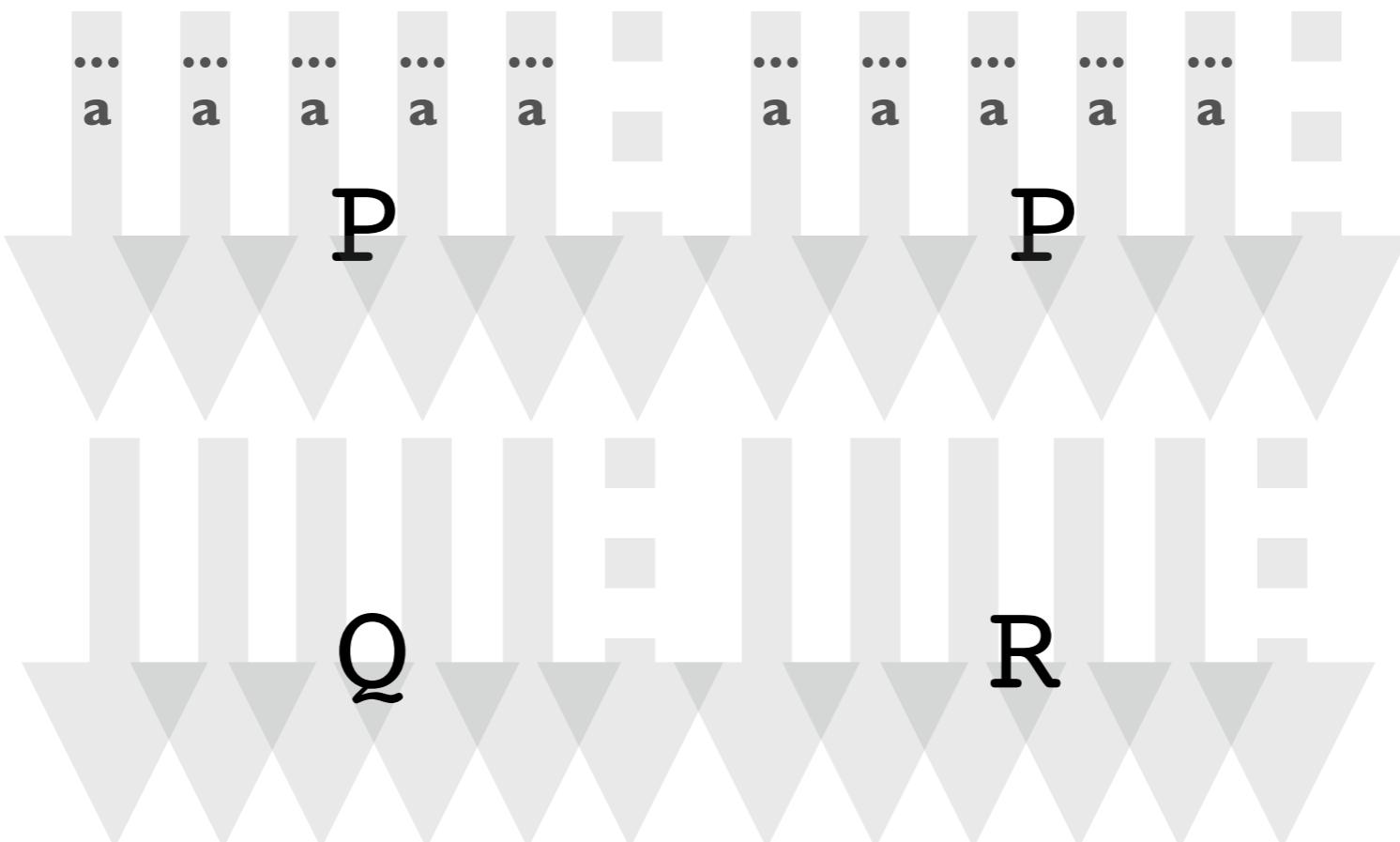


LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



... ($a \rightarrow Q$ | $b \rightarrow R$)

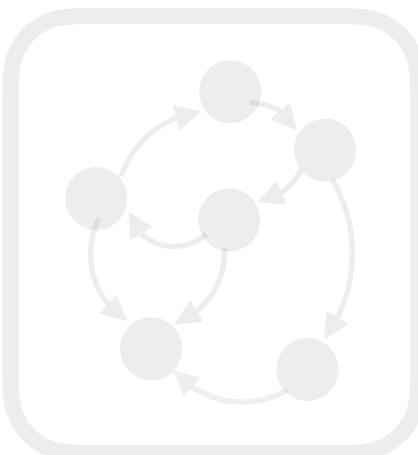
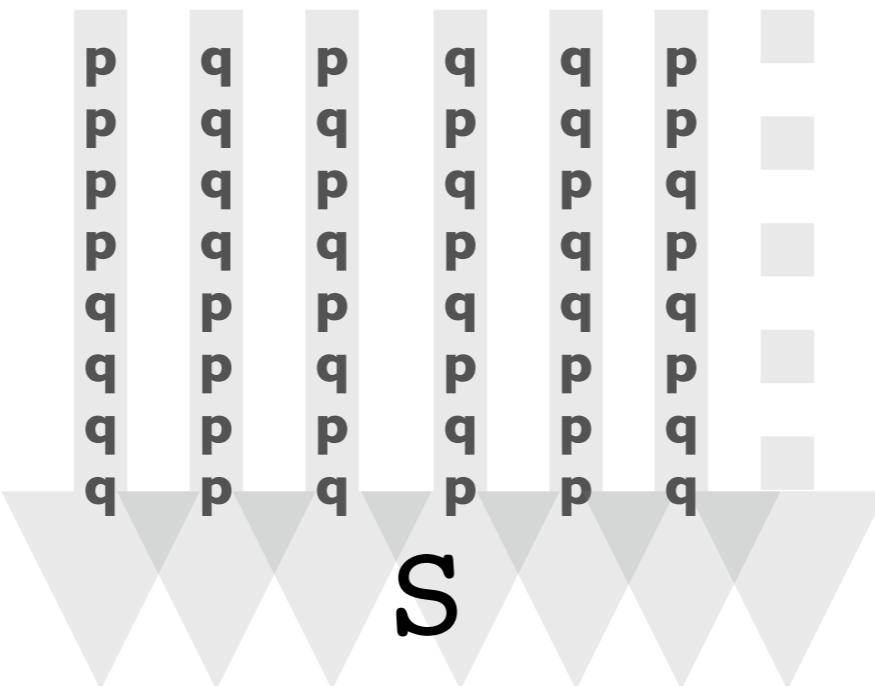
No Determinismo
| $a \rightarrow R$).



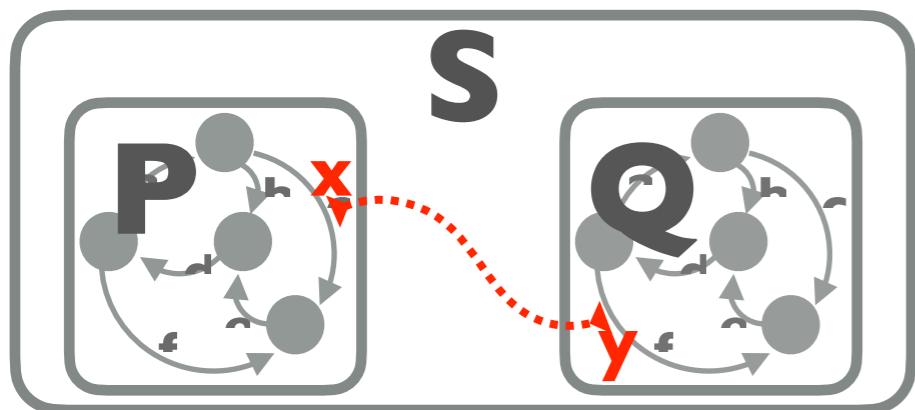
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



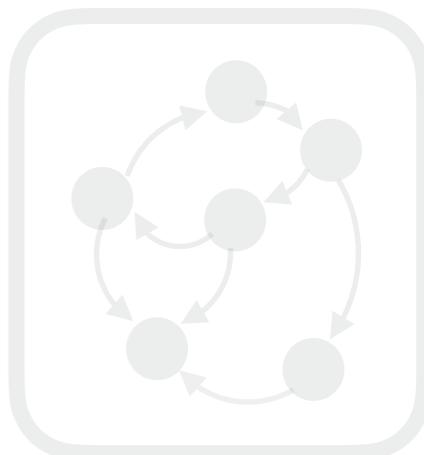
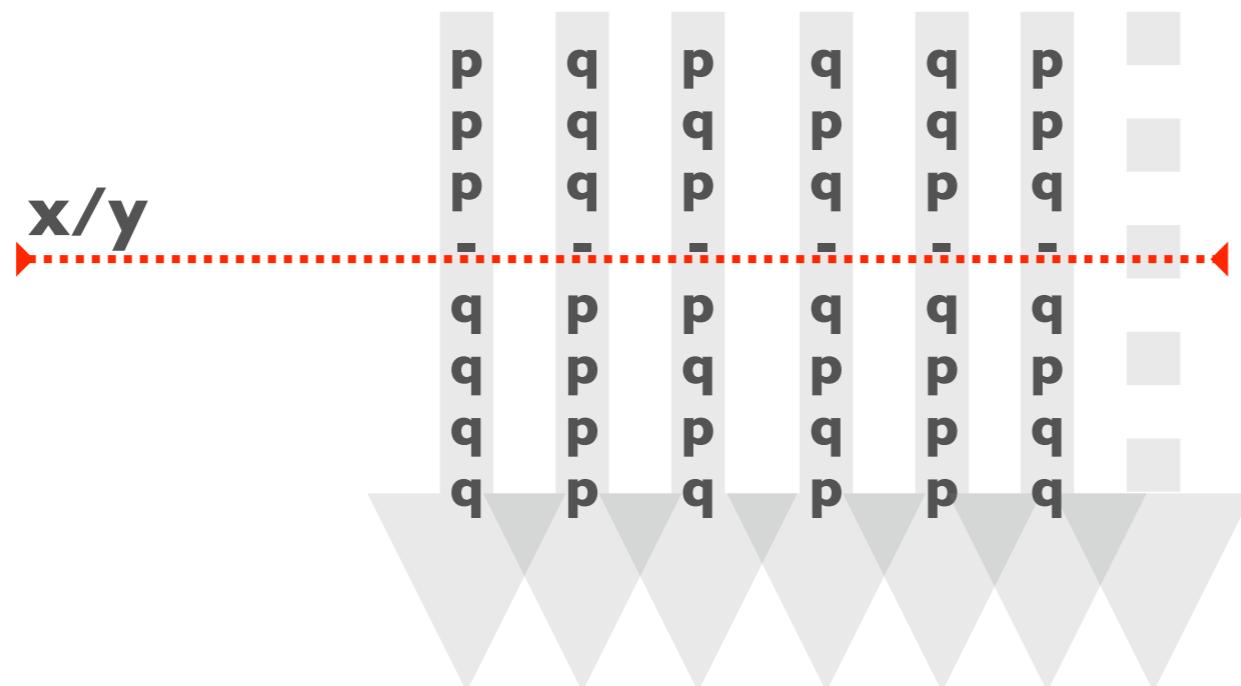
$$|| S = (P \parallel Q) .$$



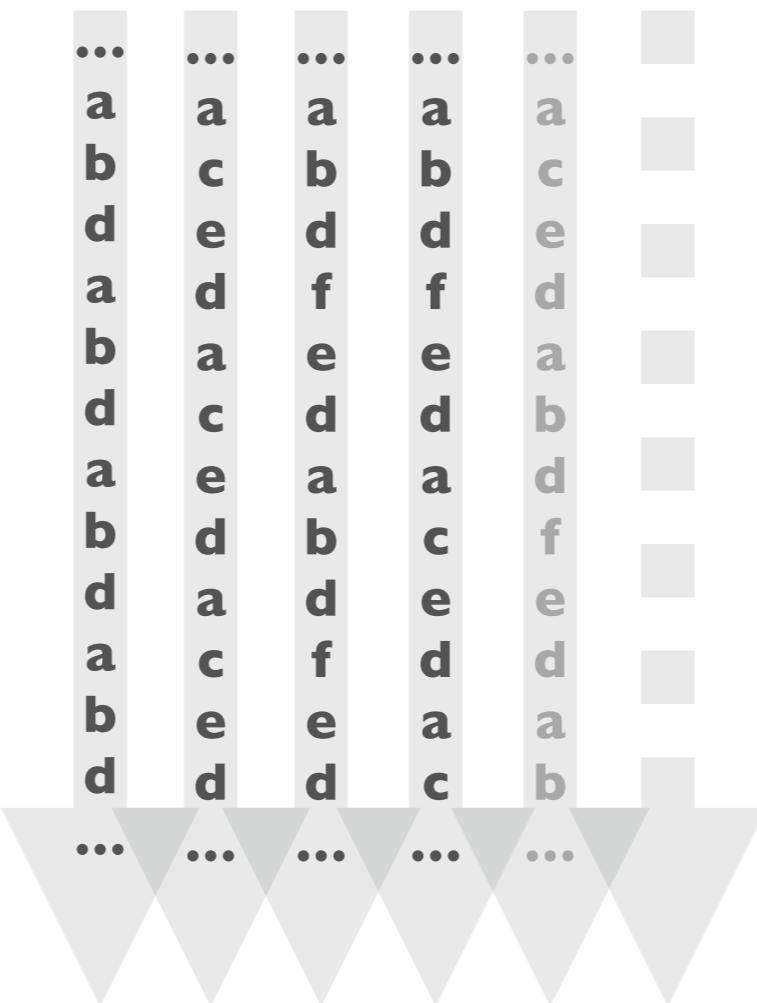
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



$$|| S = (P \parallel Q) \{P.x/Q.y\} .$$

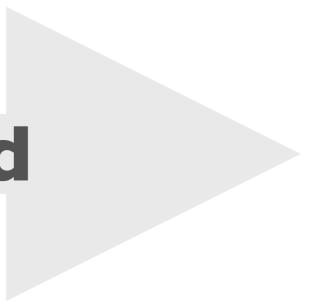


CÓMO CAPTURAR PROPIEDADES DETRAZAS (INFINITAS)



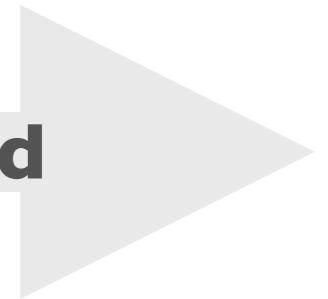
CÓMO CAPTURAR PROPIEDADES DE TRAZAS (INFINITAS)

a b d a b d a b d d a a b d a d a d d



CÓMO CAPTURAR PROPIEDADES DE TRAZAS (INFINITAS)

a b d a b d a b d d a a b d a d a d d

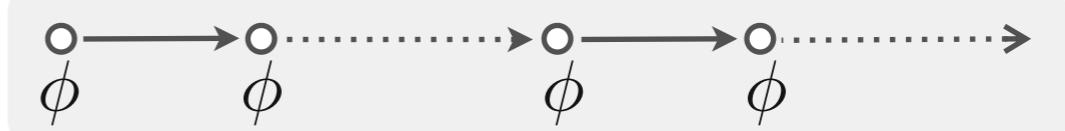


- Next:
 - Globally
 - Finally
 - Until

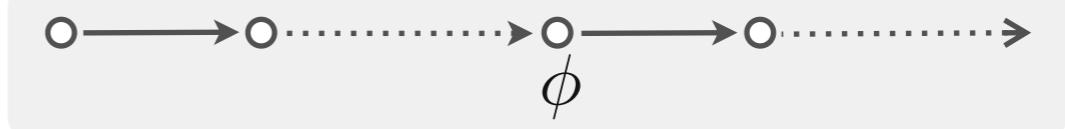
X ϕ



G φ



F ϕ

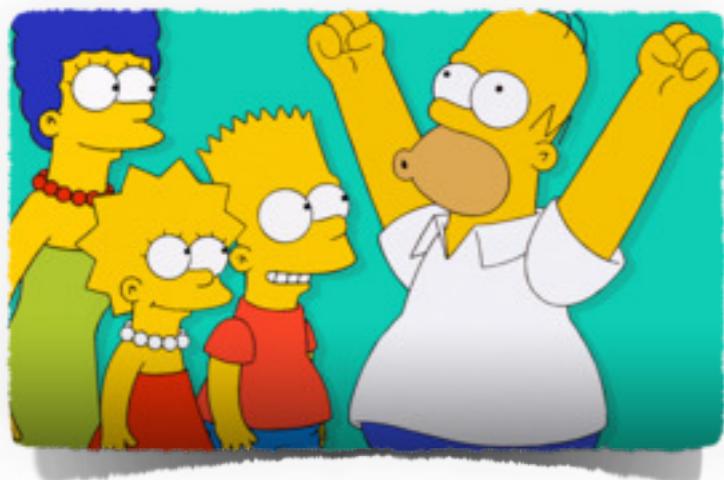


U ϕ

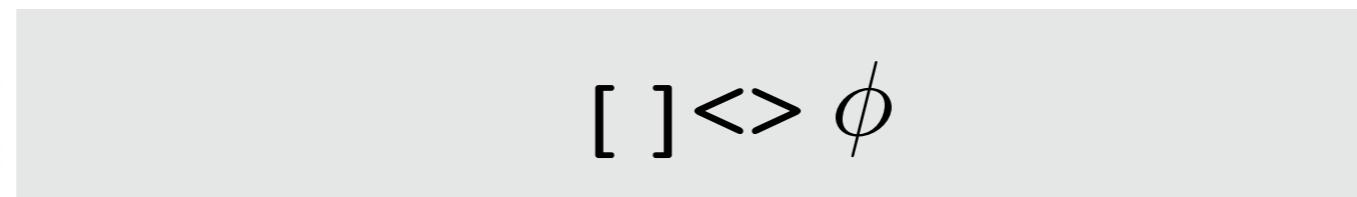


PATRONES DE PROPIEDADES LTL

Safety: “algo malo no va a ocurrir”



Liveness: “algo bueno ocurrirá siempre”



PATRONES DE PROPIEDADES LTL

Dwyer et. al “*Patterns in Property Specifications for Finite-State Verification*”

• **Ocurrencia**

- Ausencia
- Universal
- Existencia
- Existencia acotada

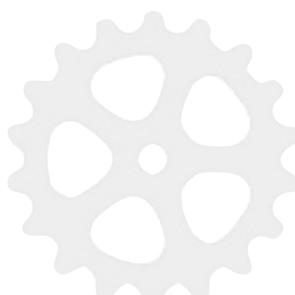
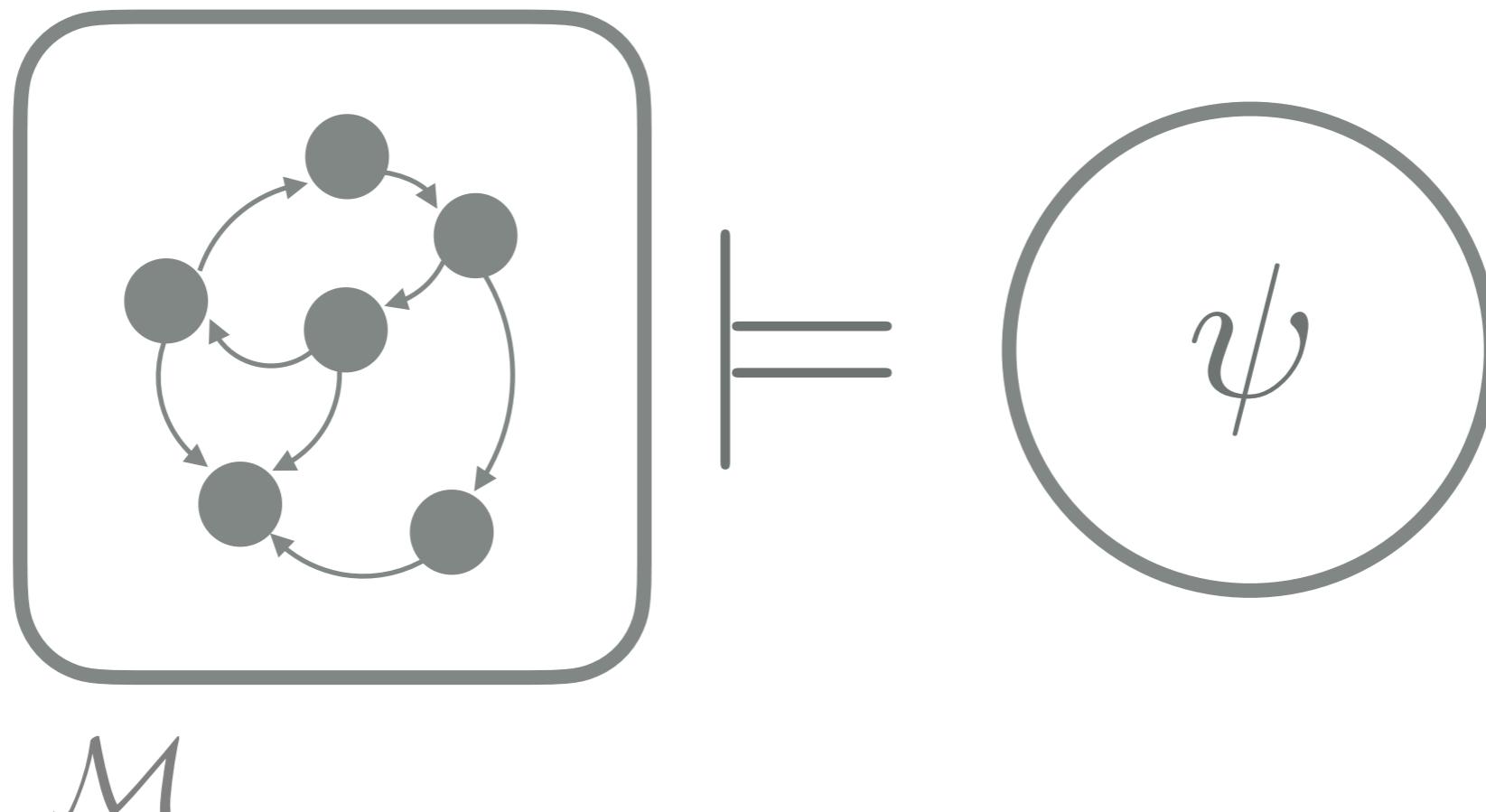
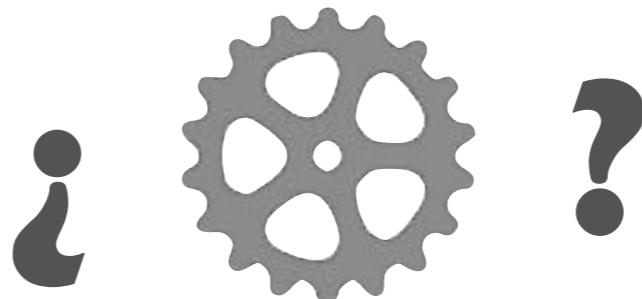
• **Orden**

- Precedencia
- Respuesta
- Cadena de Precedencia
- Cadena de Respuesta



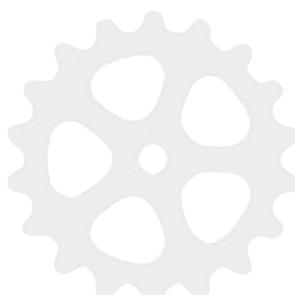
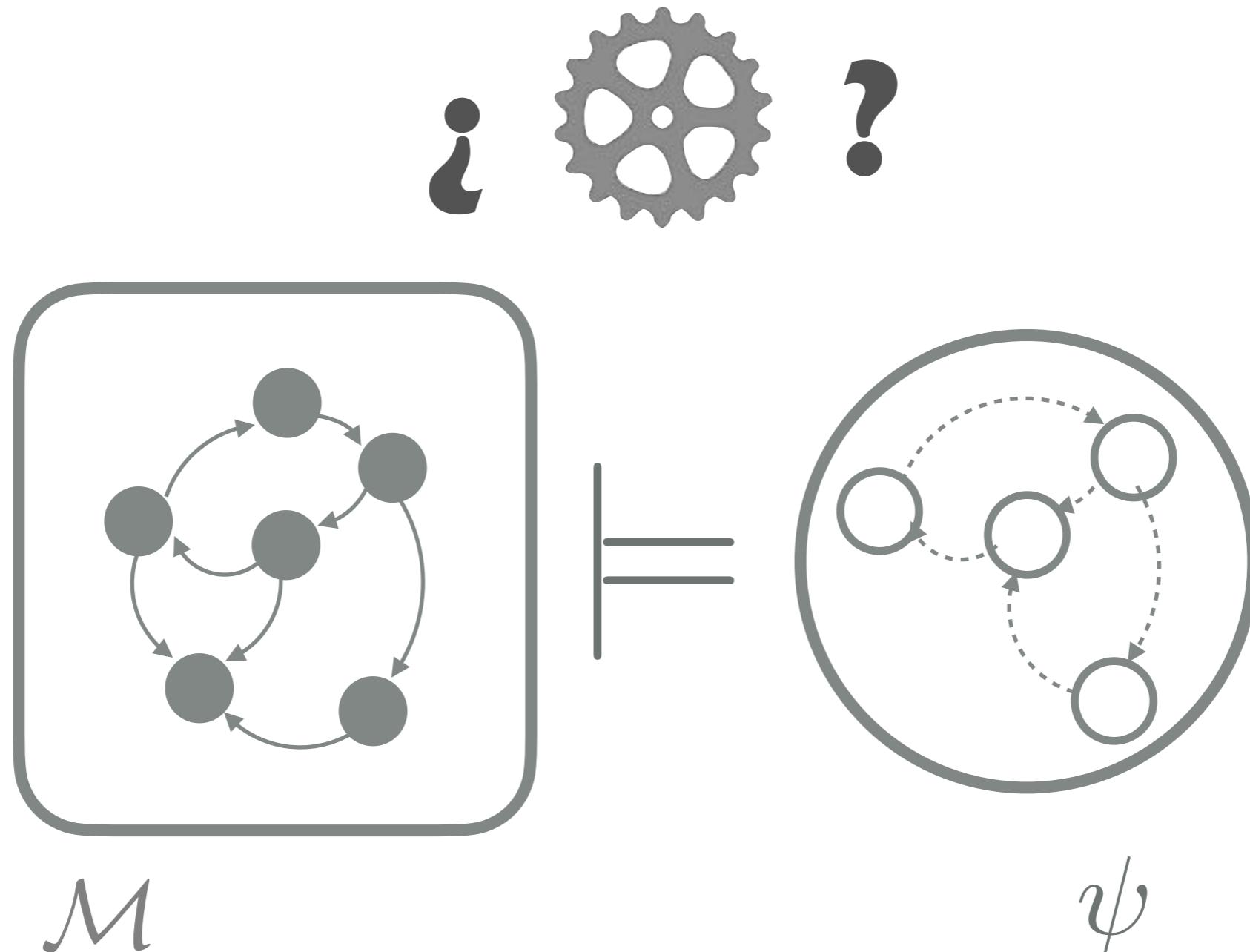
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING



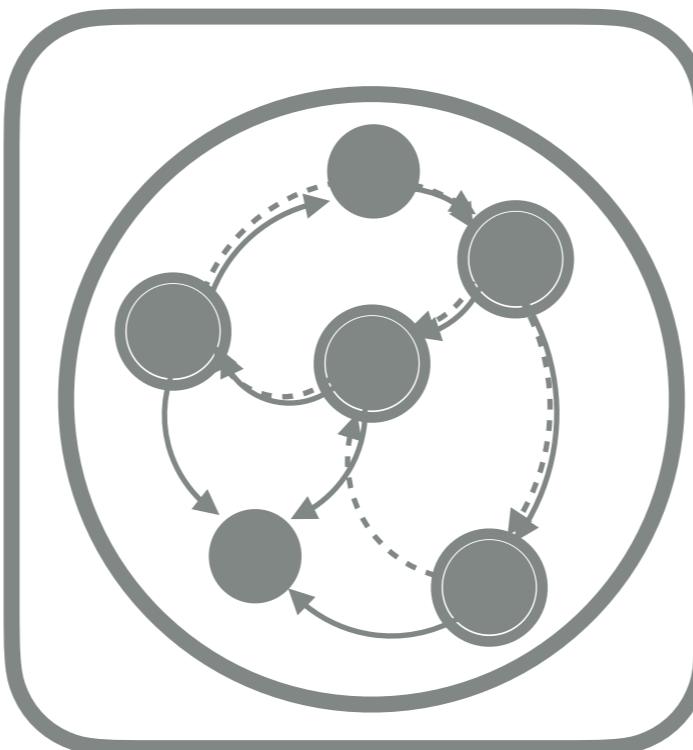
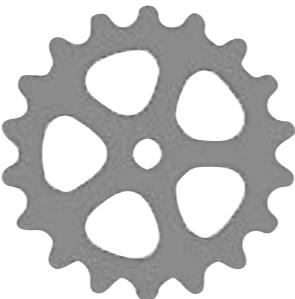
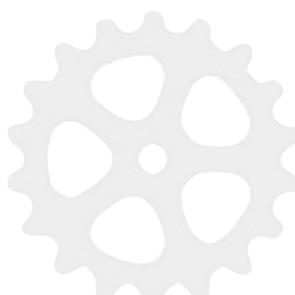
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING



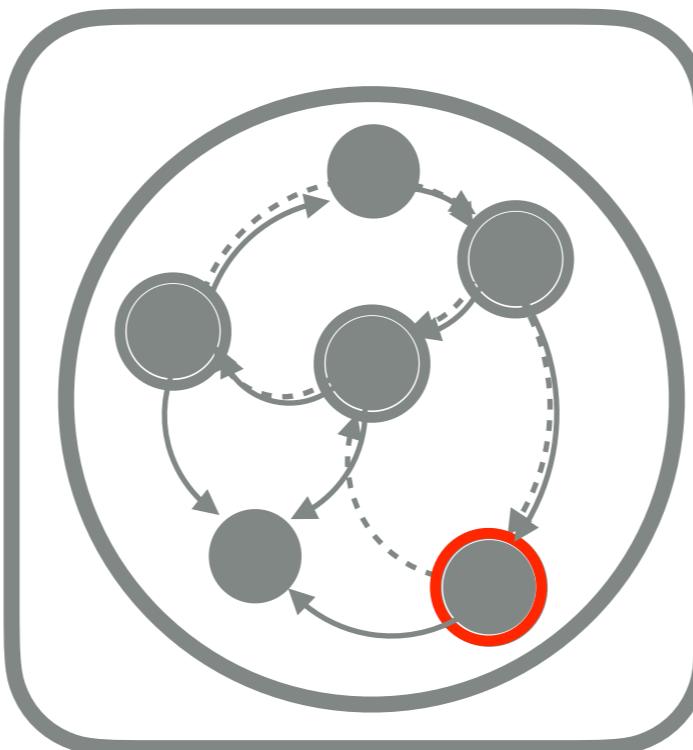
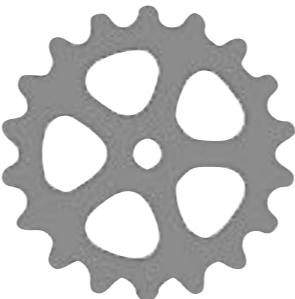
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

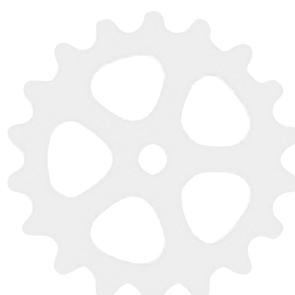
 \mathcal{M} ψ 

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

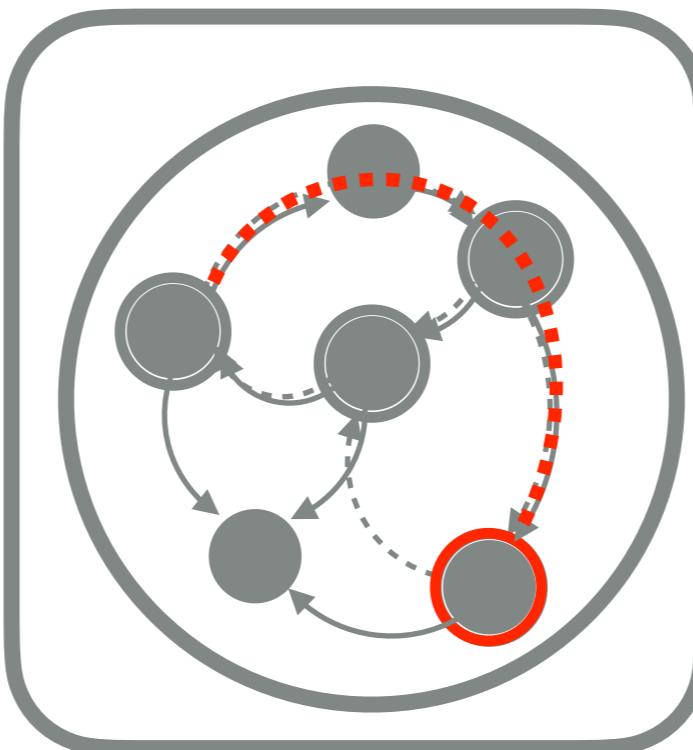
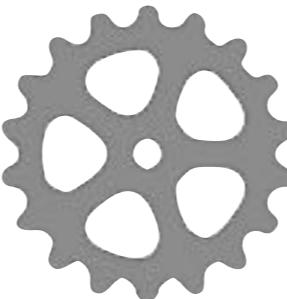


$$\mathcal{M} \quad \neg\psi$$

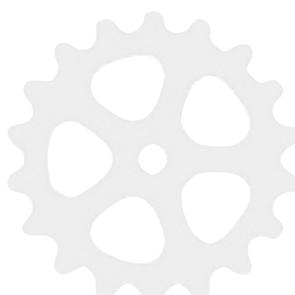


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

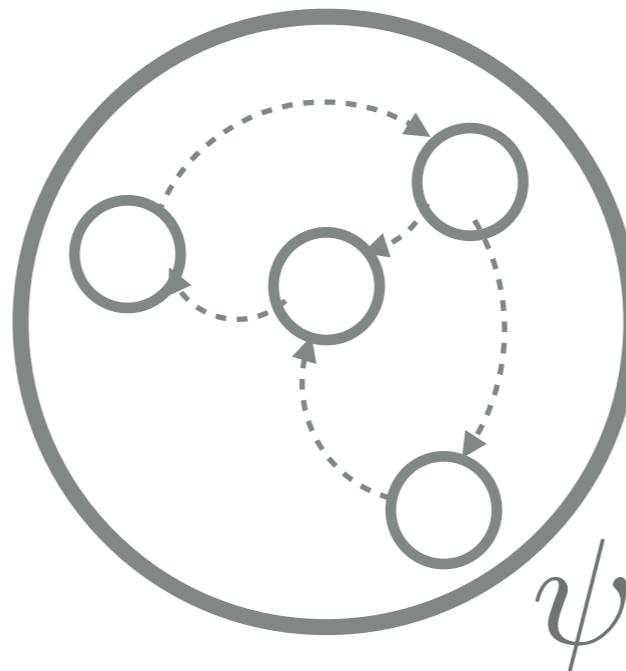


$$\mathcal{M} \models \neg \psi$$

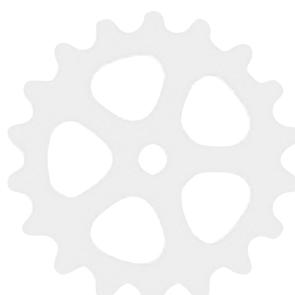


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

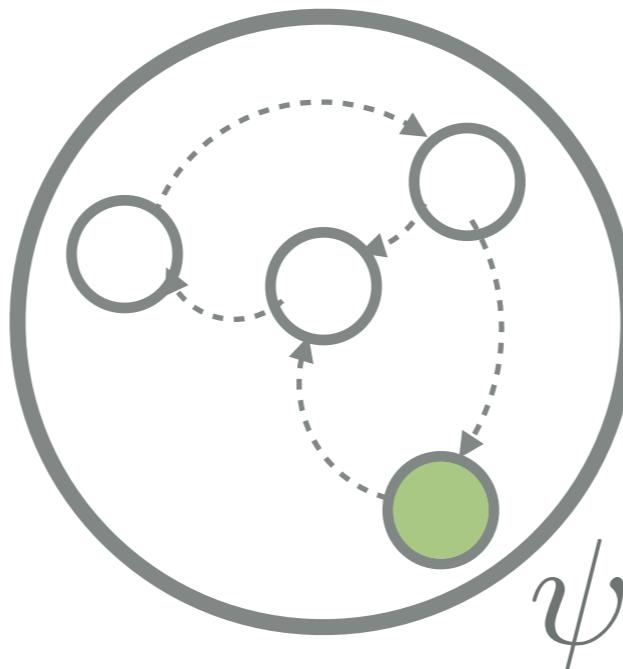


a b d a b d a b d d a a b d a d a d d

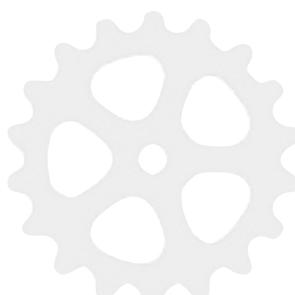


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

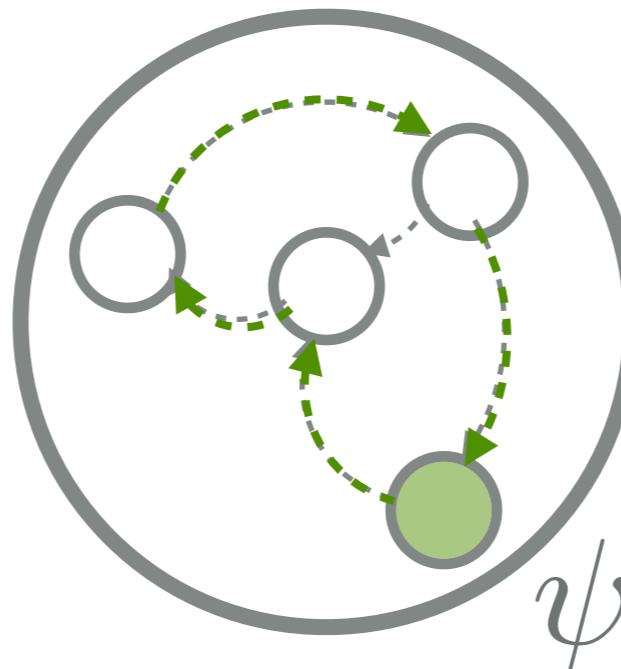


a b d a b d a b d d a a b d a d a d d

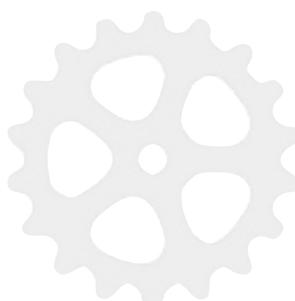


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

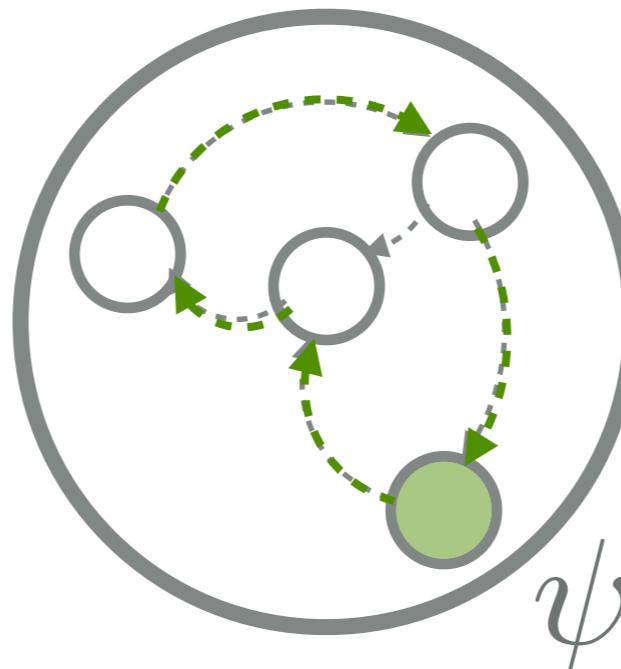


a b d a b d a b d d a a b d a d a d d

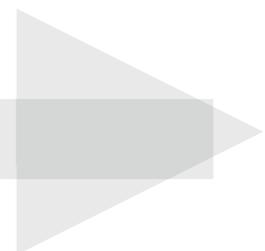


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

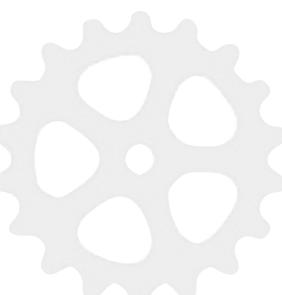


a b d a b d a b d d a a b d a d a d d

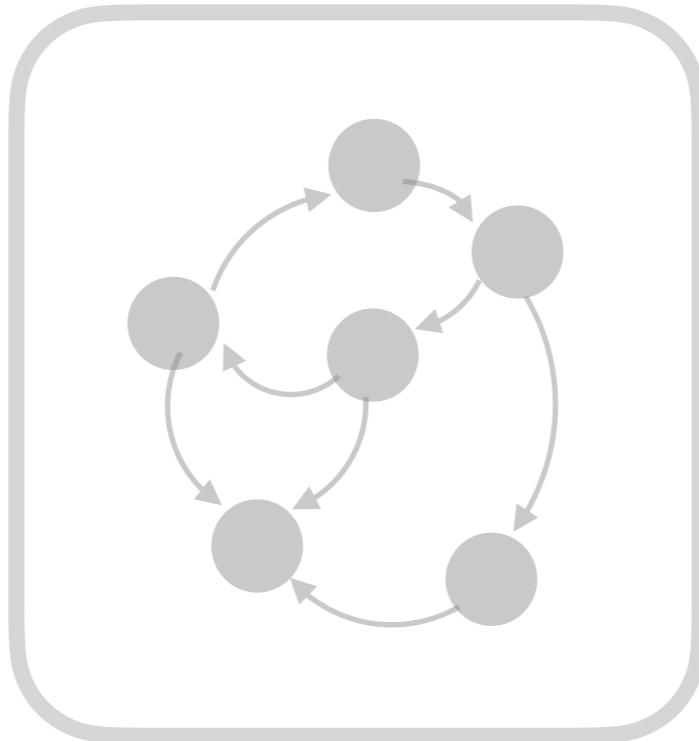


Autómata de Büchi:

“una cadena es aceptada si infinitamente pasa por estados de aceptación”



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **SAFETY**



a b d a b d a b d d a a b d a d b b b

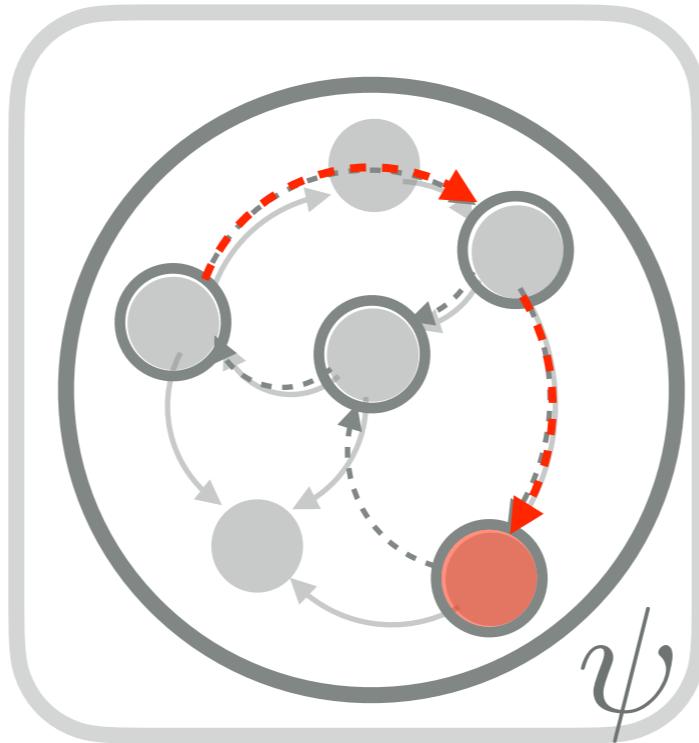
a a b d a b d a b d d a a b d a d a d

b d a b d a b d d a a b d a d a d d d

d a b d a b d a b d d a a b d a d a d d



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **SAFETY**



a b d a b d a b d d a a b d a d b b b

a a b d a b d a b d d a a b d a d a d

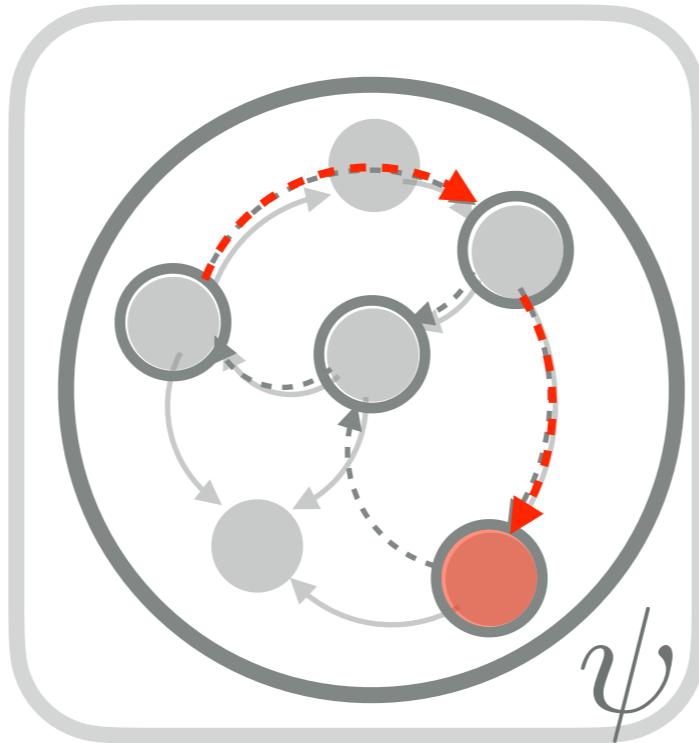
b d a b d a b d d a a b d a d a d d d

d a b d a b d a b d d a a b d a d a d d

 □ □ □ □ □ □ □ □ □ □ □



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **SAFETY**



a b d a b d a b d d a a b d a d b b b

a a b d a b d a b d d a a b d a d a d

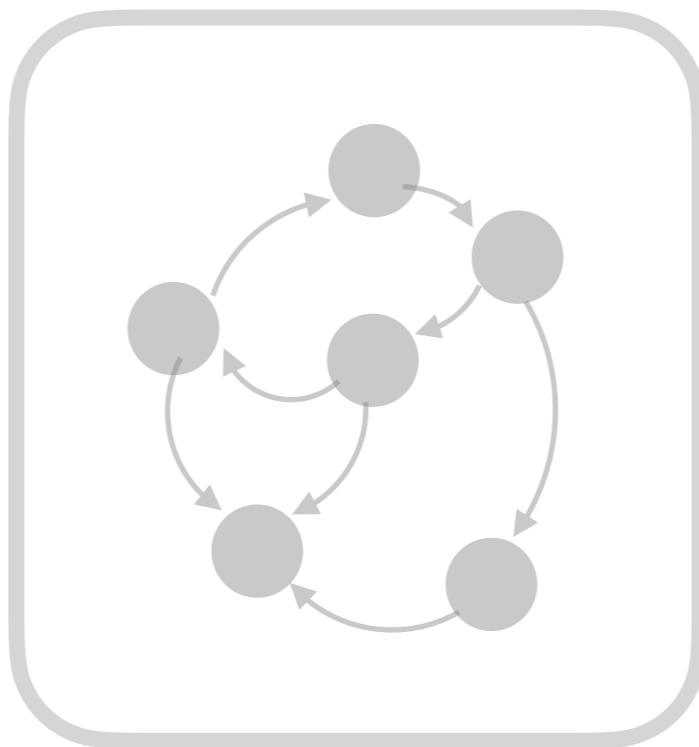
b d a b d a b d d a a b d a d a d d d

d a b d a b d a b d d a a b d a d a d d

 □ □ □ □ □ □ □ □ □ □ □



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **PROGRESO**



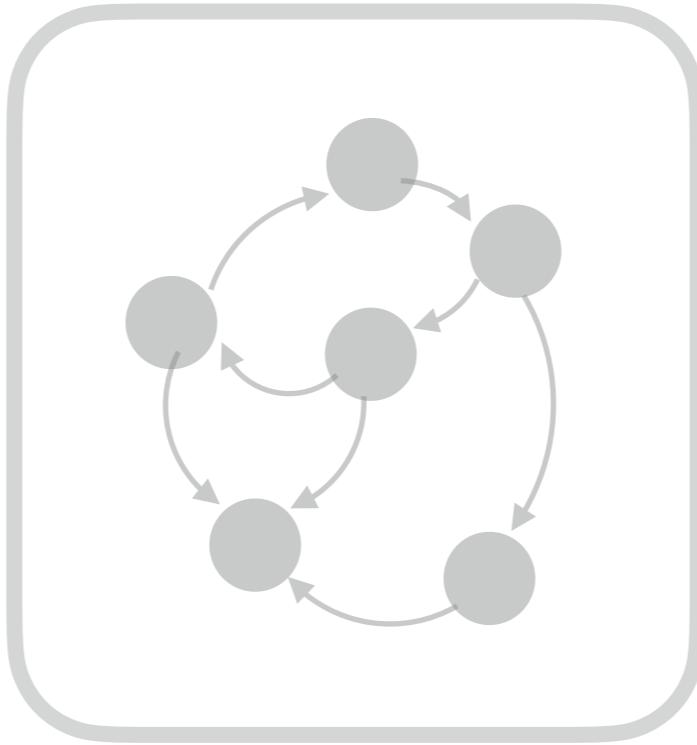
a b d a b d a b d d a a b d a d b

a a b d a b d a b d d a a b d a d

b d a b d a b d d a a b d a d a d

d a b d a b d a b d d a a b d a d

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES PROGRESO



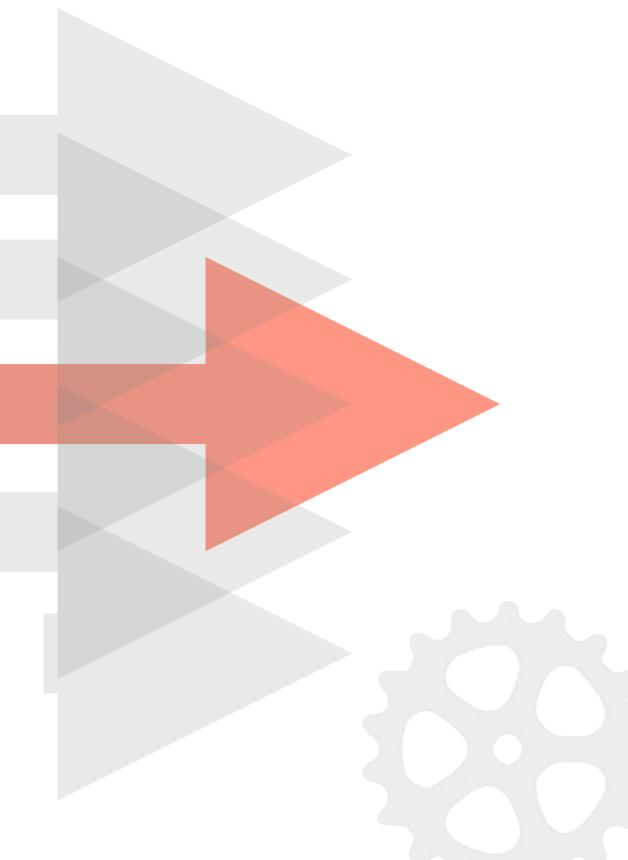
a b d a b d a b d d a a b d a d b

a a b d a b d a b d d a a b d a d

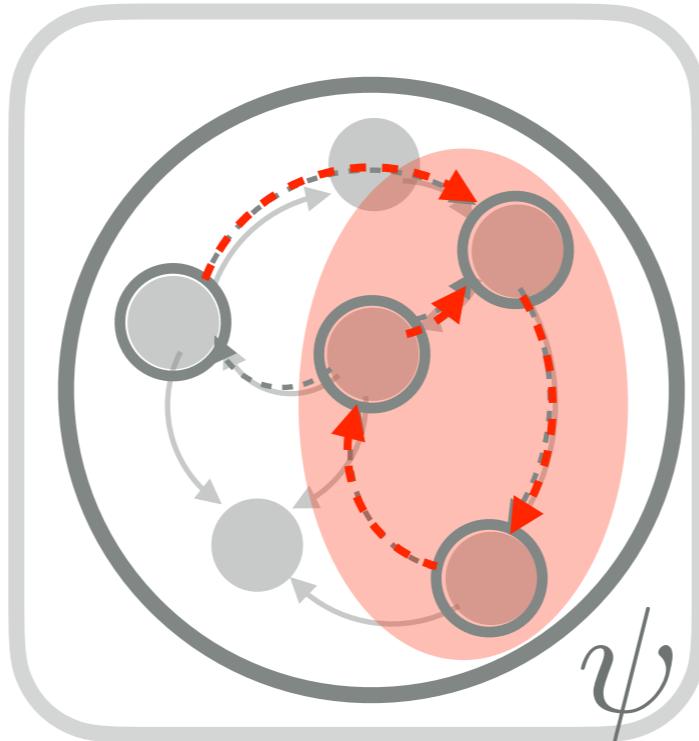
b d a b d a b d d a a b d a d a d

d a b d a b d a b d d a a b d a d

 □ □ □ □ □ □ □ □ □ □ □



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES PROGRESO

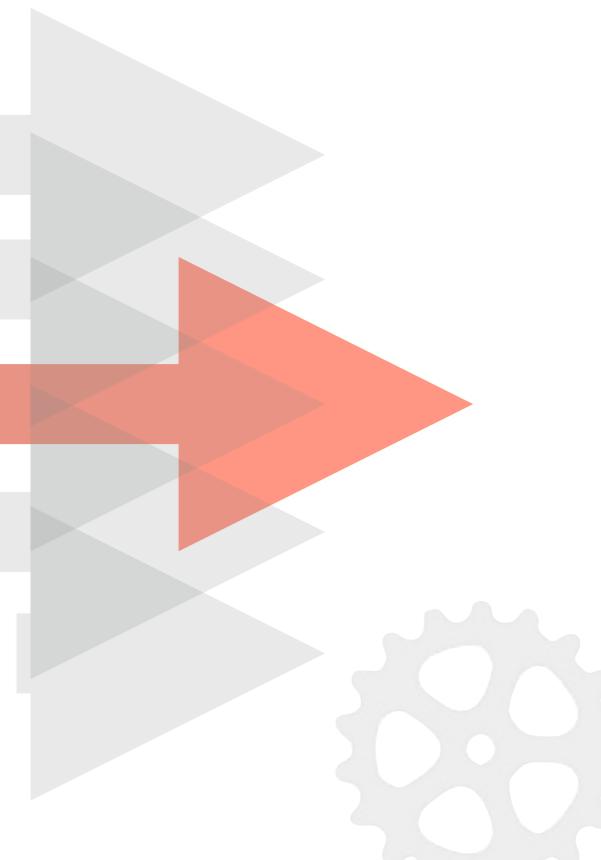


a b d a b d a b d d a a b d a d b

a a b d a b d a b d d a a b d a d

b d a b d a b d d a a b d a d a d

d a b d a b d a b d d a a b d a d



AYUDA A MEMORIA (FLUENTES)

a b d c b d b b d d a a b d a d b

Un fluente es una variable proposicional cuyo valor de verdad está asociado a la ocurrencia de eventos:

```
fluent x = <{c}, {a}> initially false
```

Conjunto de
Activación

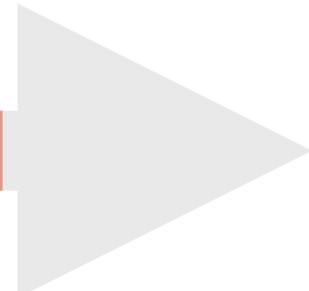
Conjunto de
Desactivación

ψ

[] (... x ...)

AYUDA A MEMORIA (FLUENTES)

a b d c b d b b d | d a a b d a d b



Un fluente es una variable proposicional cuyo valor de verdad está asociado a la ocurrencia de eventos:

```
fluent x = <{c}, {a}> initially false
```

Conjunto de
Activación

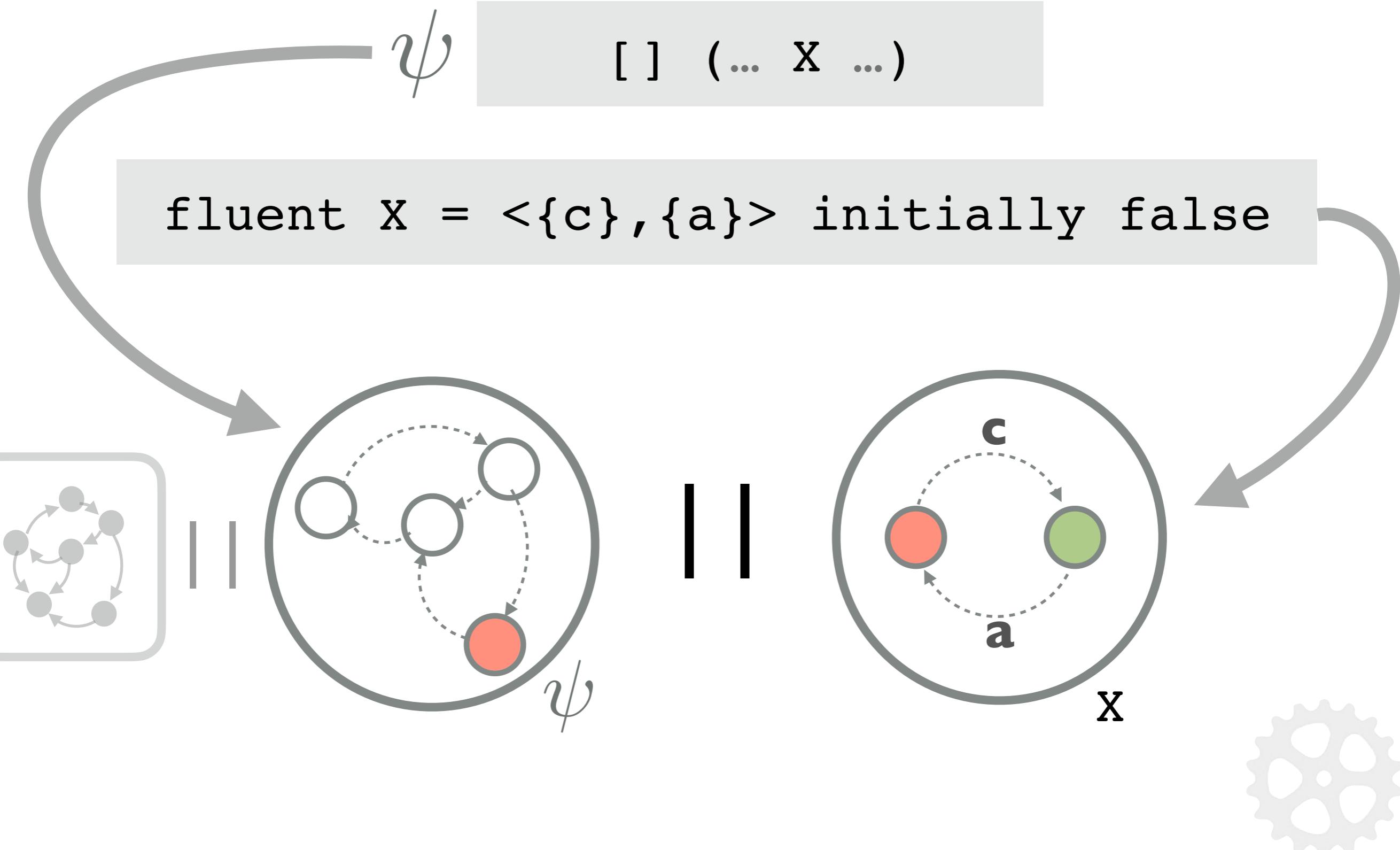
Conjunto de
Desactivación

ψ

[] (... X ...)



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES PROPOSICIONALES



CUANDO UN BIT DE MEMORIA NO ALCANZA

Cómo podemos capturar propiedades donde está presente la necesidad de **contar cantidad de eventos.**

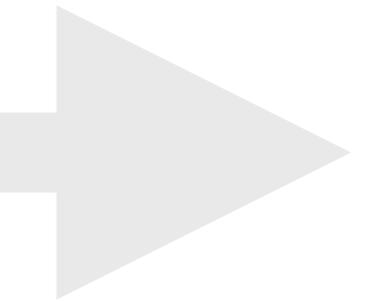
Algunos ejemplos de tomados de la literatura:

- Elevador
- Puente con capacidad
- Cajero automático
- Protocolo de TCP (con ventana dinámica)
- Lámpara temporizada
- Web Service (autenticación)
- Botón de pánico en sistema de hospitalización
- Sistema de pago electrónico

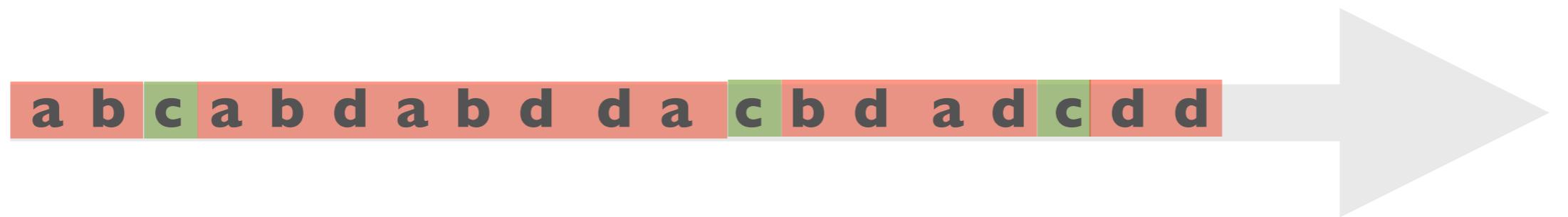


ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

a b c a b d a b d d a c b d a d c d d



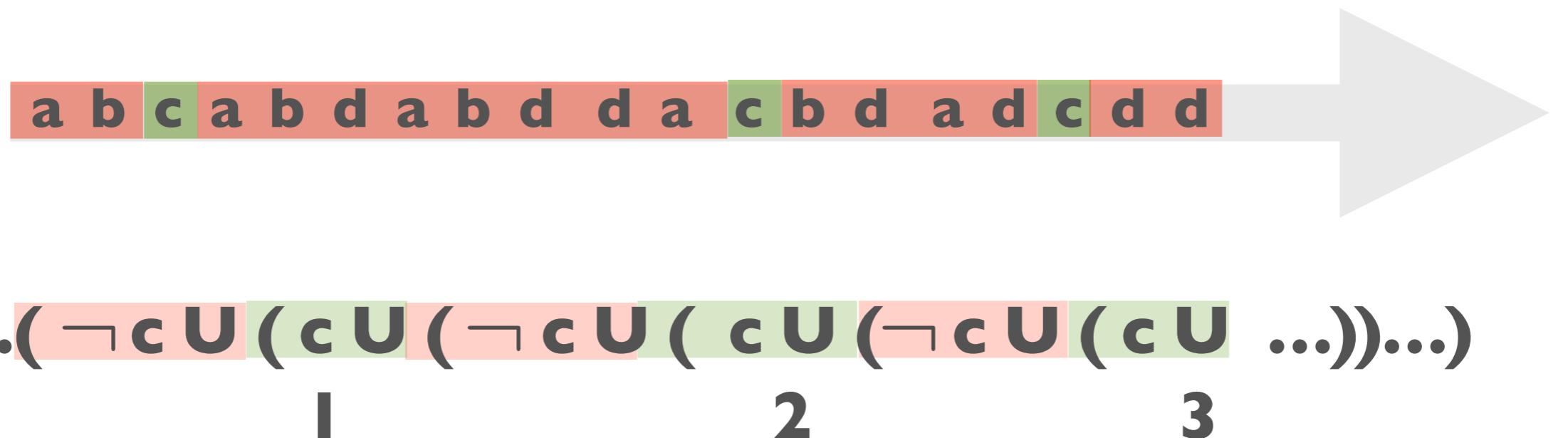
ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES



ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

The diagram illustrates the construction of a string and a corresponding infinite sequence of nested sets. The string is composed of alternating red and green segments. The first segment is red with letters a, b. The second segment is green with c. The third segment is red with a, b, d, a, b, d. The fourth segment is red with d, a. The fifth segment is green with c. The sixth segment is red with b, d, a, d. The seventh segment is green with c. The eighth segment is red with d, d. To the right of the string is a large grey arrow pointing to the right. Below the string is a sequence of nested sets represented by brackets. The first bracket is red with c. The second bracket is green with c. The third bracket is red with c. The fourth bracket is green with c. The fifth bracket is red with c. The sixth bracket is green with c. The sequence continues with ellipses (...).

ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

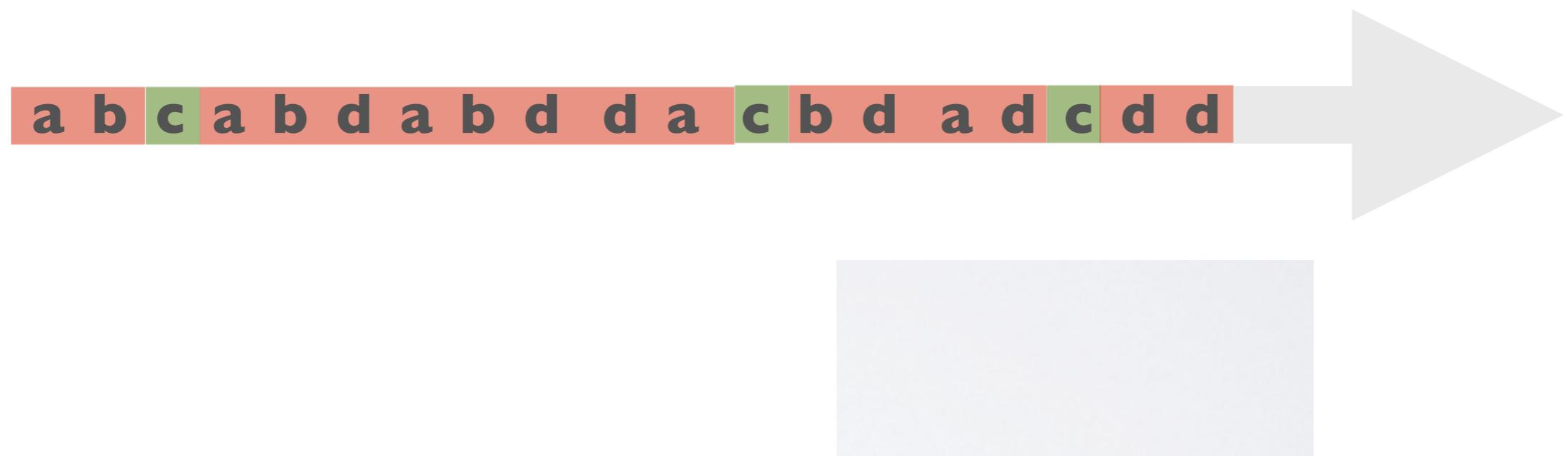


Con mínimas condiciones extras la fórmula se torna
inmanejable y propensa a errores.



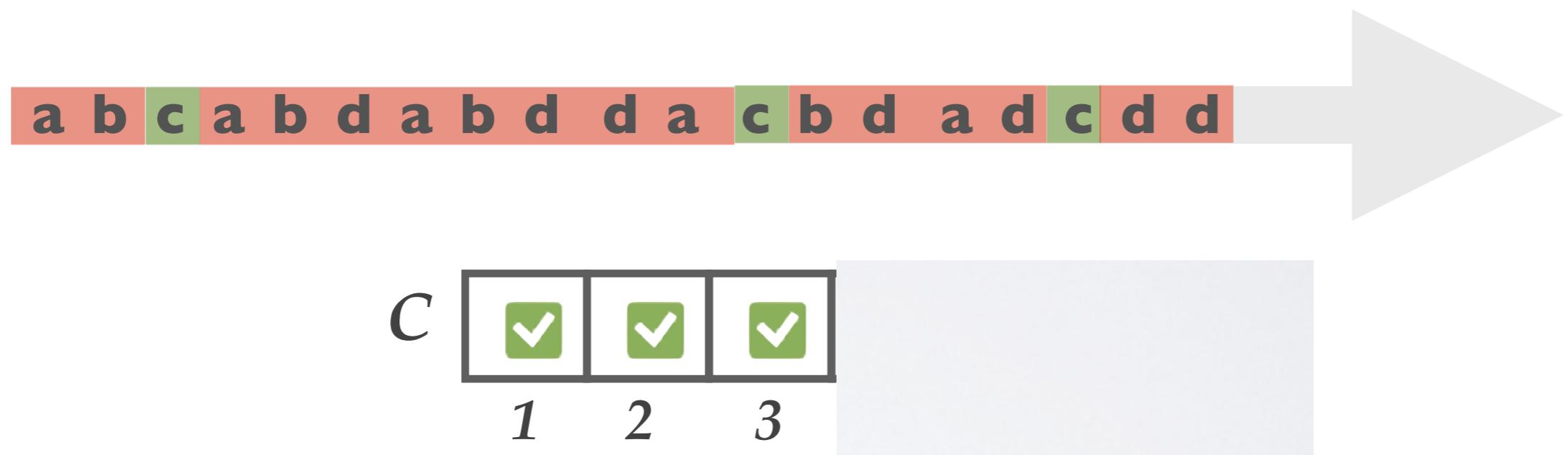
ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

Si podemos identificar cada elemento a contar podemos usar
arreglos de fluentes proposicionales



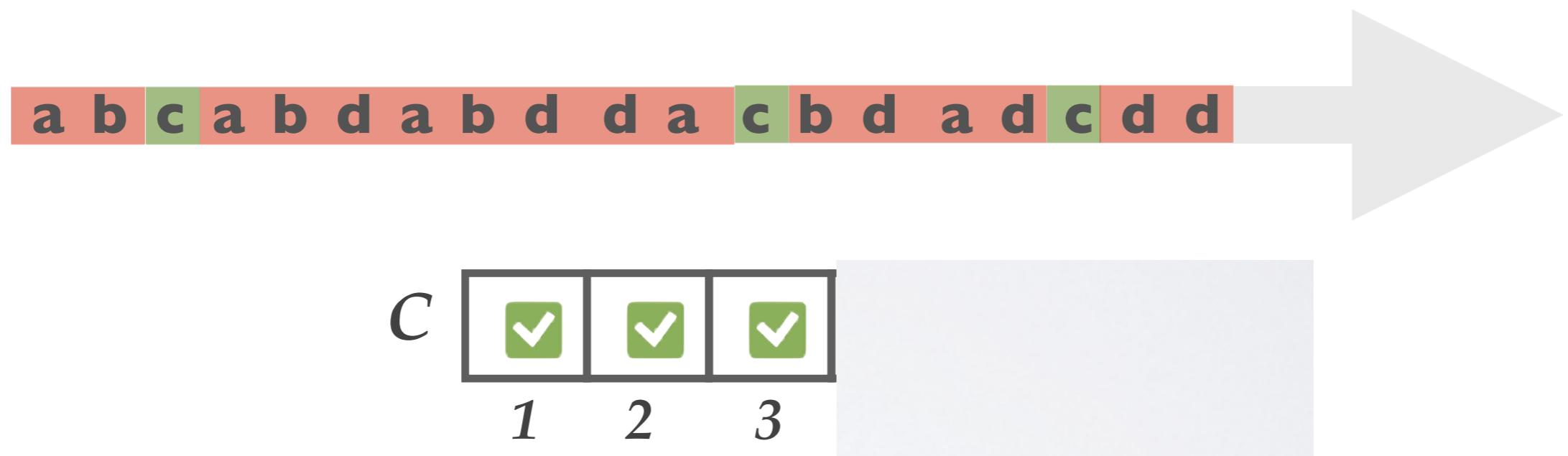
ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales



ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales

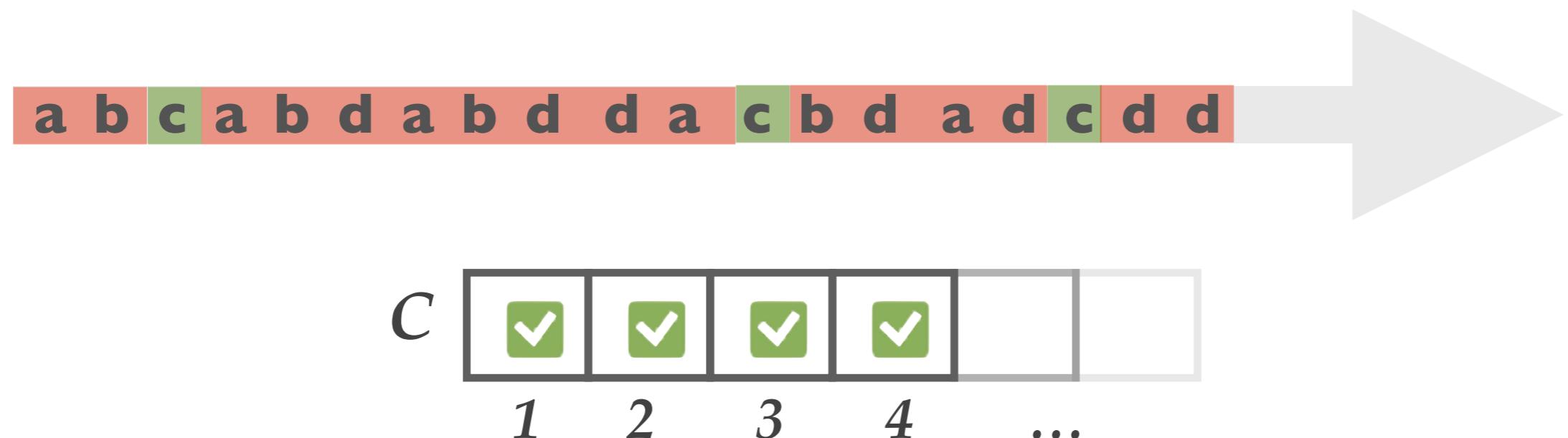


$\psi \dots (\mathbf{C}_1 \wedge \mathbf{C}_2 \wedge \mathbf{C}_3) \dots$



ALGUNAS FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales



$$\psi \dots (\mathbf{C}_1 \wedge \mathbf{C}_2 \wedge \mathbf{C}_3) \vee (\mathbf{C}_1 \wedge \mathbf{C}_2 \wedge \mathbf{C}_4) \vee (\mathbf{C}_1 \wedge \mathbf{C}_3 \wedge \mathbf{C}_4).$$

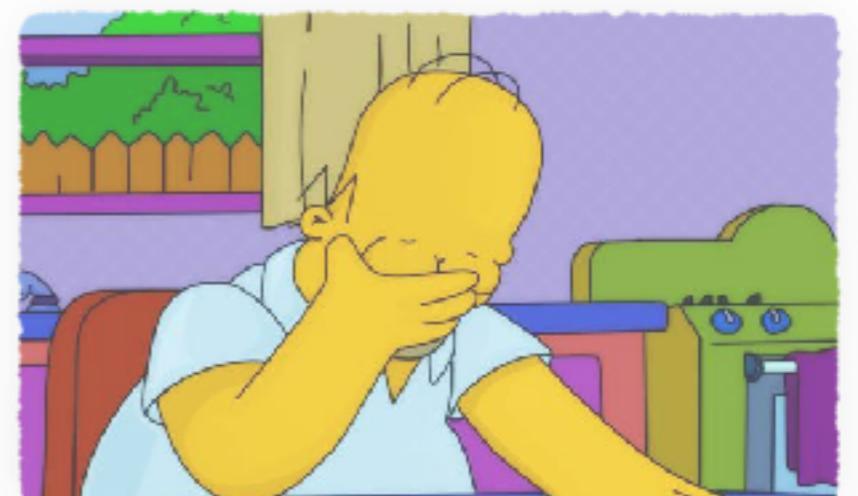


LTL CON FLUENTES PROPOSICIONALES NO ES SUFFICIENTE

Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales



i $\psi \dots (\# a \geq \# c) \dots ?$



LTL CON FLUENTES PROPOSICIONALES NO ES SUFFICIENTE

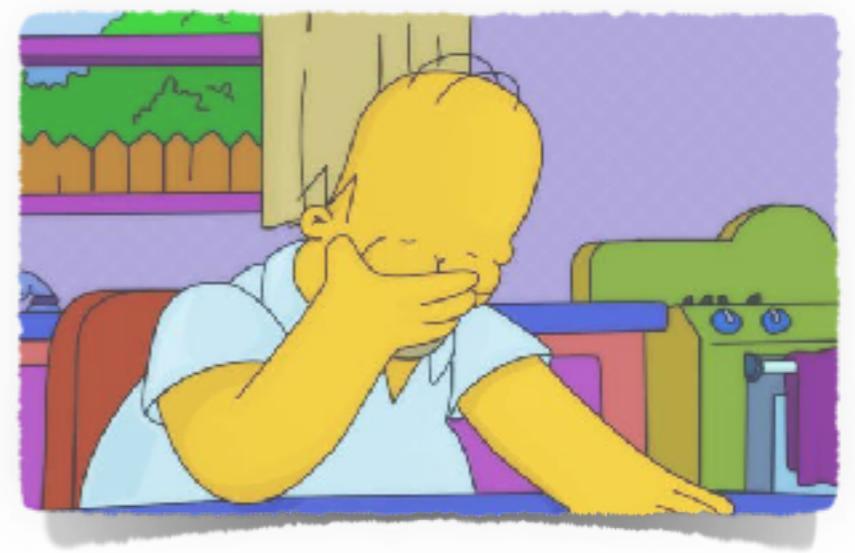
Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales



?

$\psi \dots (\# a \geq \# c) \dots$

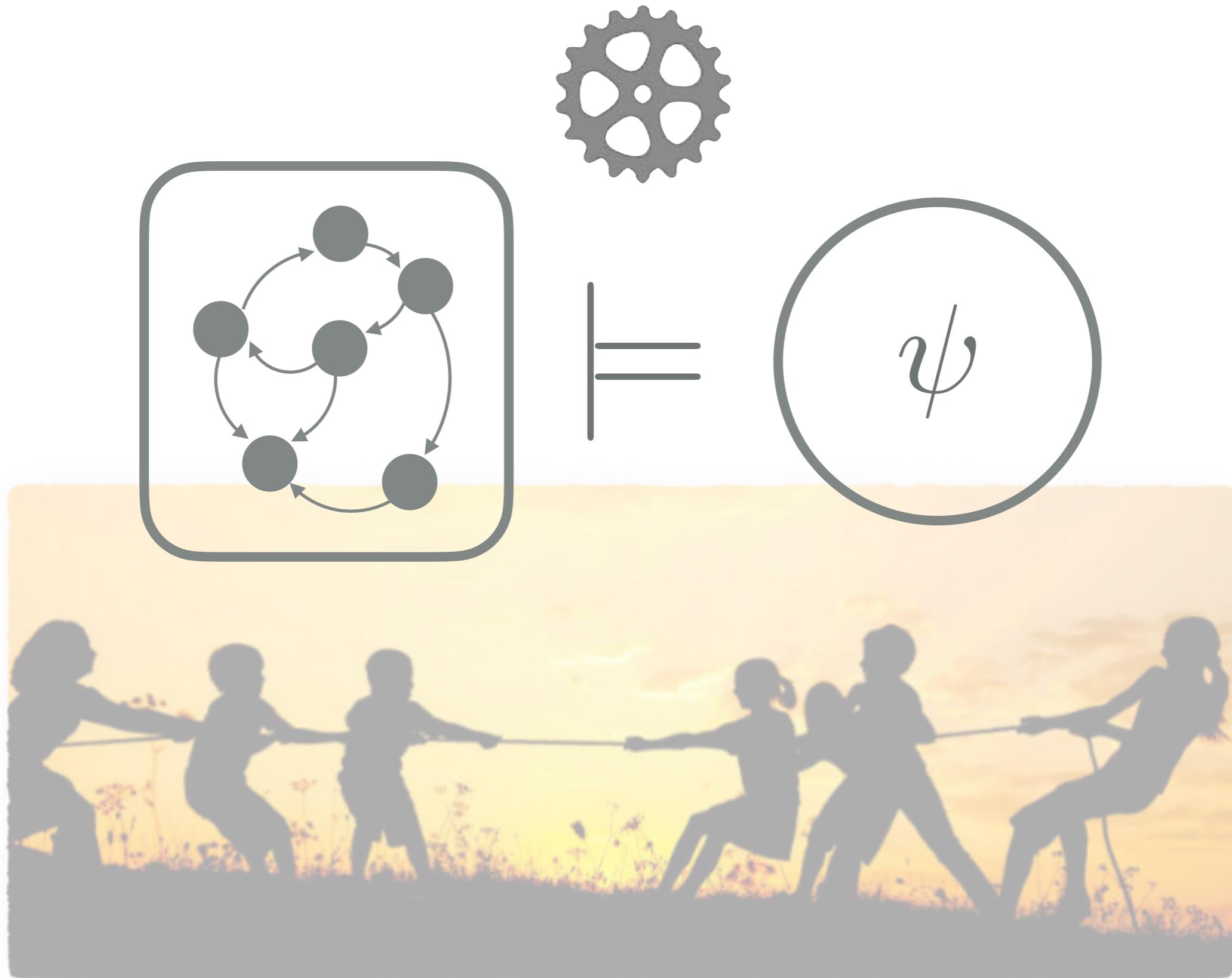
?



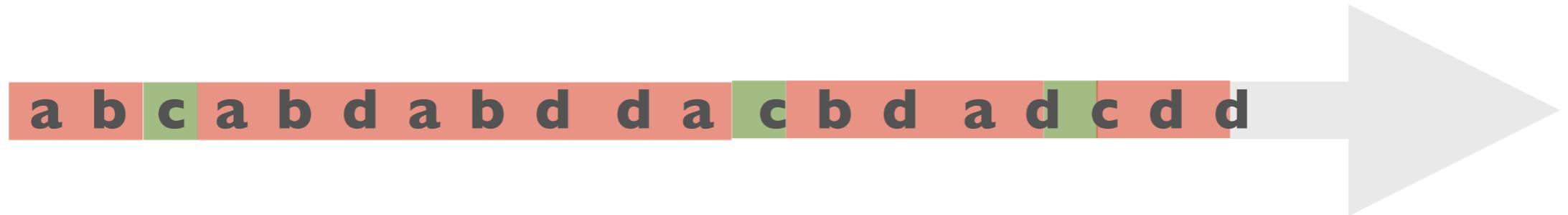
$\psi \dots (\text{!}a \wedge \text{!}c) \vee (\text{!}a \wedge \text{!}c) \vee (2a \wedge \text{!}c) \vee (2a \wedge \text{!}c) \vee (2a \dots) \dots$



EXPRESIVIDAD VS. EFICIENCIA (AUTOMATIZACIÓN)



FLUENTES CONTADORES



Un **fluente contador** es una variable entera cuyo valor está asociado a la ocurrencia de eventos:

```
cfluent k = <{c}, {}, {}> initially 0
```

Conjunto de
incremento

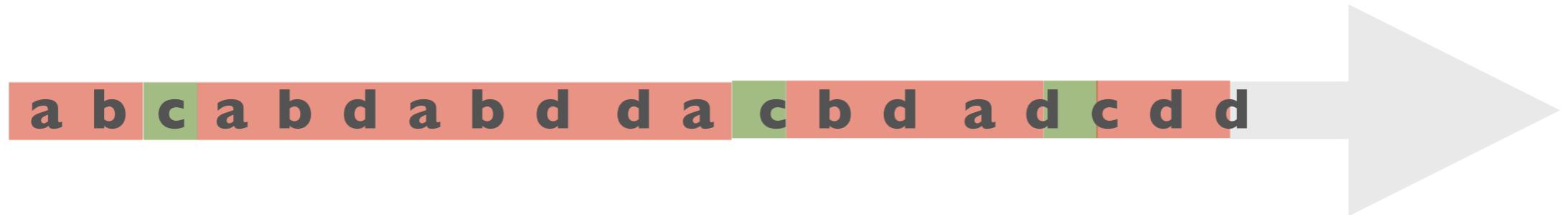
Conjunto de
decrement

Conjunto de
reinicio

valor
inicial



FLUENTES CONTADORES



Un **fluente contador** es una variable entera cuyo valor está asociado a la ocurrencia de eventos:

```
cfluent k = <{c}, {}, {}> initially 0
```

Conjunto de
incremento

Conjunto de
decrement

Conjunto de
reinicio

valor
inicial

ψ

[] (k = 3)

**Expresiones
contadoras**



FLUENTES CONTADORES LÍMITES

Dado el potencial infinito en los rangos de fluentes contadores (espacio infinito), para poder aplicar la técnica de Model Checking, los **limitamos**

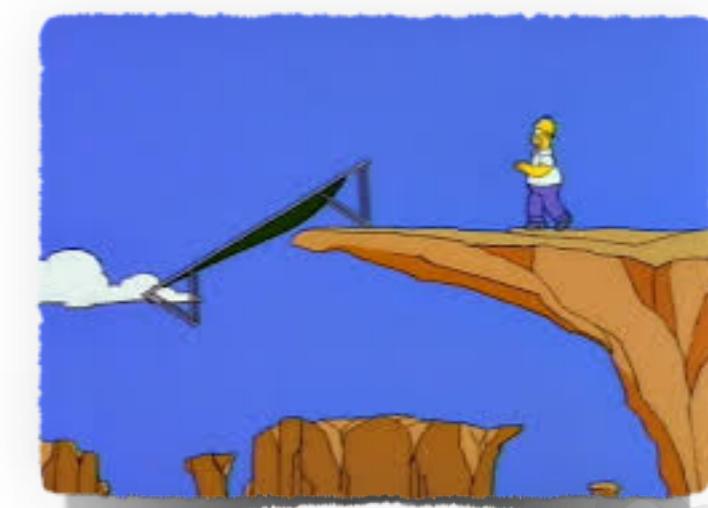
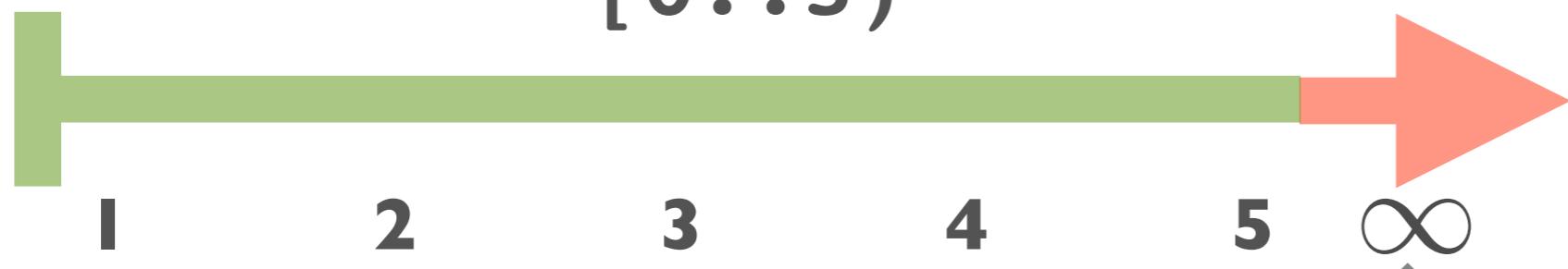


Límite **estricto** $[0..5]$



Límite **NO estricto**

$[0..5)$



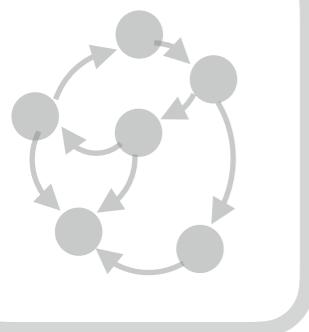
Si en algún momento del proceso de verificación alcanzamos **overflow** el resultado es

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES

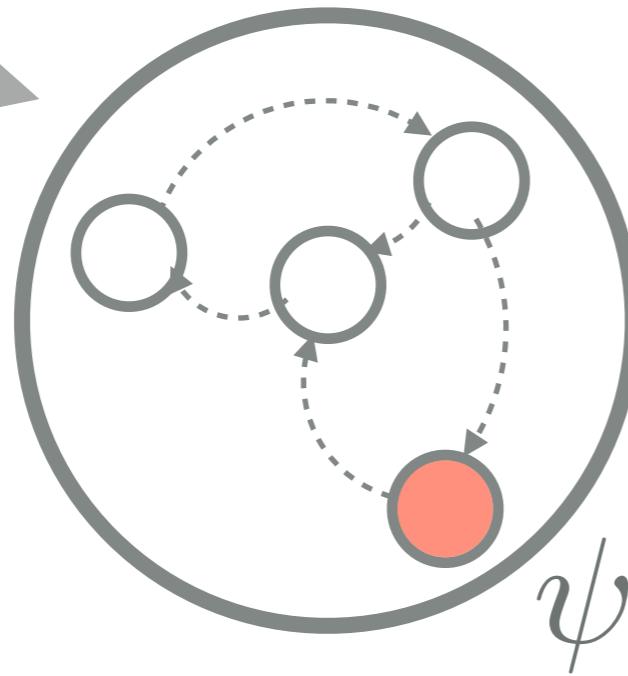
ψ

[] (... ($K \geq 5$) ...)

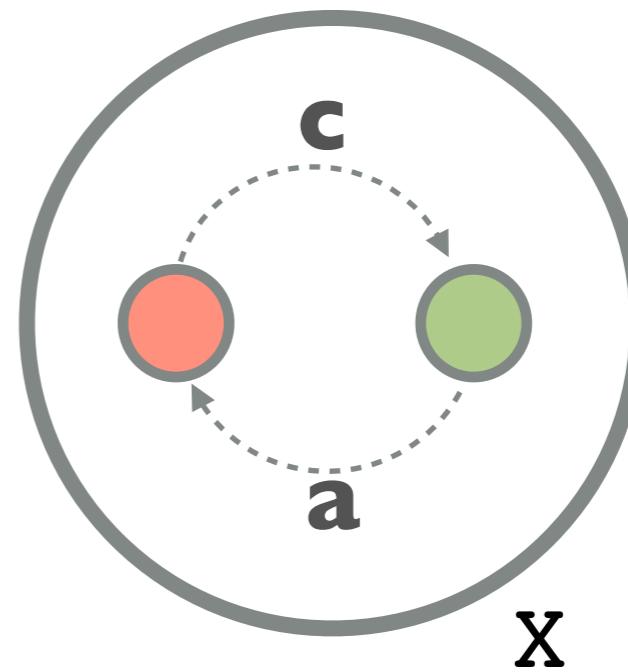
fluent $K = \langle \{c\}, \{a\}, \{d\} \rangle$ initially 0



II



III

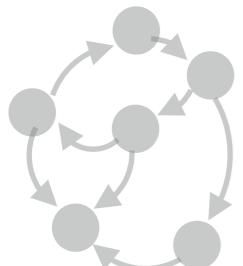


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES

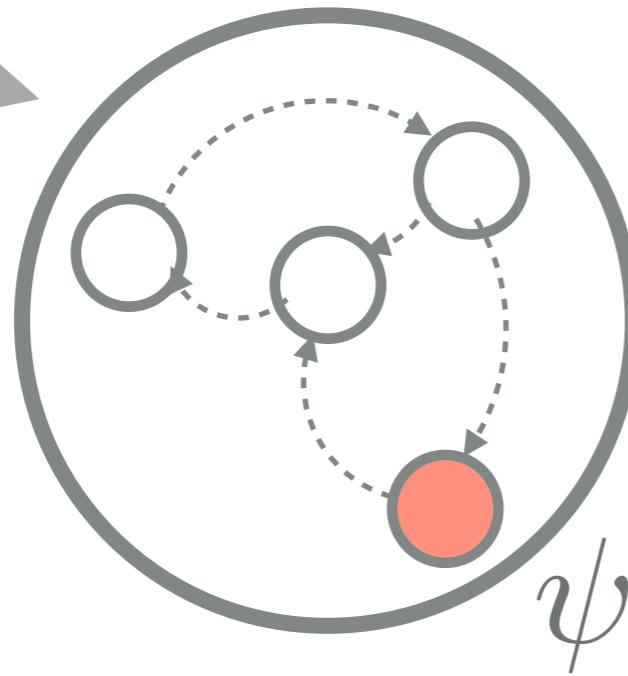
ψ

[] (... ($K \geq 5$) ...)

fluent $K = \langle \{c\}, \{a\}, \{d\} \rangle$ initially 0



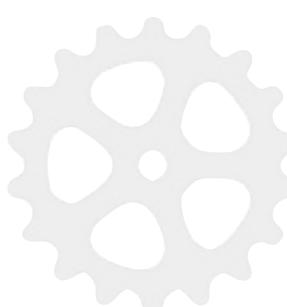
||



ψ

||

?

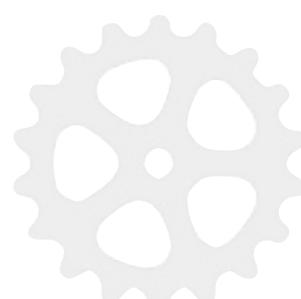
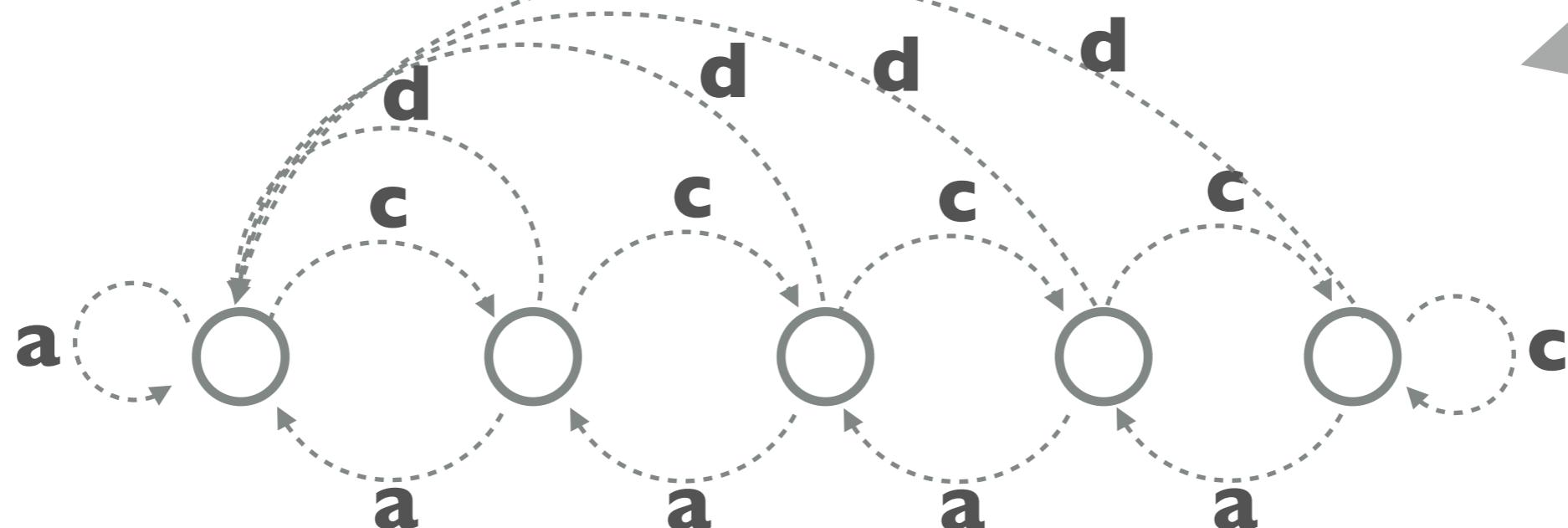


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON FLUENTES CONTADORES** (*LÍMITES ESTRICIOS*)

ψ

[] (... (K >= 3) ...)

```
fluent K = <{c}, {a}, {d}> initially 0 apply [0..4]
```

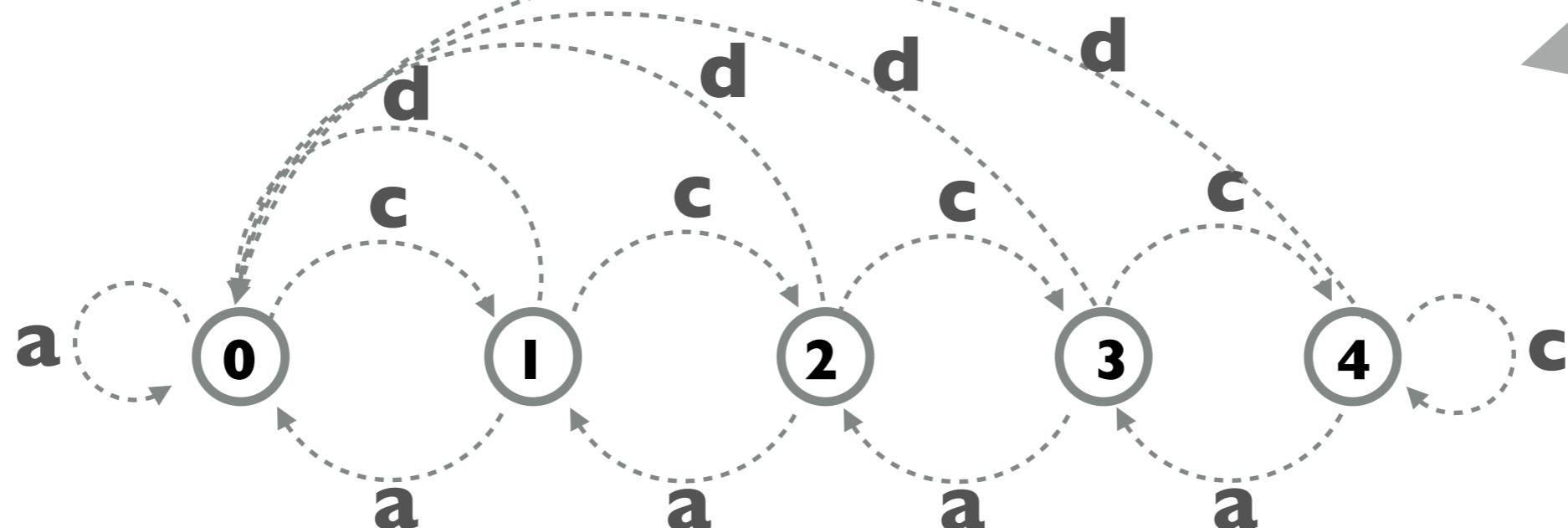


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES (*LÍMITES ESTRICIOS*)

ψ

[] (... (K >= 3) ...)

fluent K = <{c}, {a}, {d}> initially 0 apply [0..4]

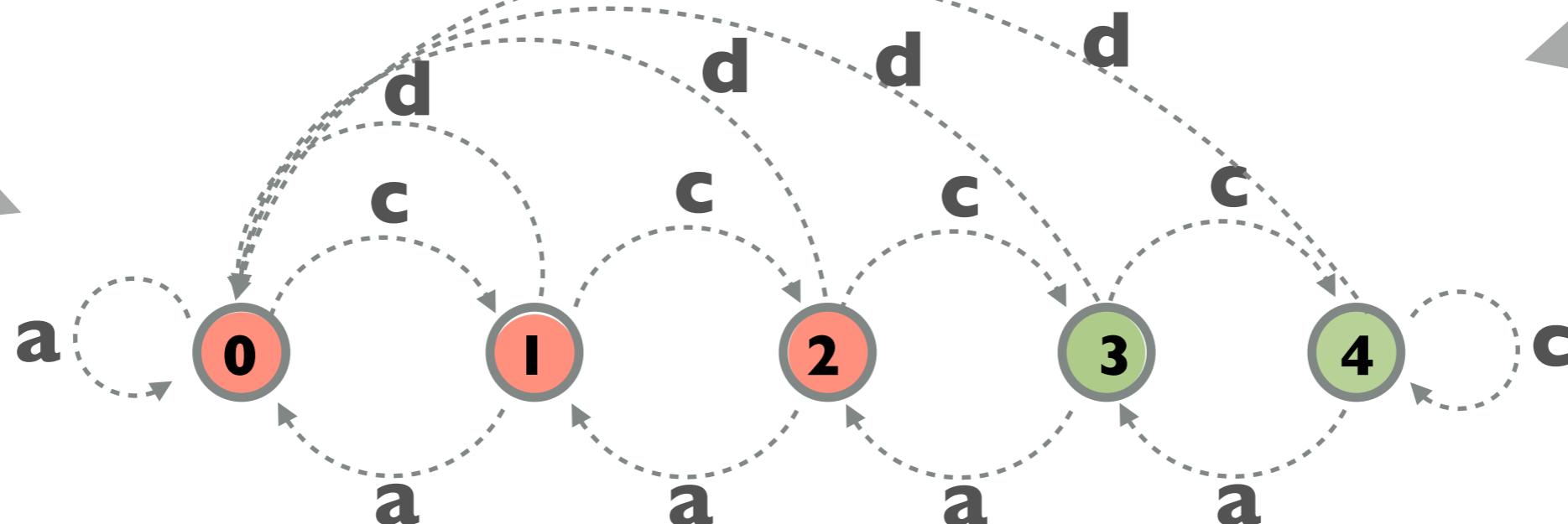


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES (LÍMITES ESTRICIOS)

ψ

[] (... ($K \geq 3$) ...)

fluent $K = \langle \{c\}, \{a\}, \{d\} \rangle$ initially 0 apply [0..4]

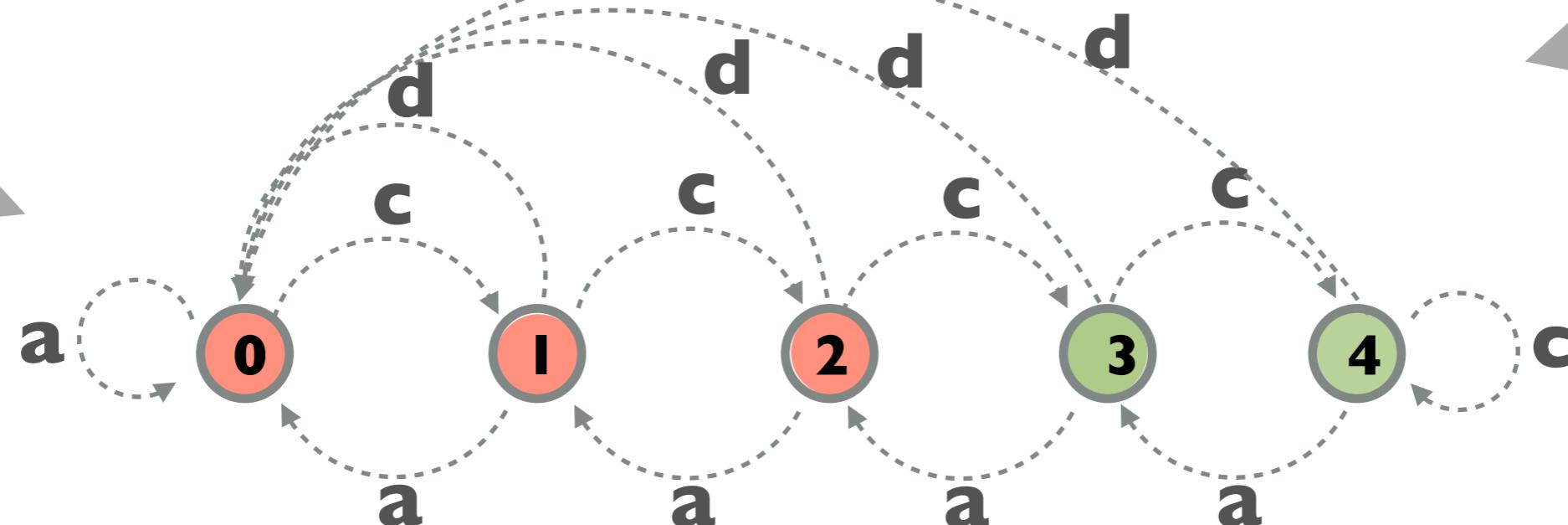


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES (LÍMITES **NO** ESTRICIOS)

ψ

[] (... ($K \geq 3$) ...)

fluent $K = <\{c\}, \{a\}, \{d\}>$ initially 0 apply [0..4)

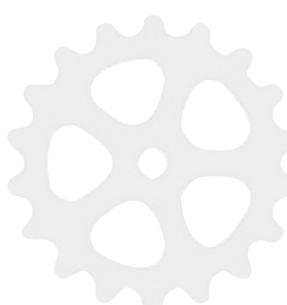
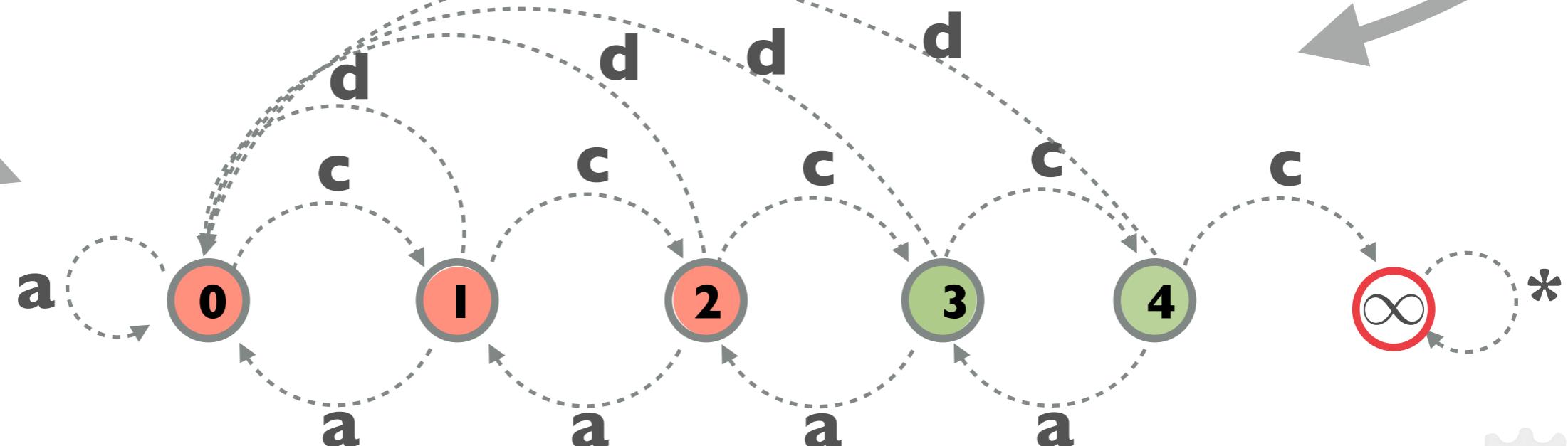


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON FLUENTES CONTADORES (LÍMITES **NO** ESTRICIOS)

ψ

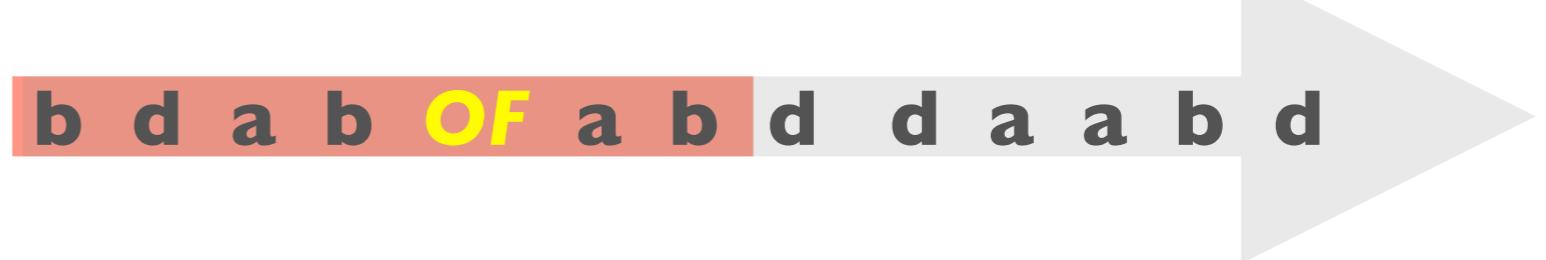
[] (... ($K \geq 3$) ...)

fluent $K = <\{c\}, \{a\}, \{d\}>$ initially 0 apply [0..4)



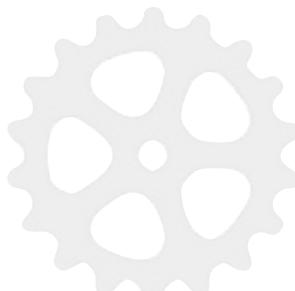
CÓMO TRATAR LOS **FLUENT OVERFLOW** (VERIFICACIÓN INCONCLUSA)

Safety:



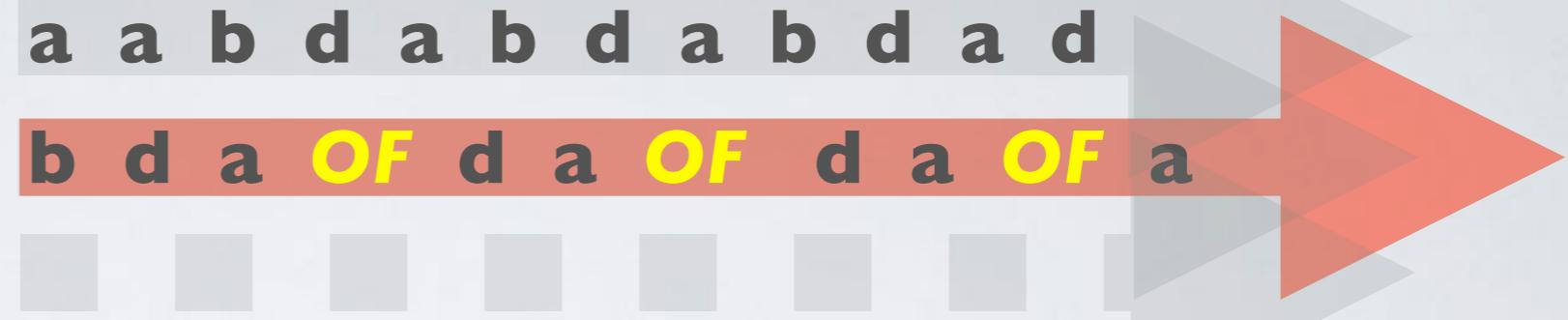
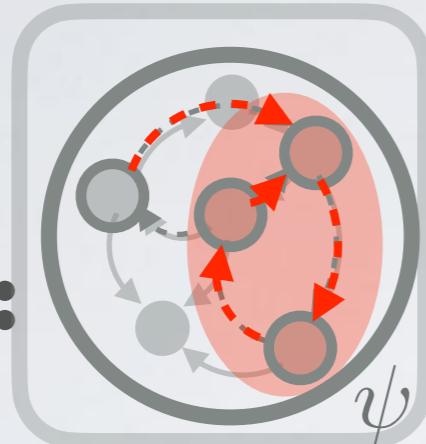
Si la traza encontrada presenta un evento de **overflow** no se tiene en cuenta.

- Si hay trazas **sin overflow** que alcancen error **(NO VALE)** contra-ejemplo real
- Si **no hay** trazas que alcancen error **(VALE)** no hay contra-ejemplos
- Si **hay y todas** las trazas que error alcanzan **tienen overflow (NO SÉ)** contra-ejemplos espurios



CÓMO TRATAR LOS **FLUENT OVERFLOW** (VERIFICACIÓN INCONCLUSA)

Progress:



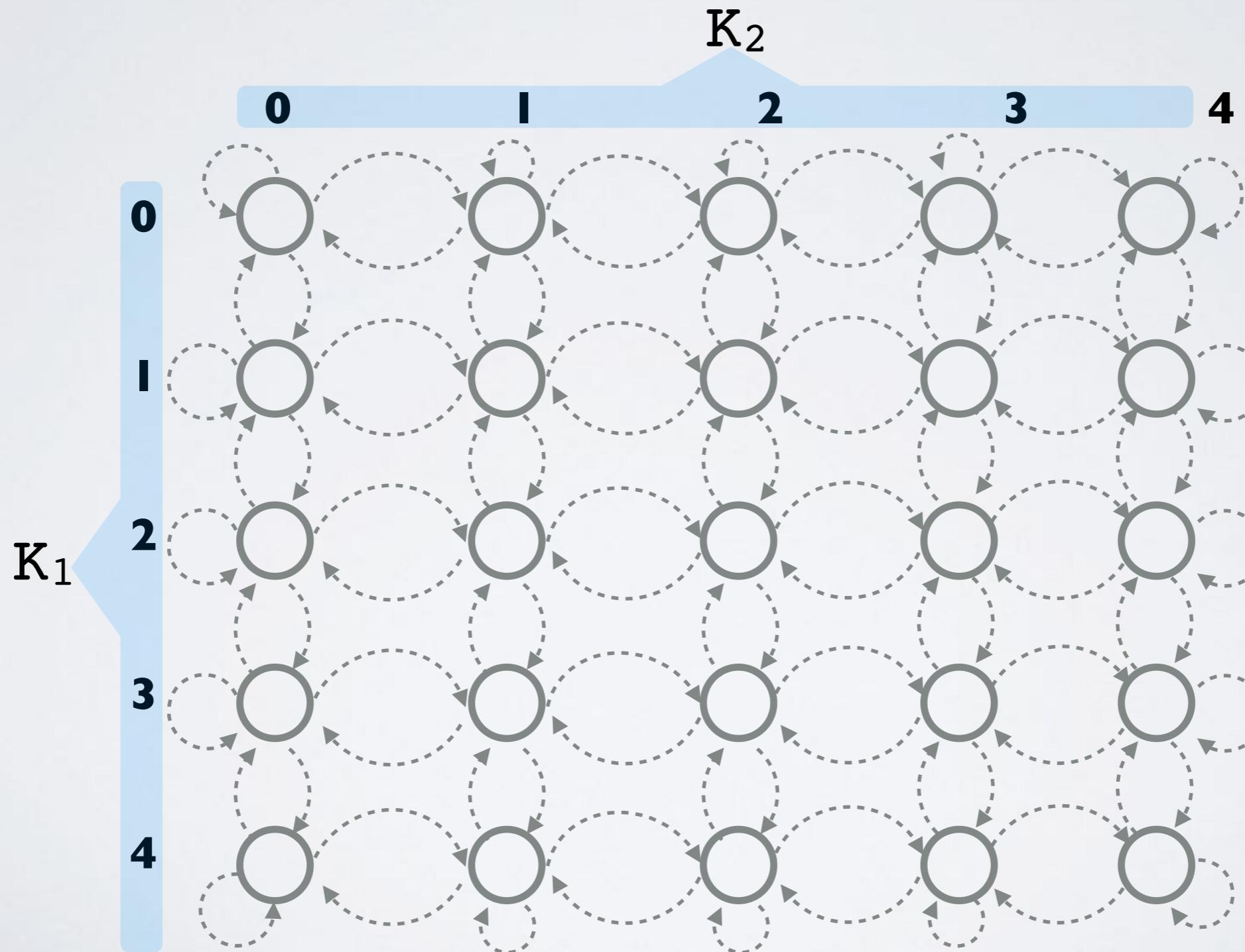
Si el componente conexo presenta un evento de **overflow** no se tiene en cuenta.

- Si hay componentes conexos **sin overflow (NO VALE)** contra-ejemplo real
- Si **no hay** componentes conexos **(VALE)** no hay contra-ejemplos
- Si **hay y todos** los comp. conexos **tienen overflow (NO SÉ)** contra-ejemplos espurios

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON EXPRESIONES CONTADORAS COMPLEJAS

ψ

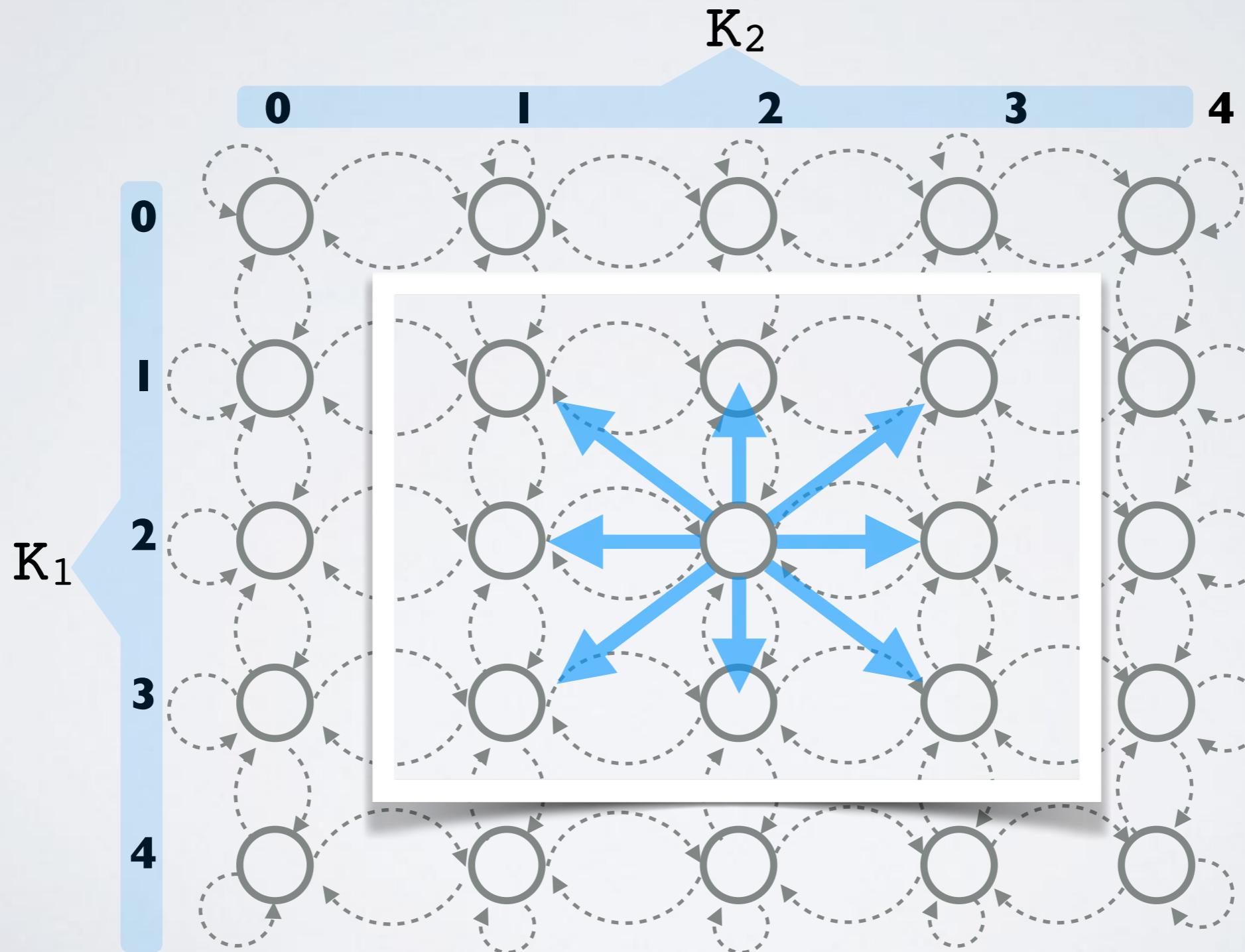
[] (... ($K_1 \geq K_2$) ...)



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON EXPRESIONES CONTADORAS COMPLEJAS

ψ

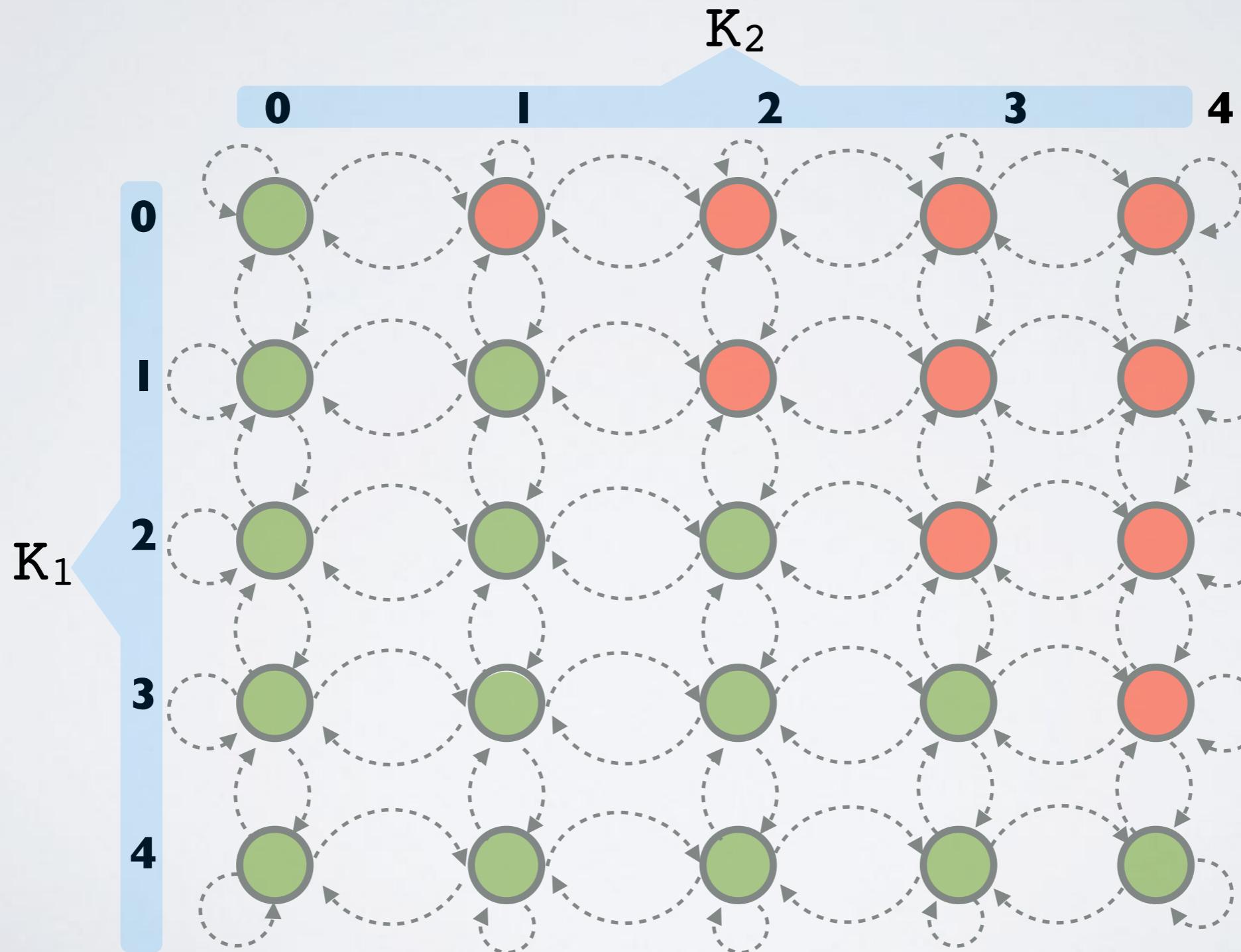
[] (... ($K_1 \geq K_2$) ...)



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON EXPRESIONES CONTADORAS COMPLEJAS

ψ

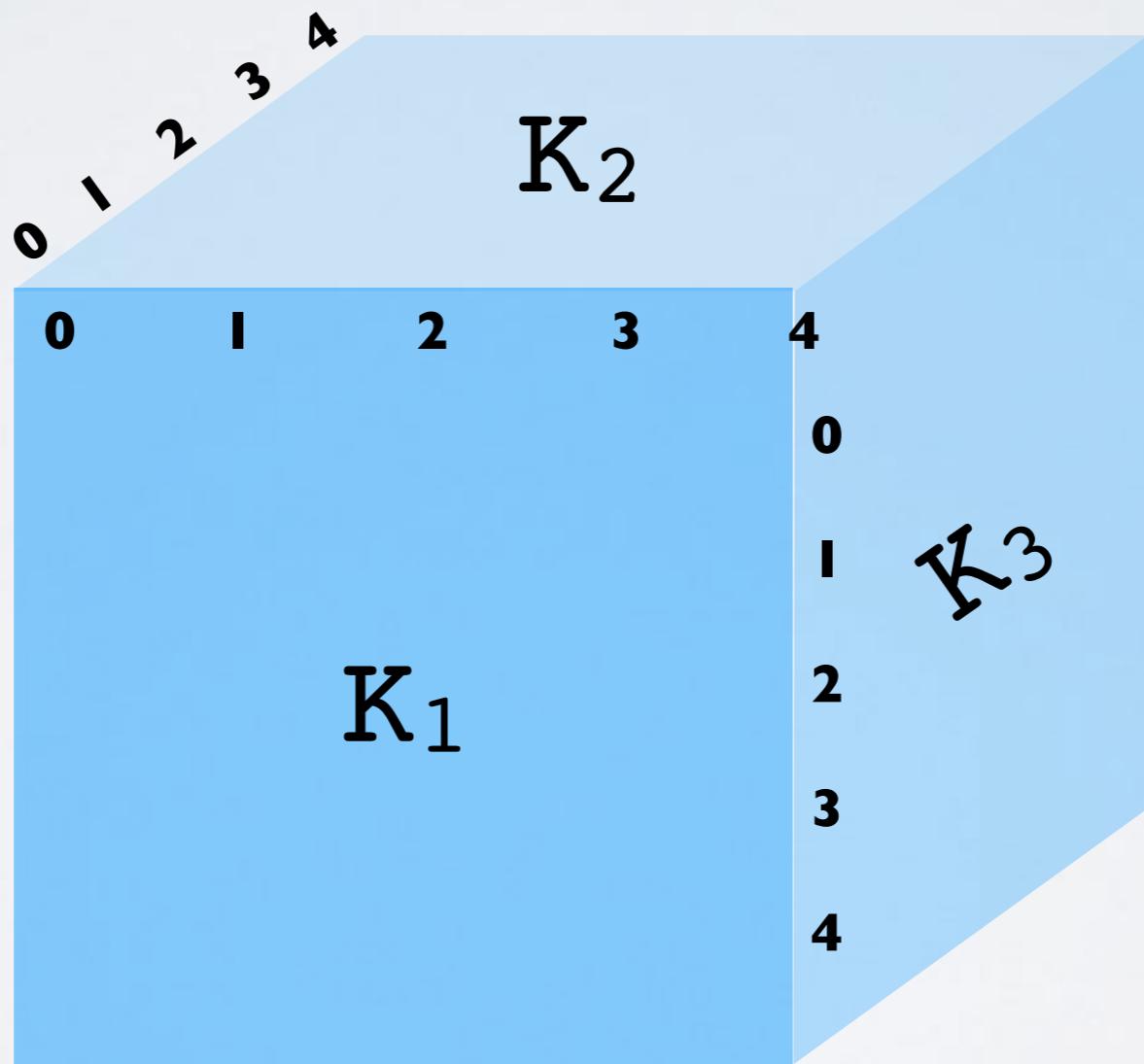
[] (... ($K_1 \geq K_2$) ...)



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON EXPRESIONES CONTADORAS COMPLEJAS

ψ

[] (... ($K_1 \geq K_2 + K_3$) ...)



COUNTING FLUENT LABELLED TRANSITION SYSTEM ANALYZER

CLTSA - SingleLaneBridgeCapacity.lts

File Edit Check Build Window Help Options

Animator

```
red.1.enter
red.1.exit
red.2.enter
red.3.enter
```

Run Step Configure Report...

red.1.enter RED.1 : False
 red.1.exit BLUE.1 : False
 red.2.enter RED.3 : True
 red.2.exit BLUE.3 : False
 red.3.enter RED.2 : True
 red.3.exit BLUE.2 : False

blue.1.enter CARS_ON_BRIDGE : 2 [0..9]
 blue.1.exit
 blue.2.enter
 blue.2.exit
 blue.3.enter
 blue.3.exit

Counting Fluents
CARS_ON_BRIDGE : 2 [0..9]

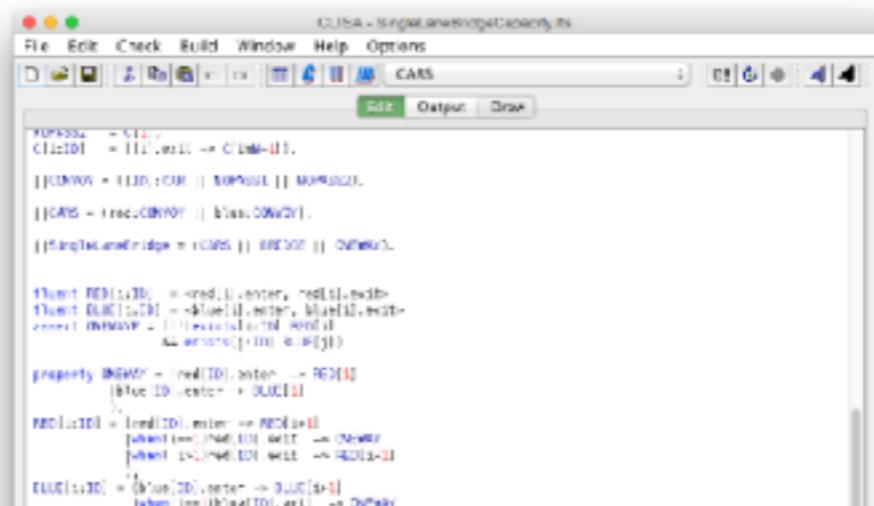
Counting Expressions
CARS_ON_BRIDGE <= 2 : True (2 <= 2)

```
NCROSS2 = C[1].  
C[1][0] = C[1].next -> C[1][0+1]  
| CONVON ((ID1).Cin | NCROSS2 )  
| CARS (red:CONVON | blue:CONVO  
| SingleLaneBridge -> CARS | DRCD  
  
f (len(R-0[i:0]) = <red[i].enter,  
t1:ent C[0][1:20] = <blue[i].enter  
assert ONWMAX (((exists(i)[i:0]  
56 exists(j)[j:0])  
  
property ONEWAY = (red[0].enter  
| blue[0].enter -> E1UE  
  
RED[i:ID1] = (red[ID1].enter -> RED  
when( i==1)red[0].exit  
when( i>1)red[ID1].exit  
  
E1UE[i:ID1] = (blue[ID1].enter -> E1UE  
when( i==1)blue[0].exit -> ONWMAX  
when( i>1)blue[ID1].exit -> E1UE[i-1])  
  
cfluent CARS_ON_BRIDGE = (red[0].enter,blue[0].enter), (red[ID1].exit,blue[ID1].exit), f1 + c  
assert SWF_CAPACITY 11 CARS_ON_BRIDGE <= 2
```



WEB

<http://countingfluent.weebly.com>



The screenshot shows the CLTSA software interface. The title bar reads "CLTSA - singularityproperty.fsp". The menu bar includes File, Edit, Check, Build, Window, Help, Options. The toolbar includes icons for New, Open, Save, Print, Run, Stop, and Exit. The main window displays a state transition graph with nodes like CARS, CAR, CARS', CAR', and transitions labeled with events like RED, GREEN, and BLUE. Below the graph is FSP code:

```
TYPEDEF = CTE;
CLOCK = TID; event > CTE->ID;
||CLOCK = TID; CTR || EXPRESSION || NUMBER;
||CARS = freq(CLOCK) || max(CLOCK);
||StringToLabelBridge = CARS || EXPRESSION || NUMBER;
```

The code defines types, events, and properties related to counting fluents and system states.

Counting Fluent Syntax

cfluent name=<{inc_set},{dec_set},{res_set}> initially int

where *name* is the counting fluent
Identification, *inc_set* (*dec_set* resp.) is the incremental events set, i.e., those whose occurrence increment
and *res_set* is the reset

Abrir "countingfluent.weebly.com" en una pestaña nueva [one](#). And *res_set* is the reset