Diseño y Verificación de Programas Concurrentes

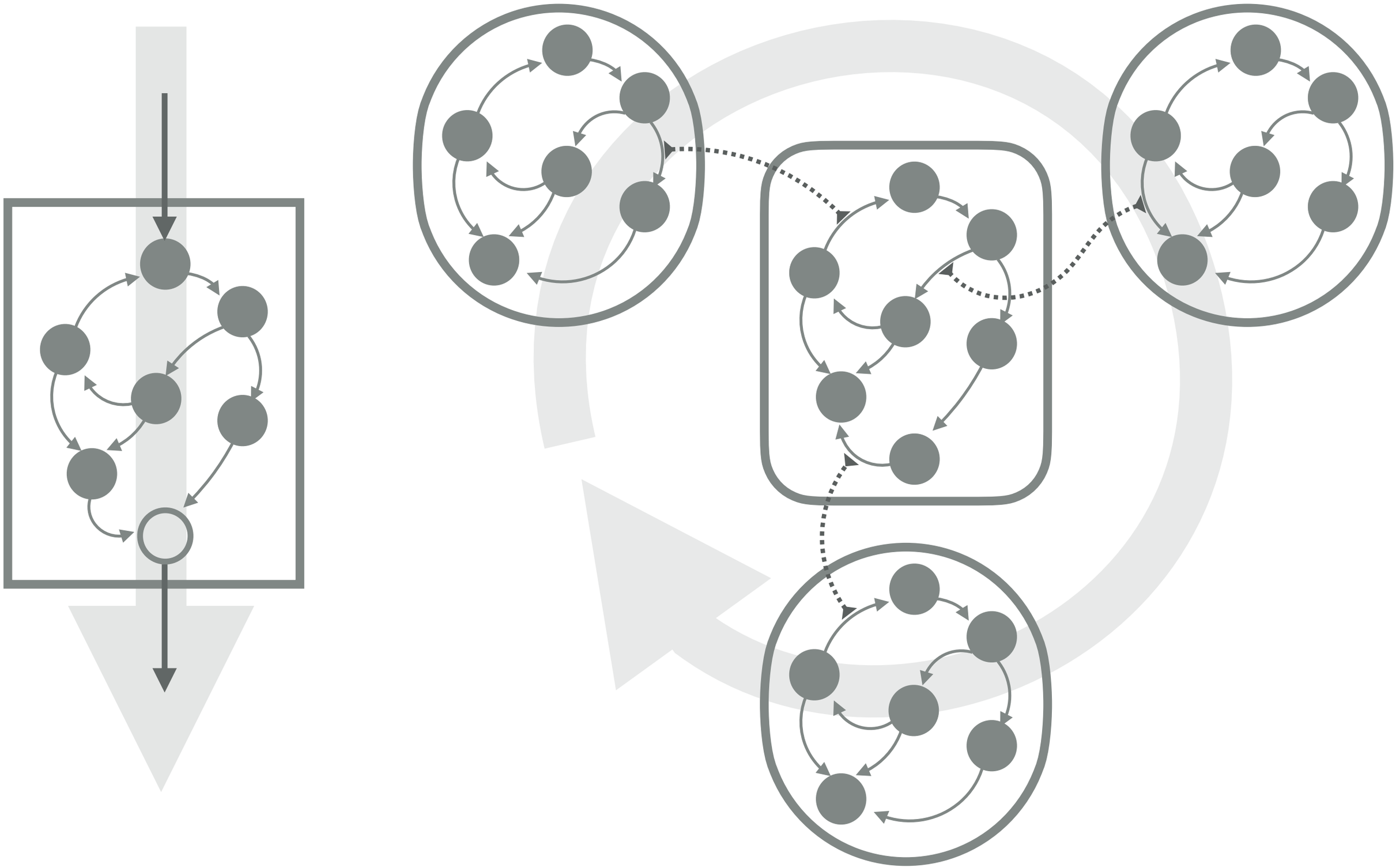
Escuela de Informática - CACIC 2021

ANALIZANDO MODELOS DE SISTEMAS REACTIVOS

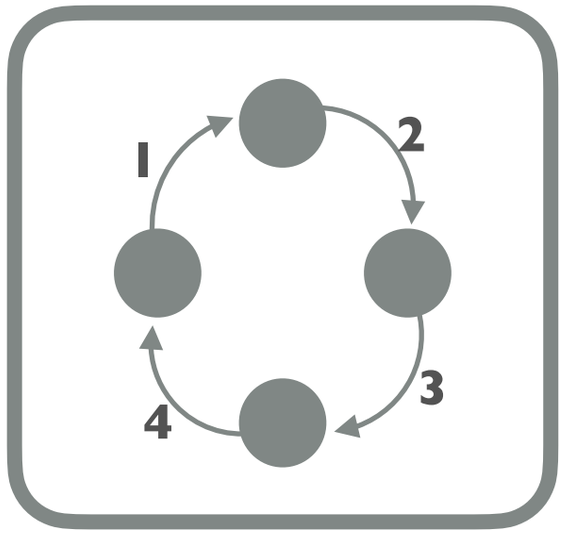
INFORMACIÓN Y MODELOS

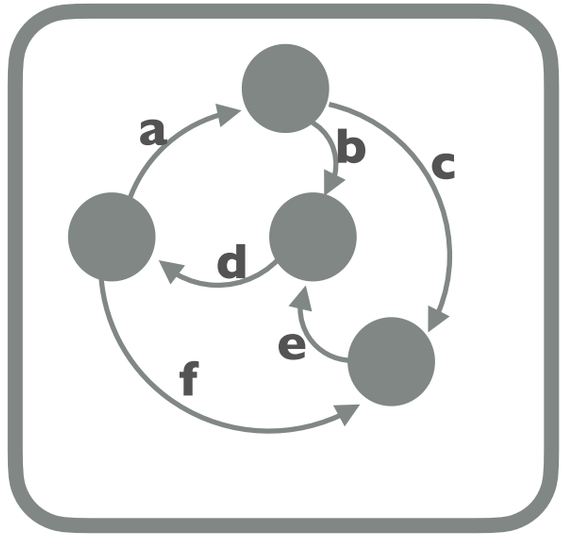
Estancia la Guitarra - Córdoba

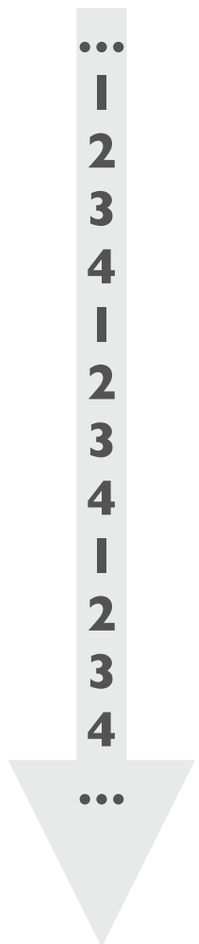
COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS



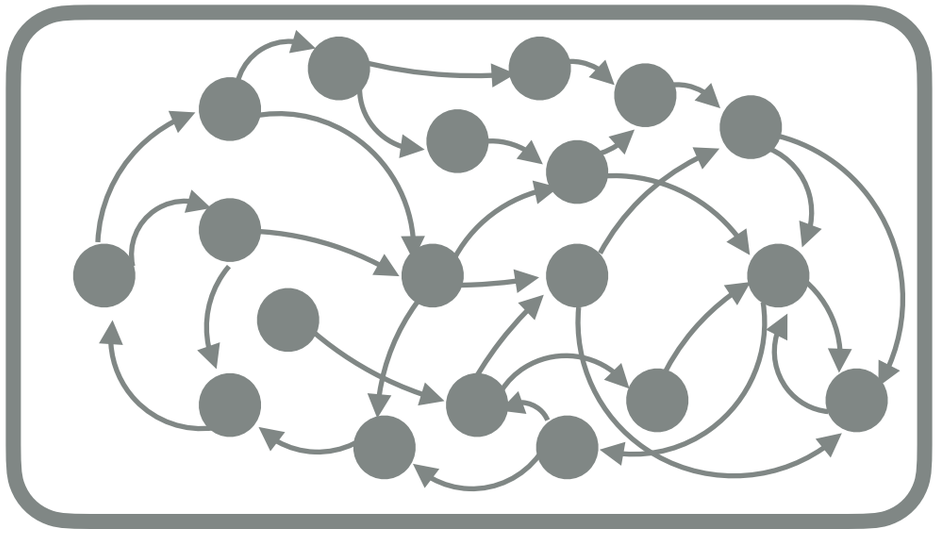
COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS

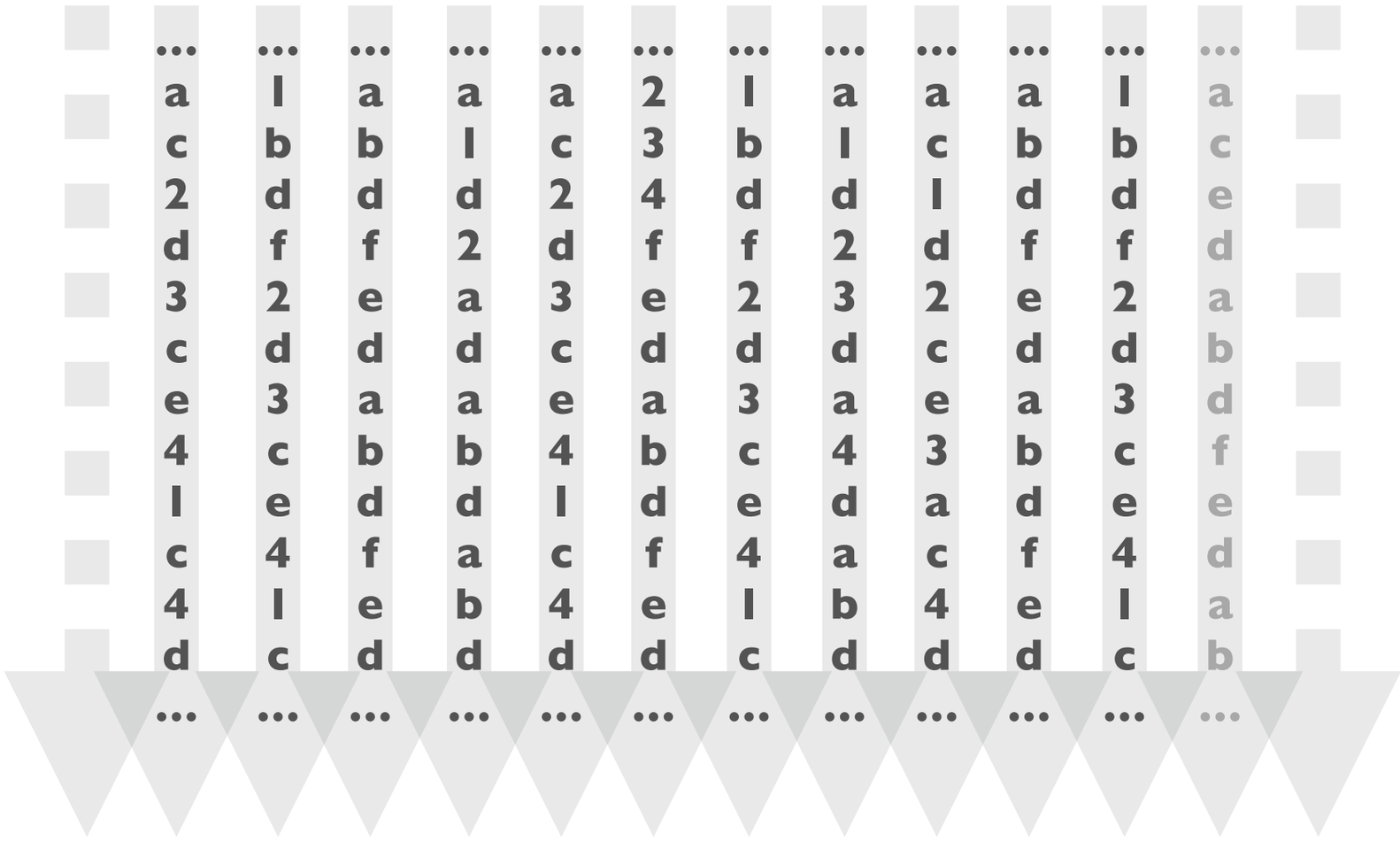




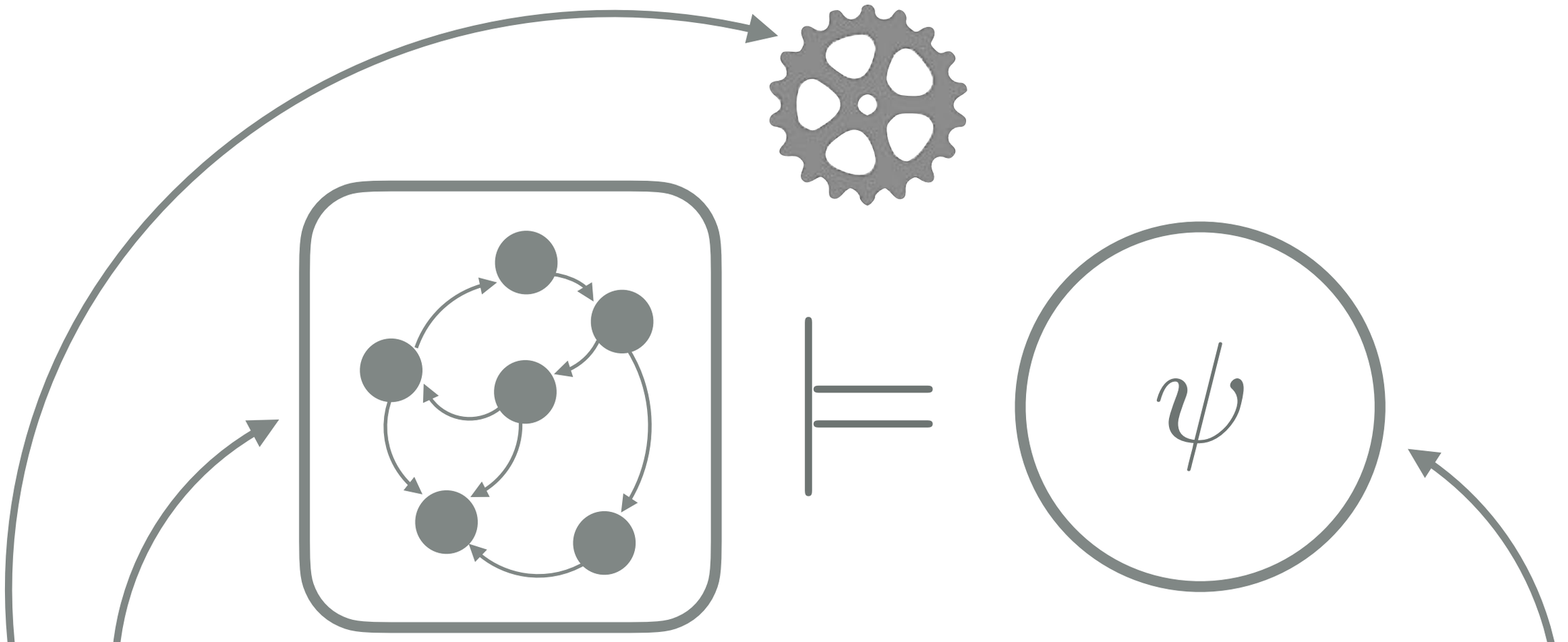


COMPORTAMIENTO ABSTRACTO DE SISTEMAS REACTIVOS



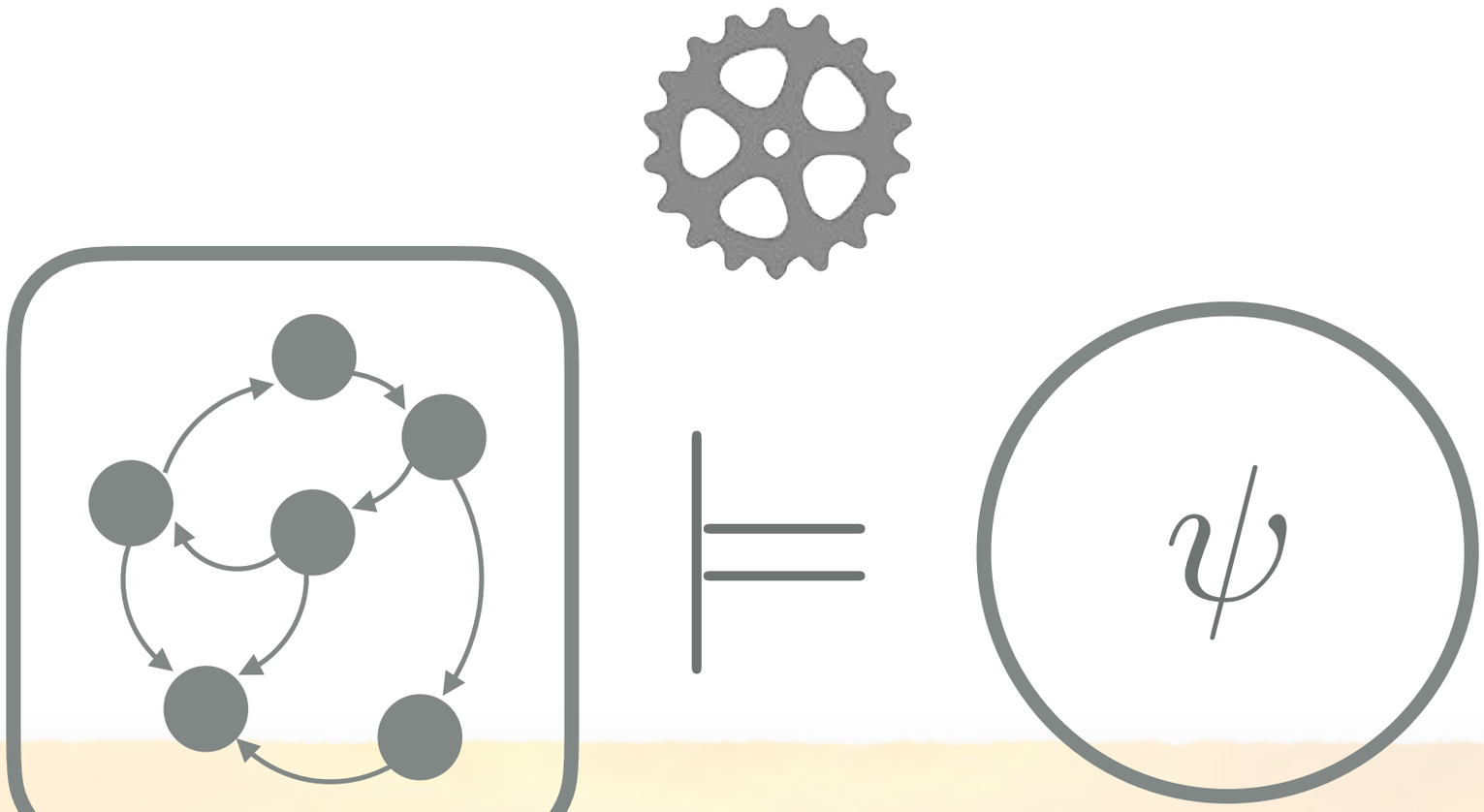


COMO ANALIZAR (VERIFICAR PROPIEDADES) DE SISTEMAS REACTIVOS



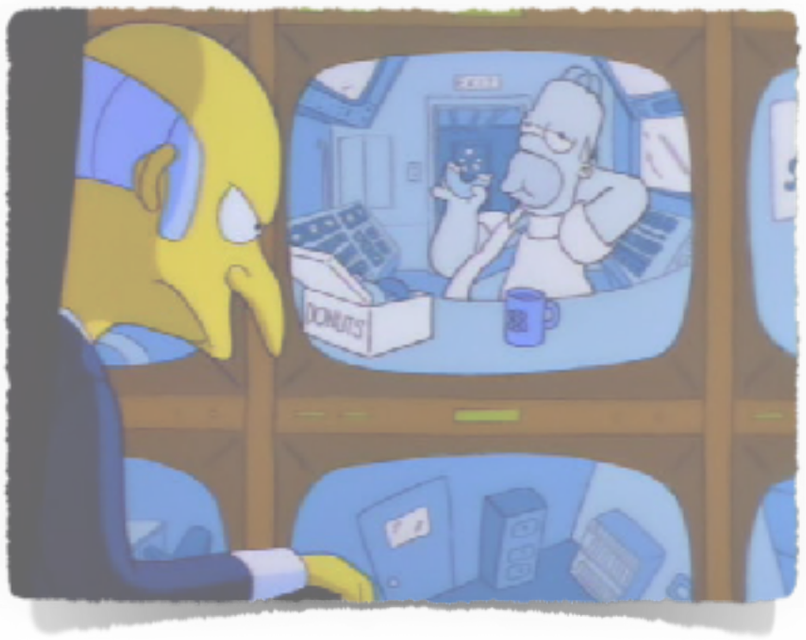
- Lenguaje para describir los modelos (Sistema)
- Lenguaje para especificar las propiedades a analizar
- Mecanismo (automático) para verificar las propiedades

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 = (...).
```

P

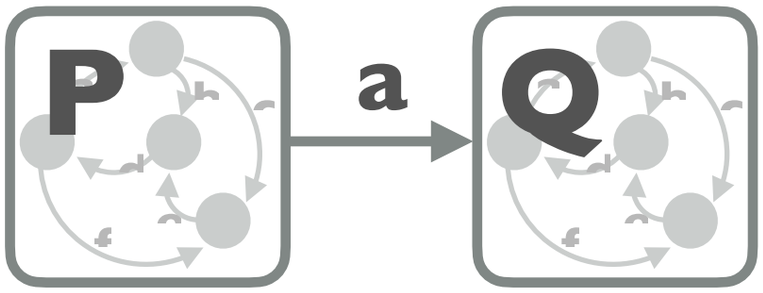
a

...

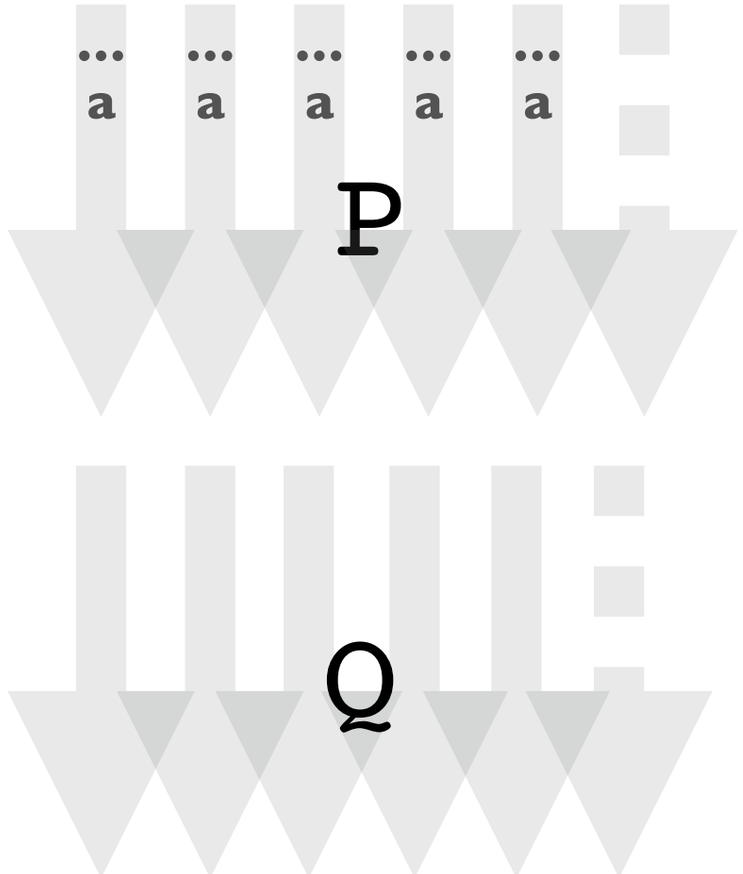
```
a -> ...
```

...
a

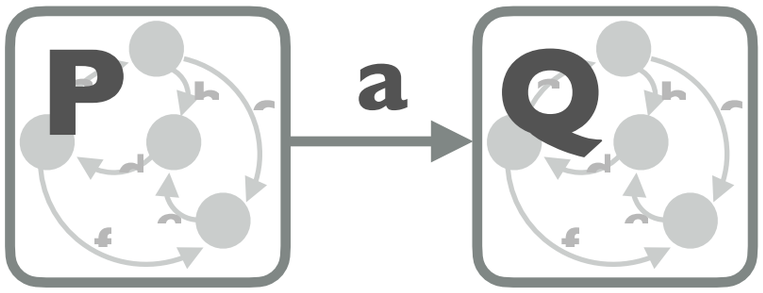
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)




```
P = (… a -> Q), Q = (…)
```



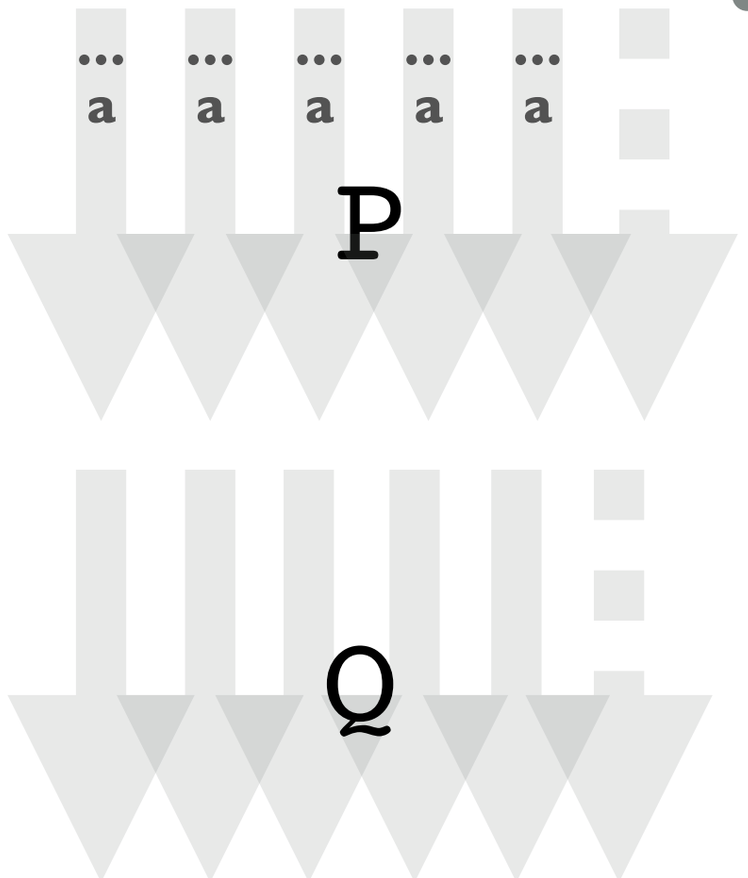
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



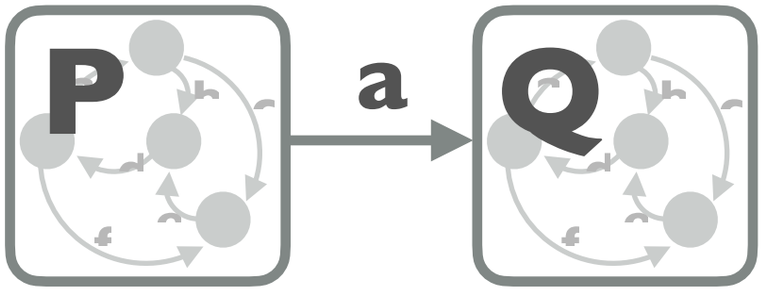

```
P = (… a -> Q), Q = (…)
```

Recursión

```
Q = (… -> P).
```



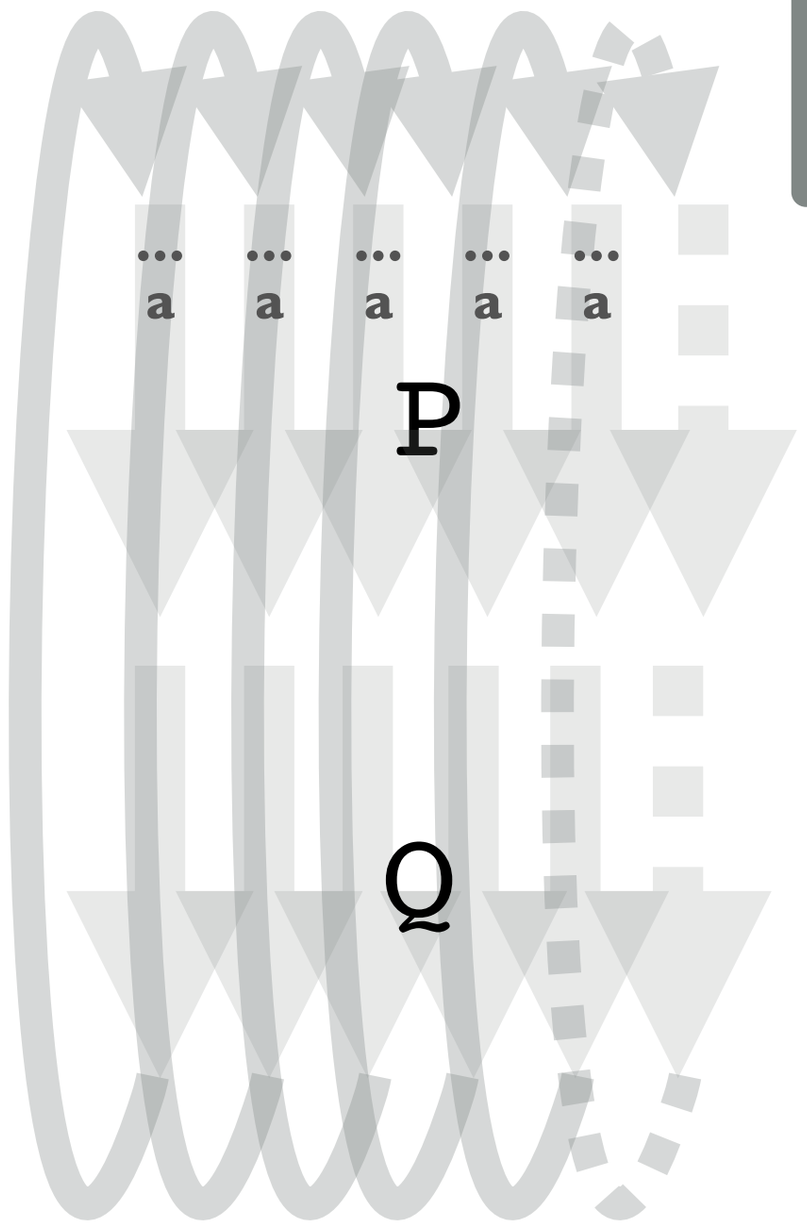
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



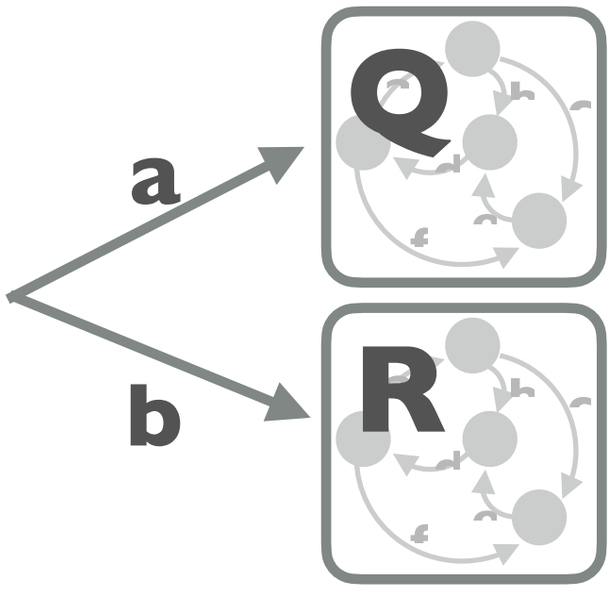

```
P = (… a -> Q), Q = (…)
```

Recursión

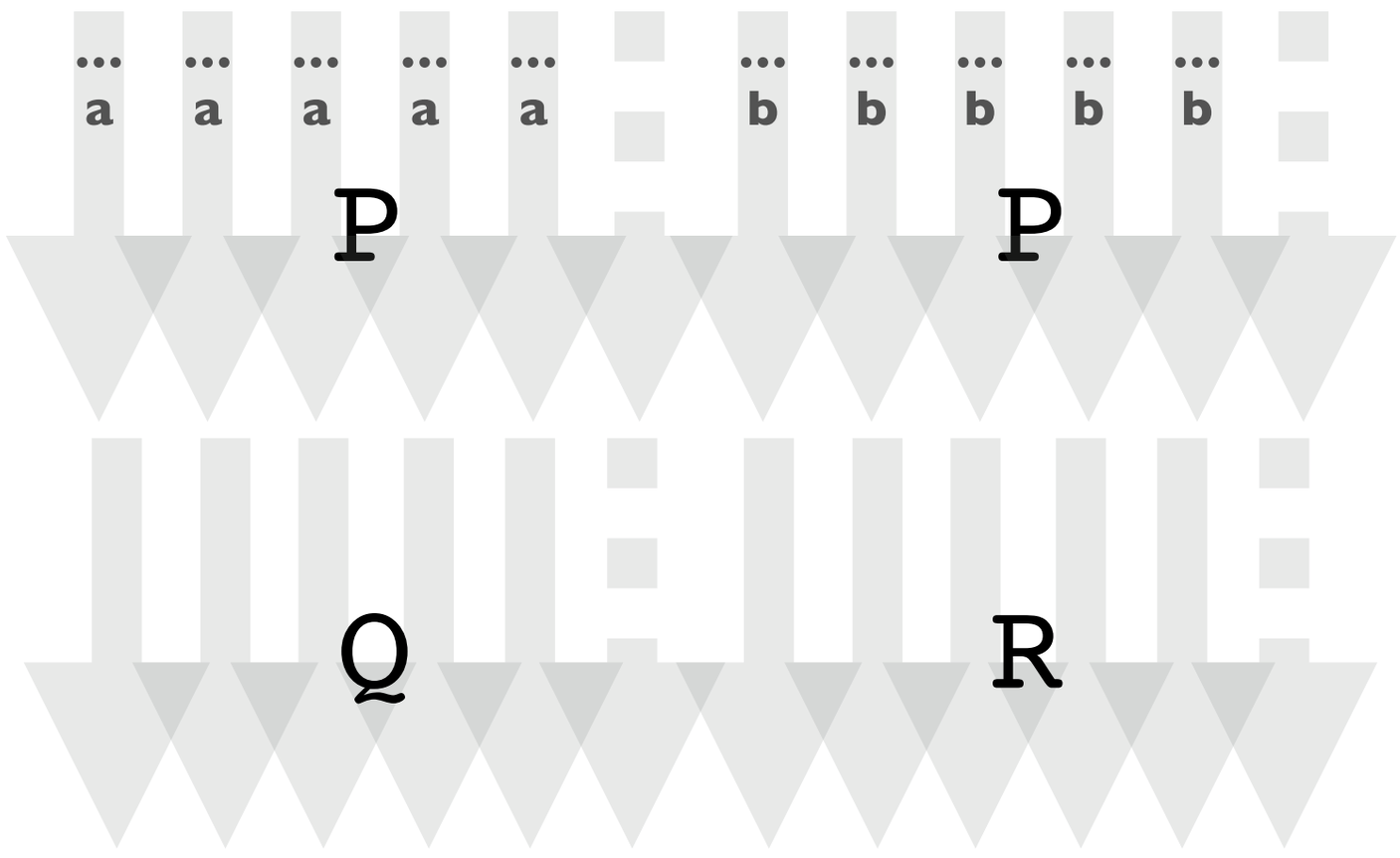
```
Q = (… -> P).
```



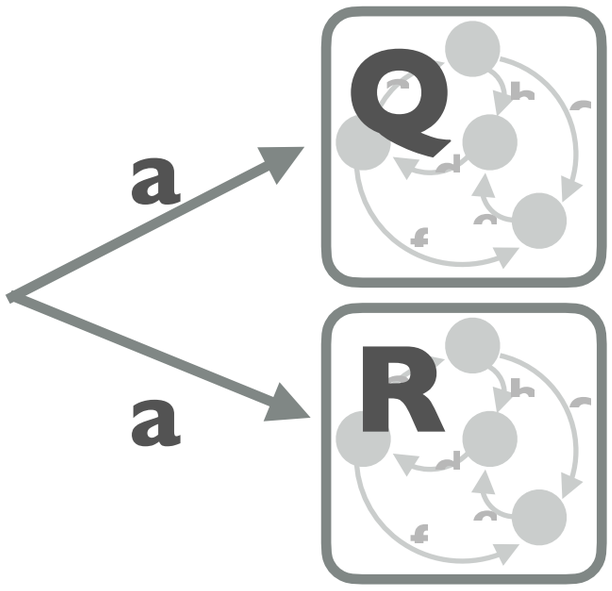
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)




```
…(a -> Q | b -> R)
```



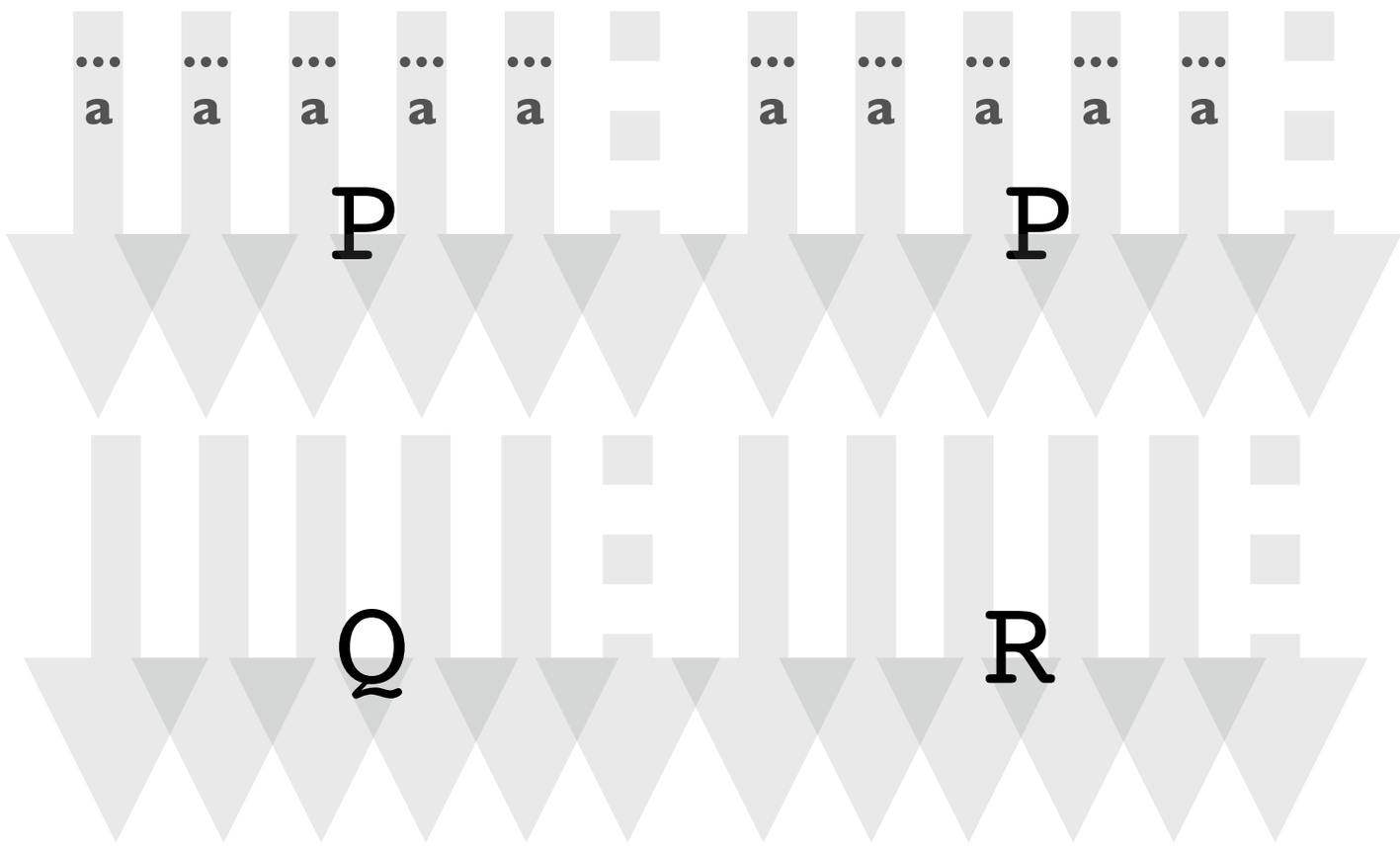
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



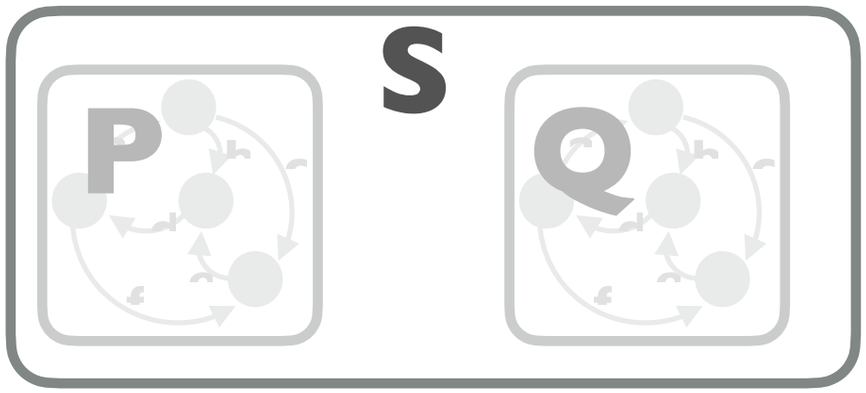

```
…(a -> Q | b -> R)
```

No Determinismo

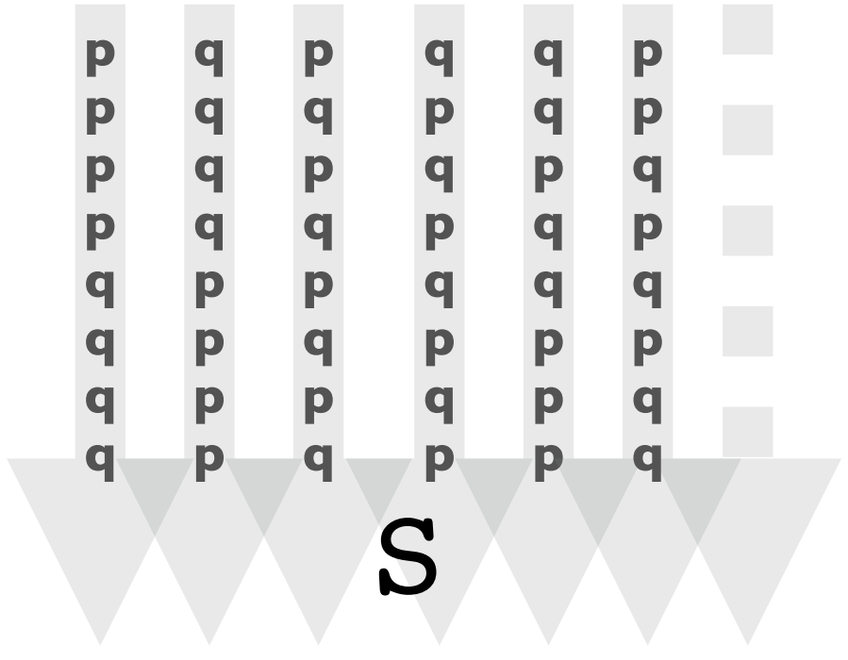
```
| a -> R).
```



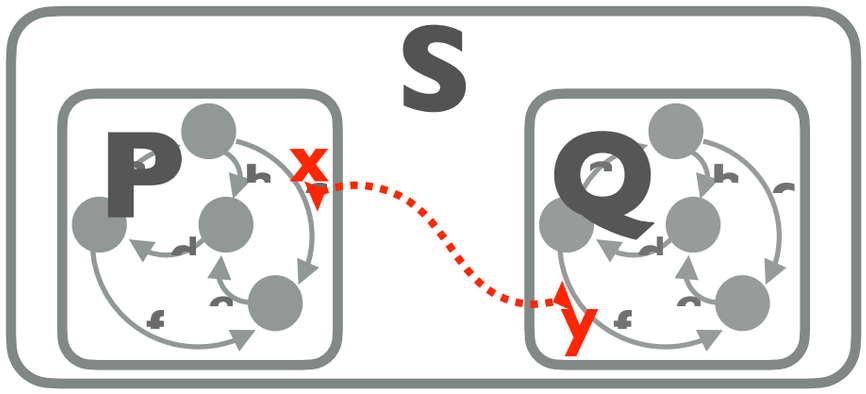
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)



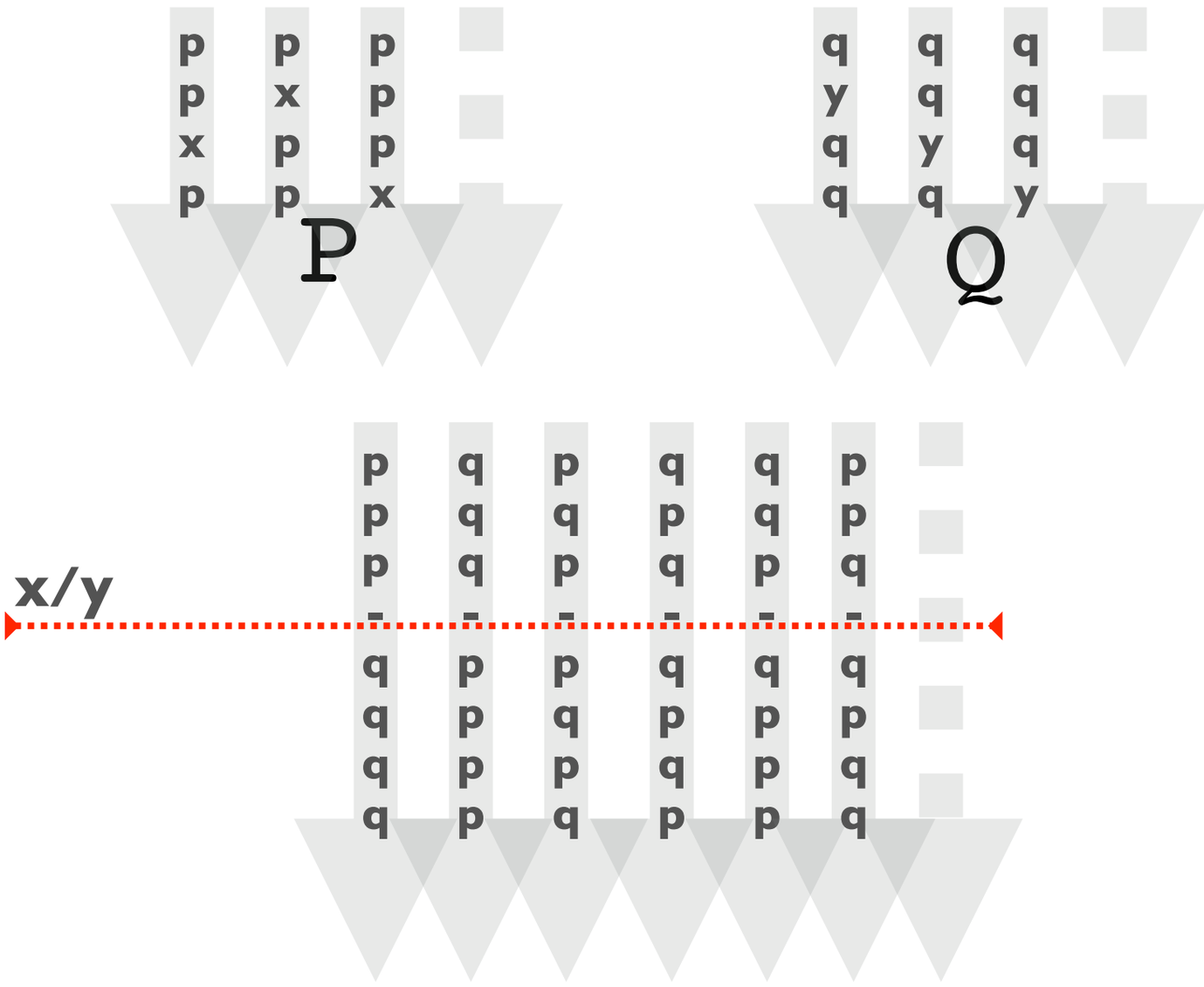

```
||S = (P || Q).
```



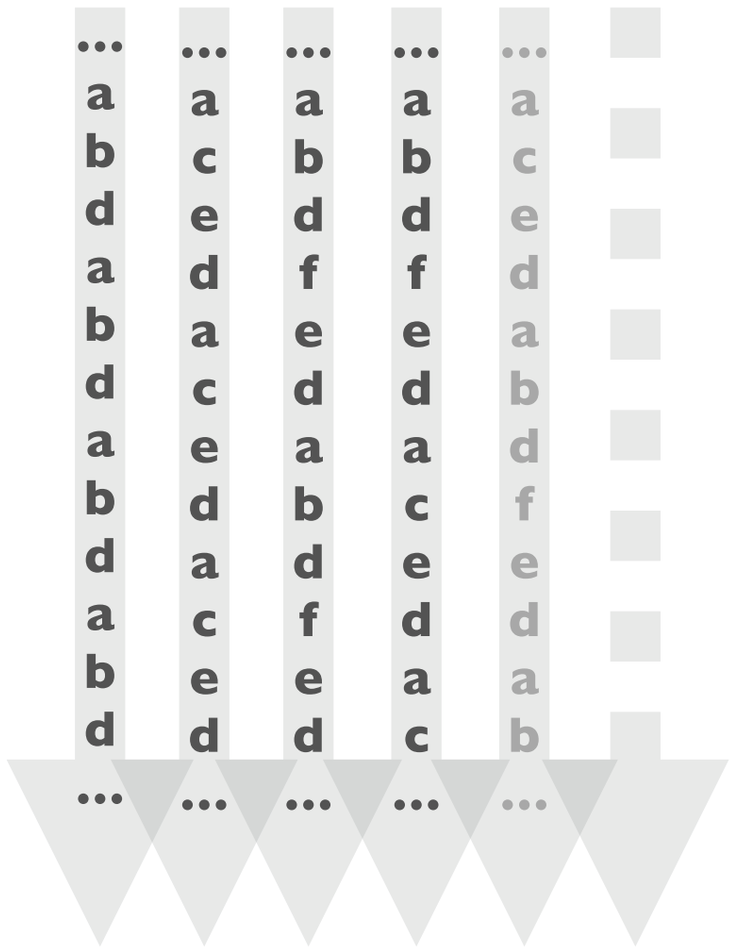
LENGUAJE DE MODELADO DE SISTEMAS REACTIVOS (FSP)




```
||S = (P || Q){P.x/Q.y}.
```



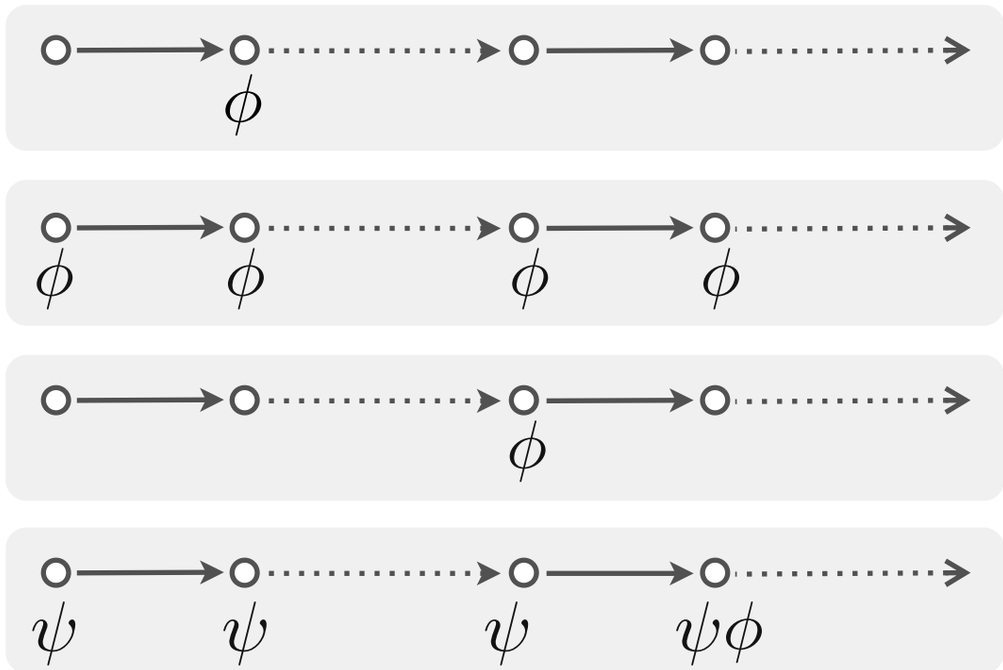
CÓMO CAPTURAR PROPIEDADES DE TRAZAS (INFINITAS)



CÓMO CAPTURAR PROPIEDADES DE TRAZAS (INFINITAS)

CÓMO CAPTURAR PROPIEDADES DE TRAZAS (INFINITAS)

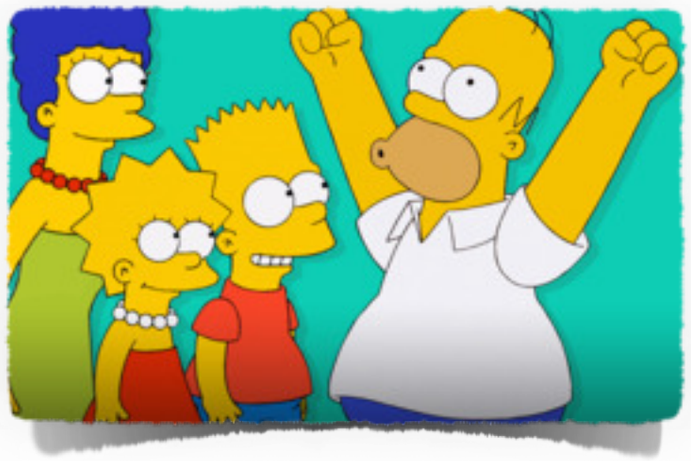
- Next: $\mathbf{X}\phi$
- Globally $\mathbf{G}\phi$
- Finally $\mathbf{F}\phi$
- Until $\psi \mathbf{U} \phi$



PATRONES DE PROPIEDADES **LTL**

Safety: “algo malo no va a ocurrir"

$$[\,]\ \phi$$



Liveness: “algo bueno ocurrirá siempre"

$$[\,]\langle\rangle\ \phi$$

ψ

PATRONES DE PROPIEDADES **LTL**

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

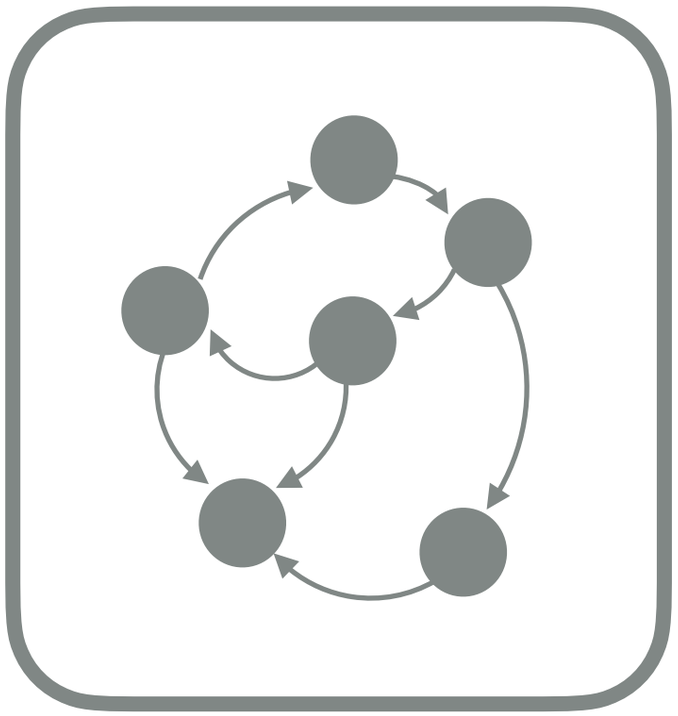
- **Ocurrencia**
  - Ausencia
  - Universal
  - Existencia
  - Existencia acotada

- **Orden**
  - Precedencia
  - Respuesta
  - Cadena de Precedencia
  - Cadena de Respuesta

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

MODEL CHECKING

¿ ⚙ ?



$\mathcal{M}$

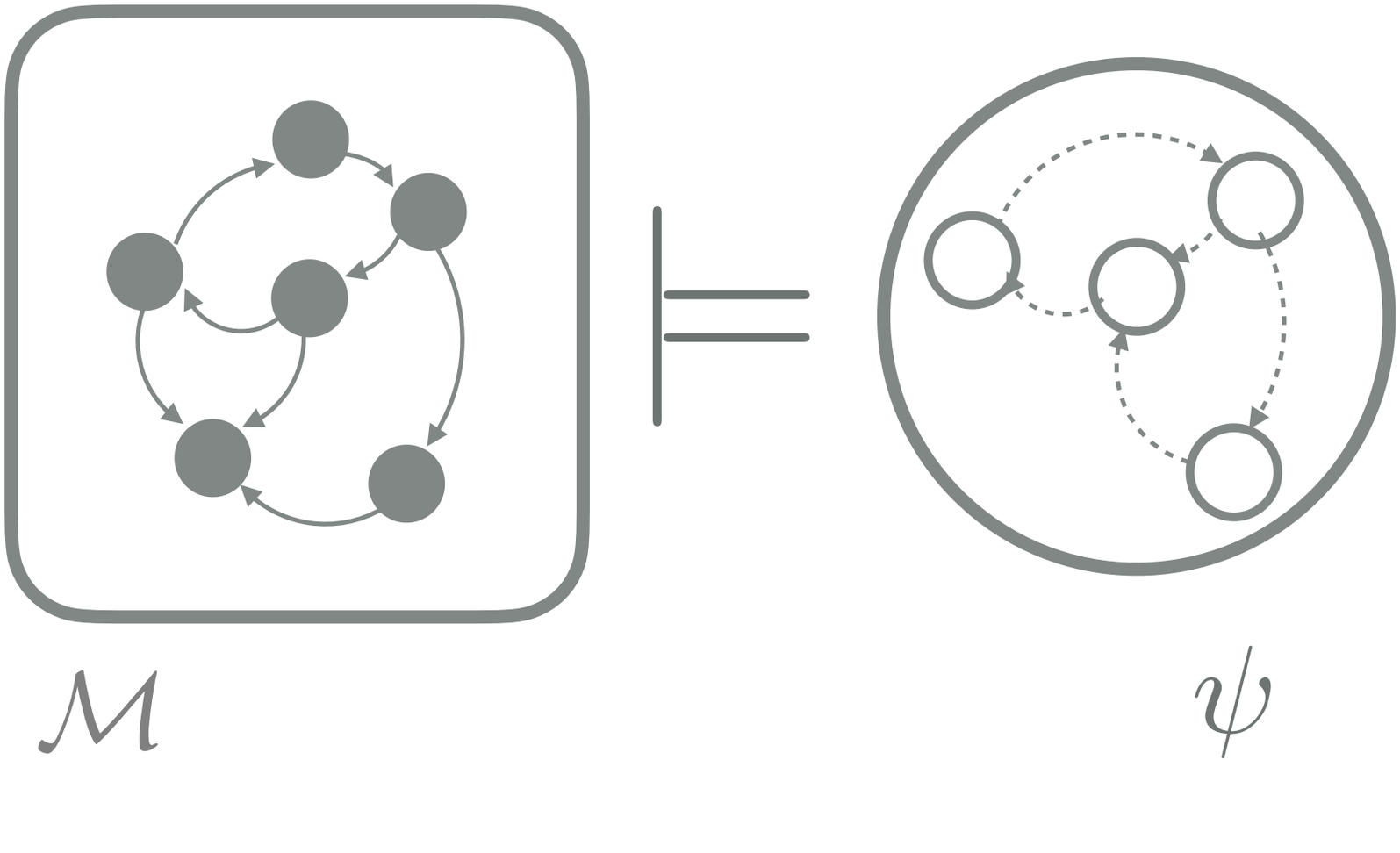
$\models$

ψ

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

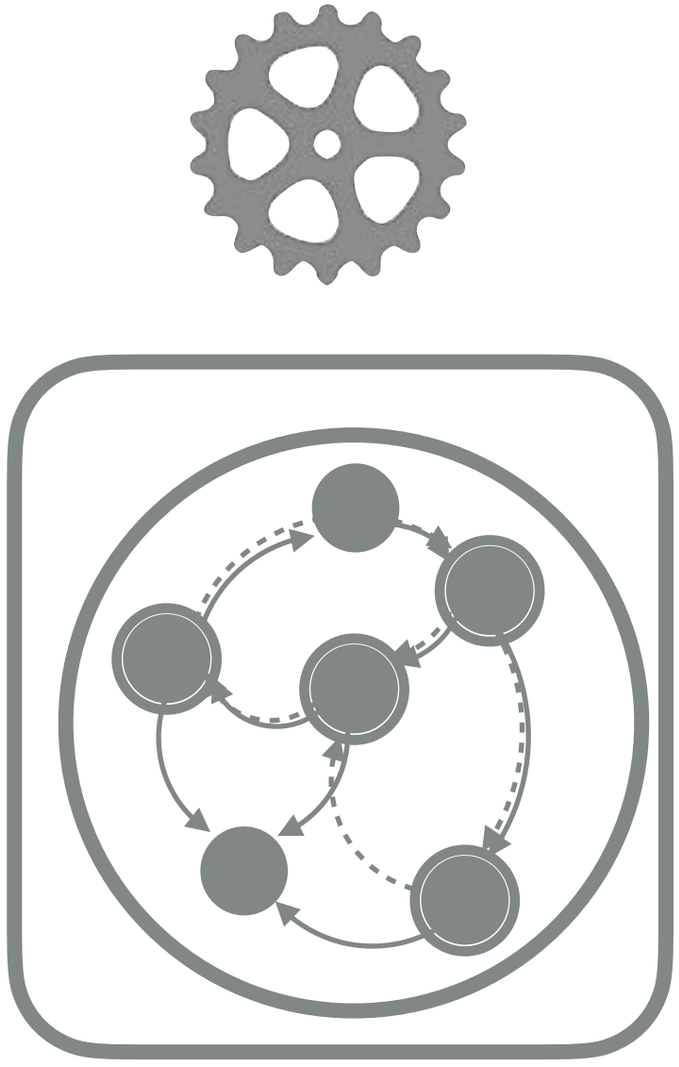
MODEL CHECKING

¿ ⚙ ?



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

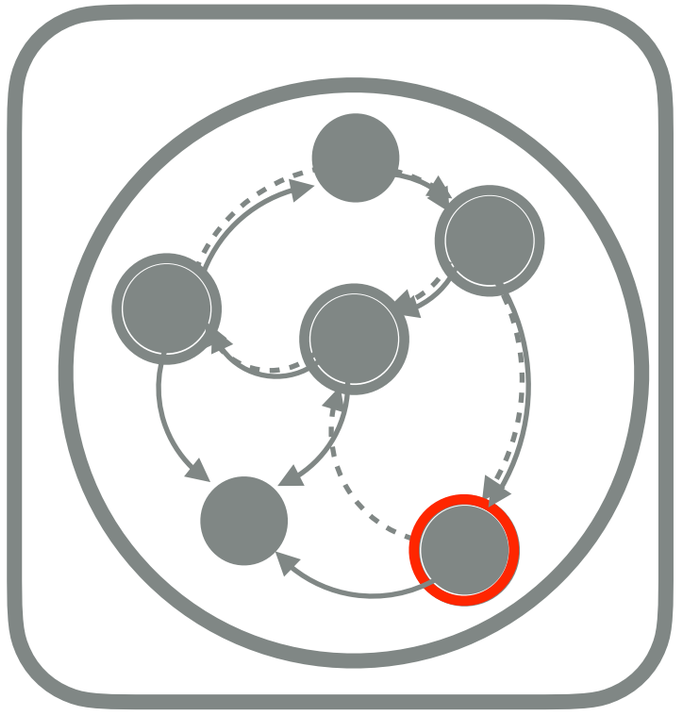
MODEL CHECKING



$\mathcal{M}$ ψ

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

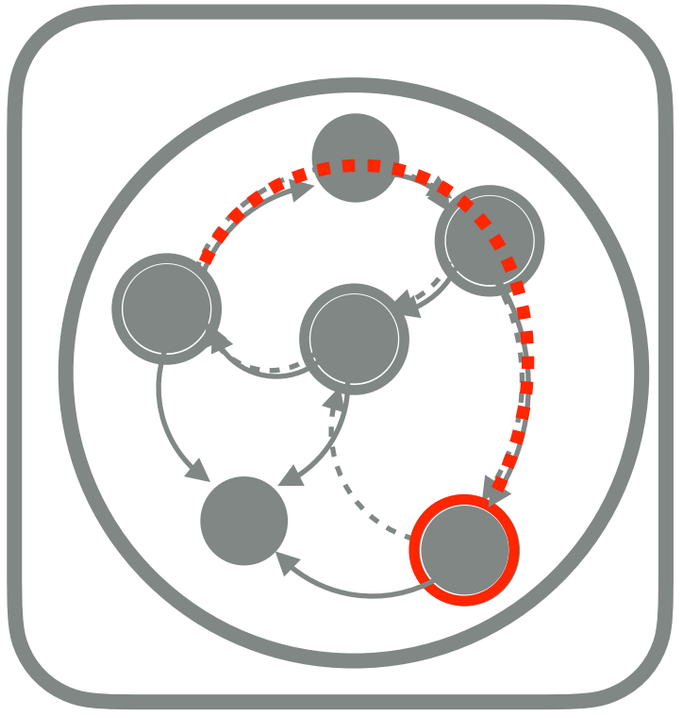
MODEL CHECKING



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

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

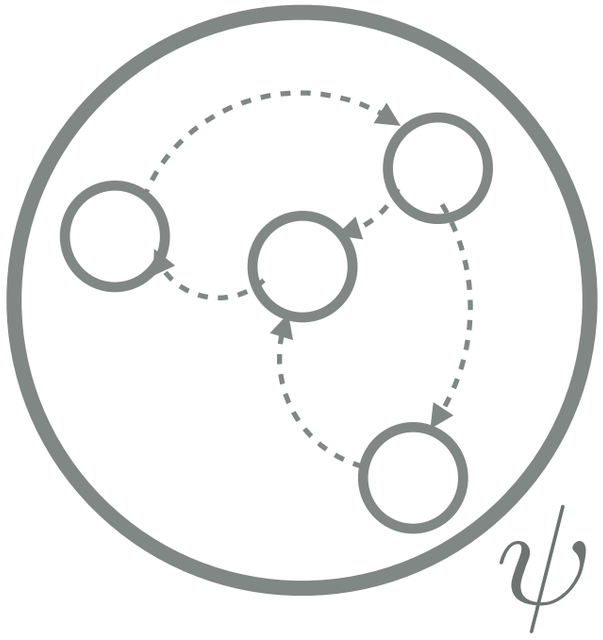
MODEL CHECKING



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

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

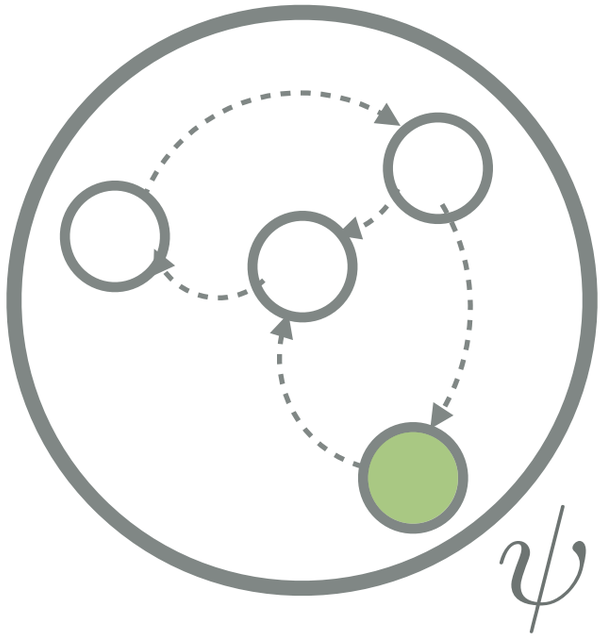
MODEL CHECKING



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

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

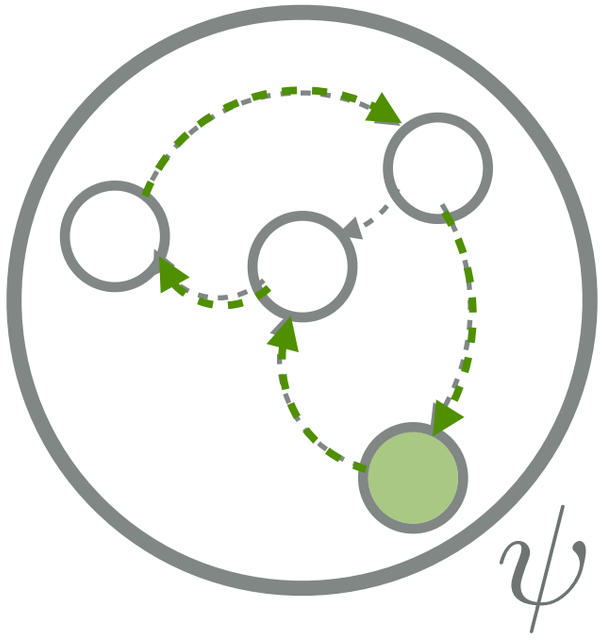
MODEL CHECKING



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

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES

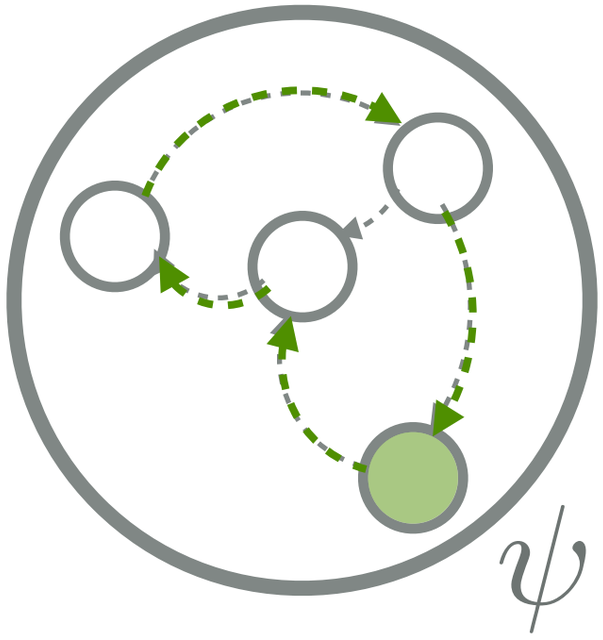
MODEL CHECKING



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

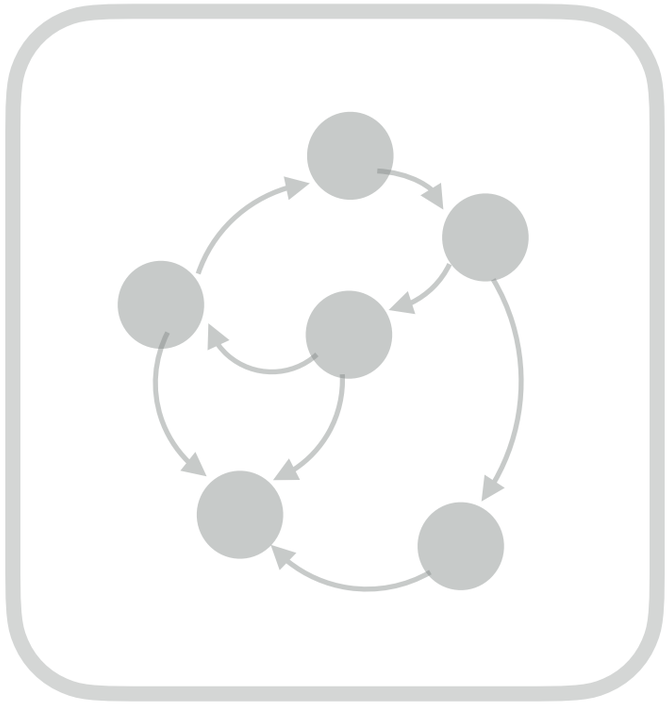


a b d a b d a b d d a a b d a d a d d

Autómata de Büchi:

"una cadena es aceptada si infinitamente pasa por estados de aceptación"

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **SAFETY**



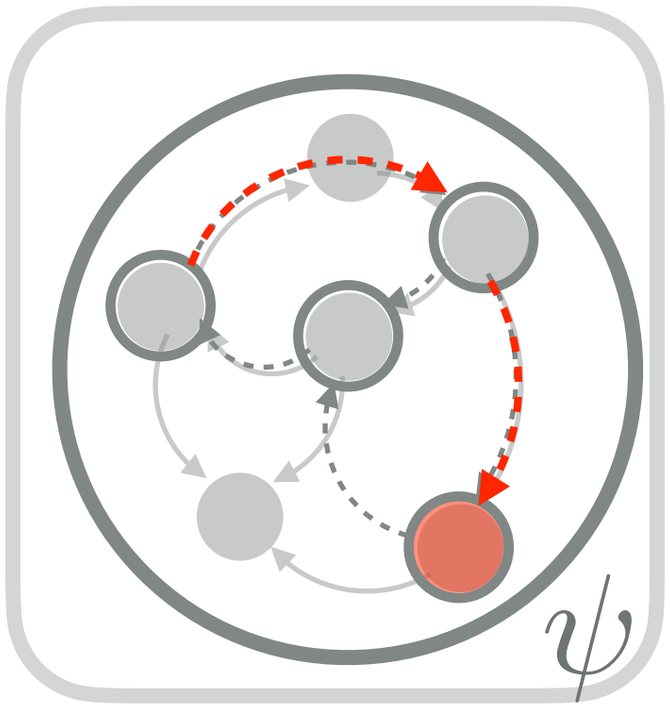
a b d a b d a b d d a a b d a d b b b

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**



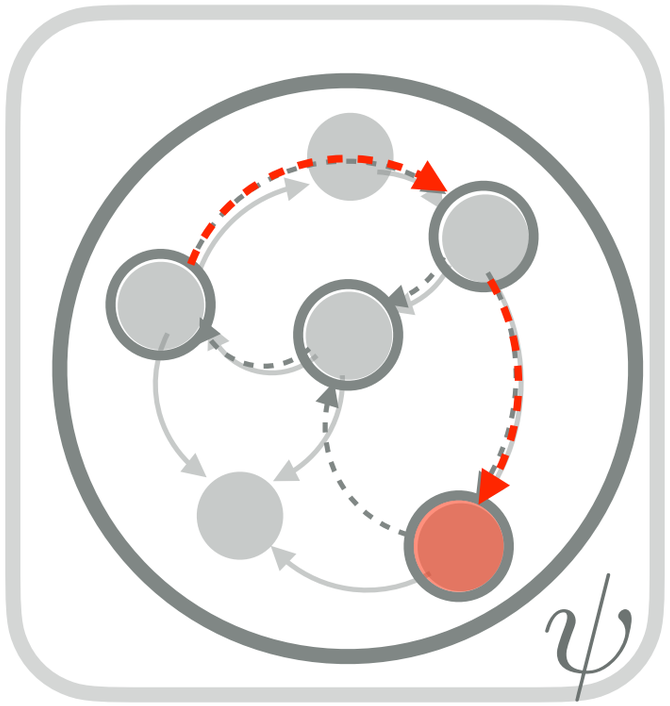
a b d a b d a b d d a a b d a d b b b

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**



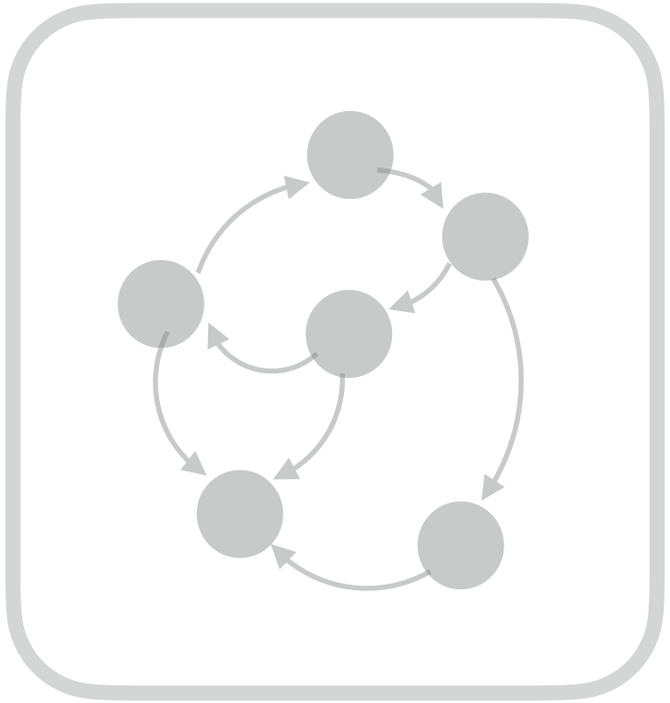
a b d a b d a b d d a a b d a d b b b

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**



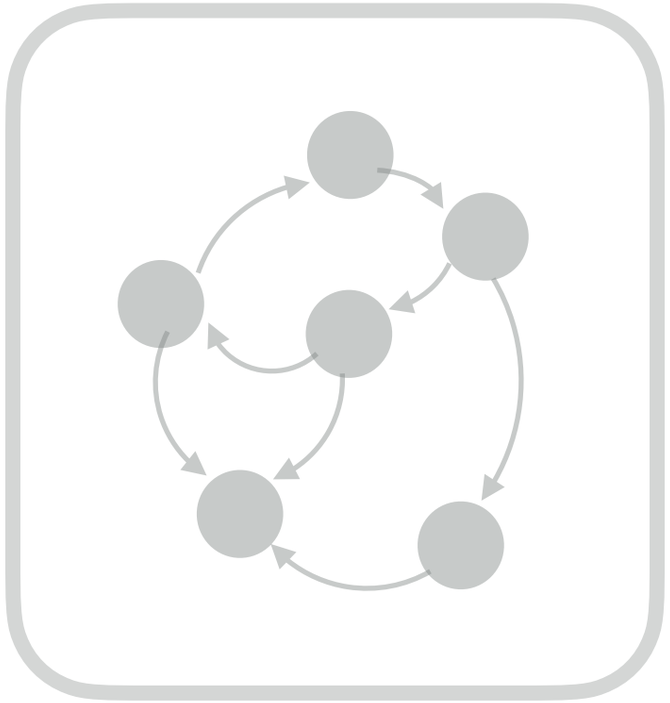
a b d a b d a b d d a a b d a d b

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**



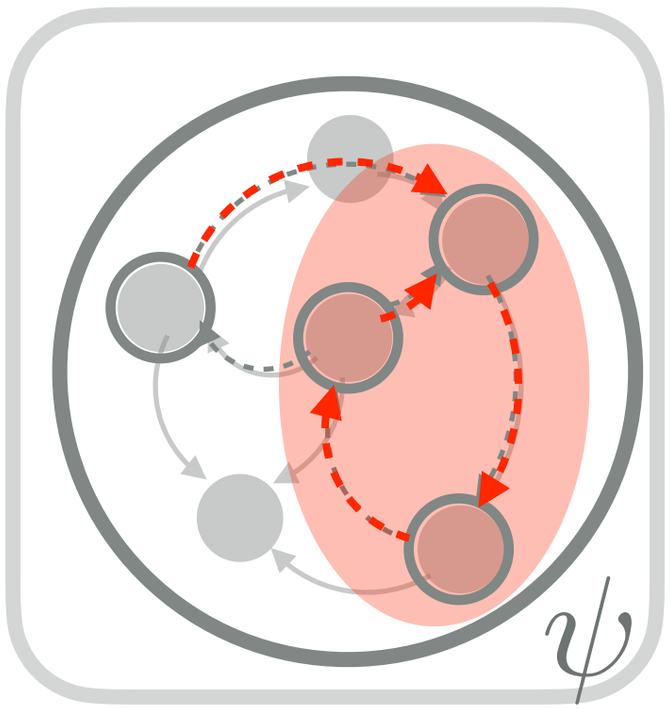
a b d a b d a b d d a a b d a d b

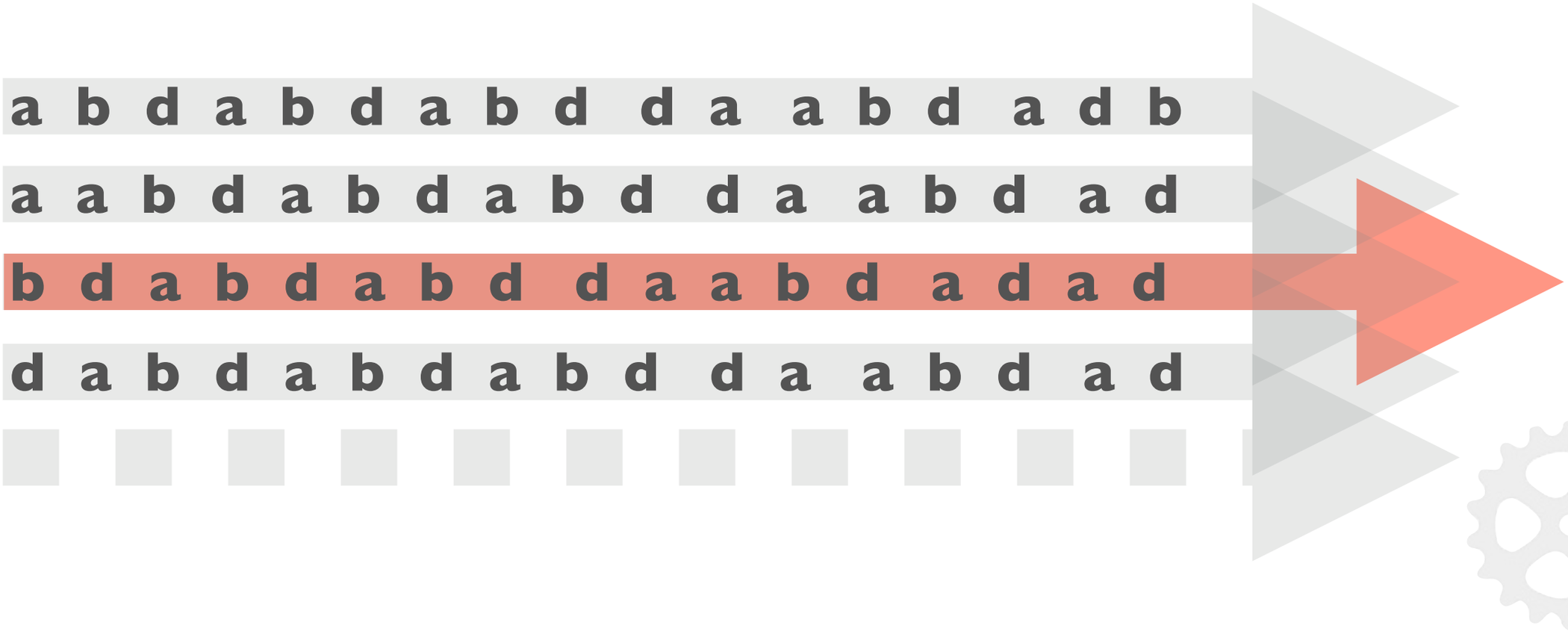
a a b d a b d a b d d a a b d a d

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**





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 ...)`

Giannakopoulou, Kramer "*Fluent model checking for event-based systems*"

AYUDA A MEMORIA (FLUENTES)

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**

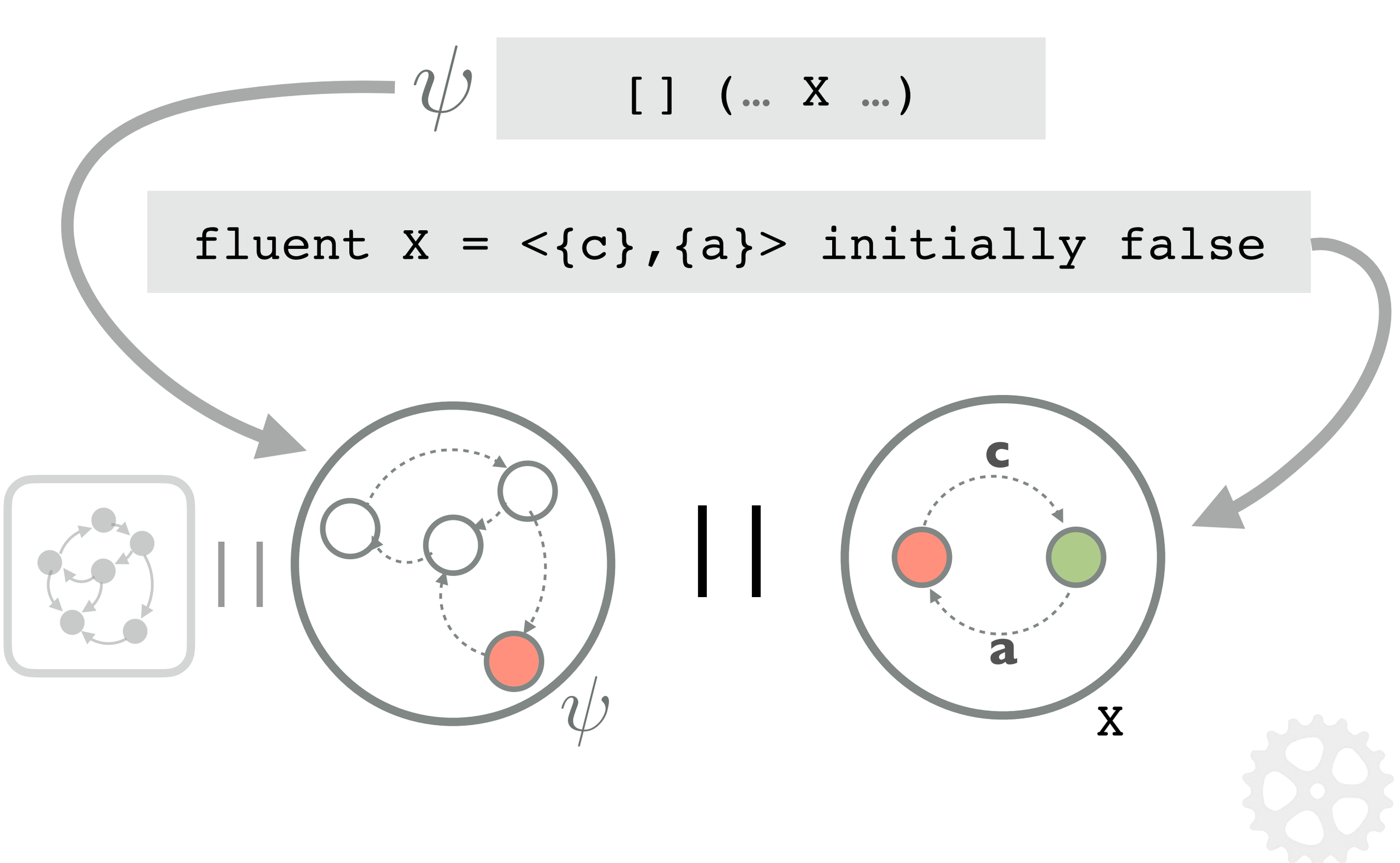
ψ `[] (... X ...)`

Giannakopoulou, Kramer "*Fluent model checking for event-based systems*"

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES CON **FLUENTES PROPOSICIONALES**

ψ `[] (… X …)`

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



CUANDO UN BIT DE MEMORIA NO ALCANZA

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

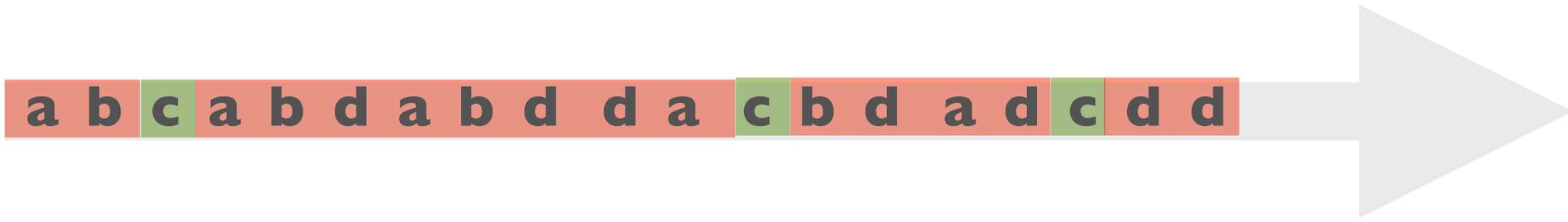
Algunos ejemplos de tomados de la literatura:

- 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

ALGUNA 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

ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES



ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

ψ ...(¬ c U (c U (¬ c U (c U (¬ c U (c U ...))...)

1 2 3

ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

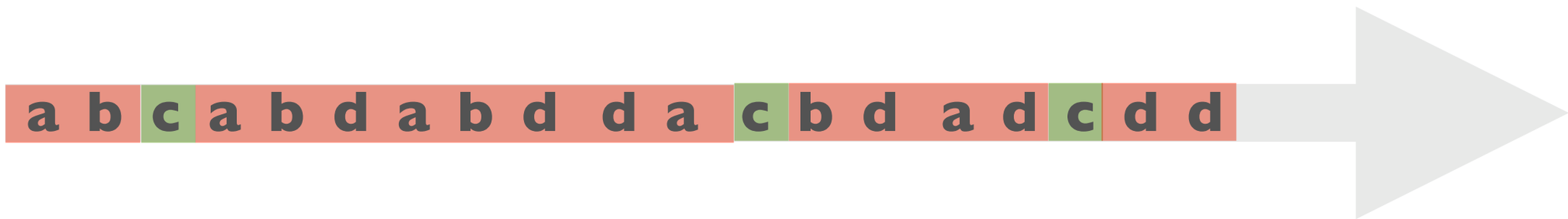
ψ ...(¬ c U (c U (¬ c U (c U (¬ c U (c U ...))...)

1 **2** **3**

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

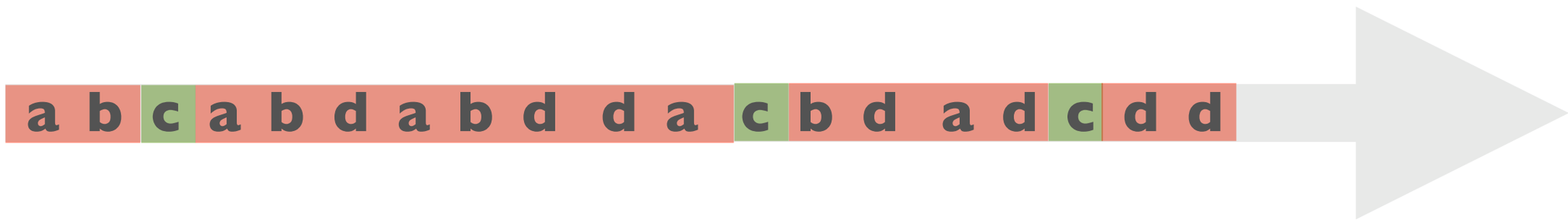
ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

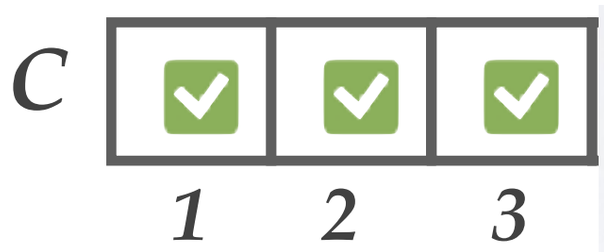
Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales



ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

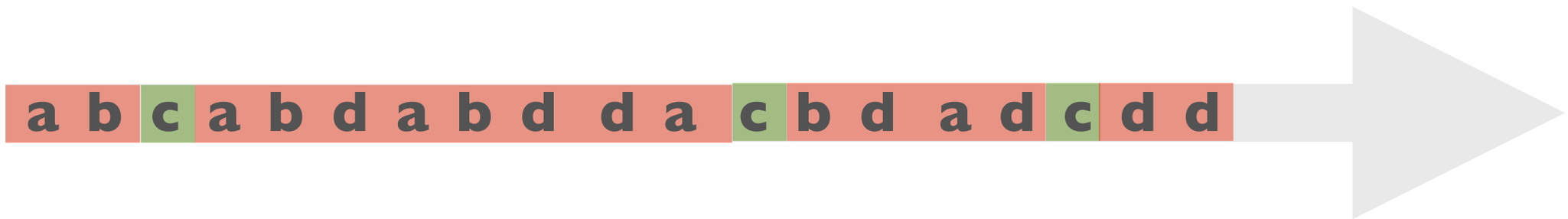
Si podemos identificar cada elemento a contar podemos usar arreglos de fluentes proposicionales

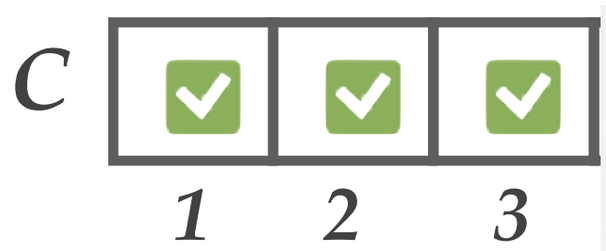




ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

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

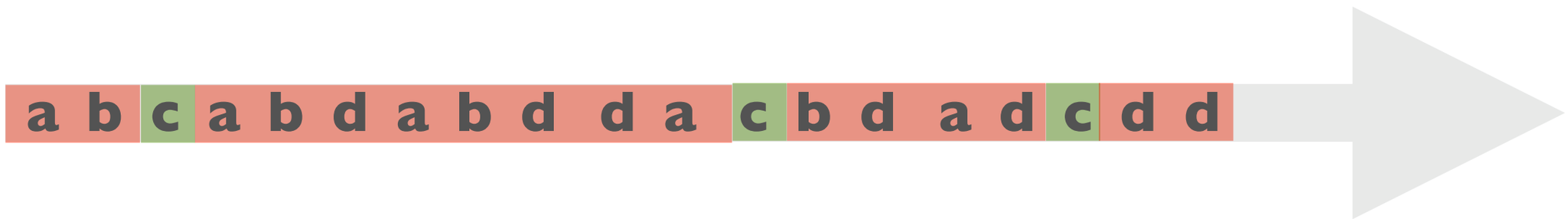


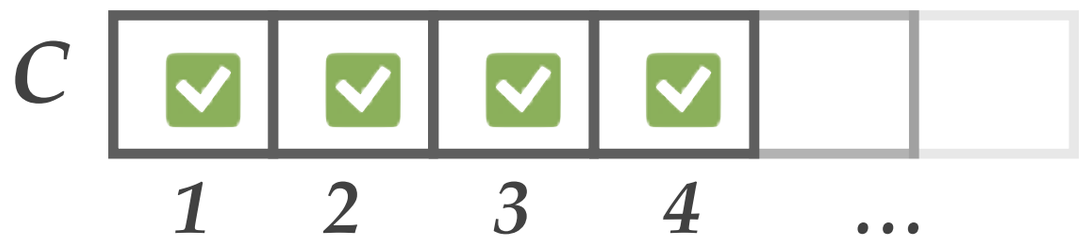


ψ ...($C_1 \wedge C_2 \wedge C_3$)...

ALGUNA FORMAS PARA ABORDAR ESTE TIPO DE PROPIEDADES

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





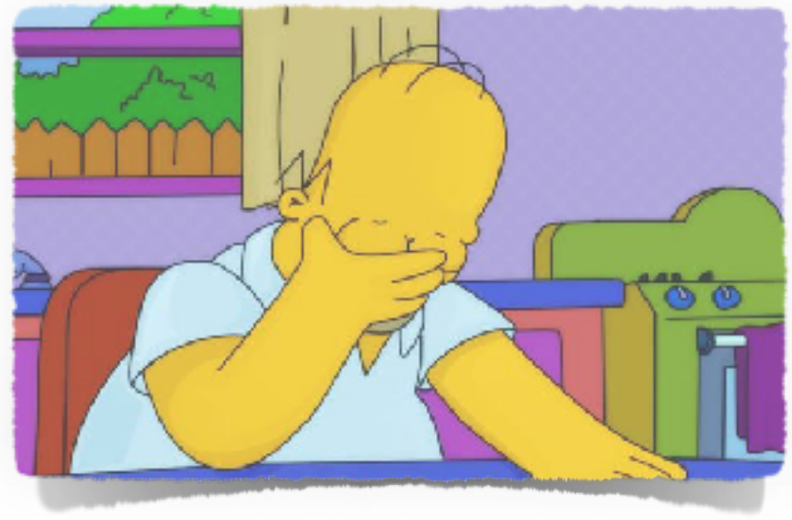
ψ ...($c_1 \wedge c_2 \wedge c_3$) $\vee$ ($c_1 \wedge c_2 \wedge c_4$) $\vee$ ($c_1 \wedge c_3 \wedge c_4$) .

LTL CON FLUENTES PROPOSICIONALES NO ES SUFICIENTE

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

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

¿ ψ **...(#a≥# c)...** ?

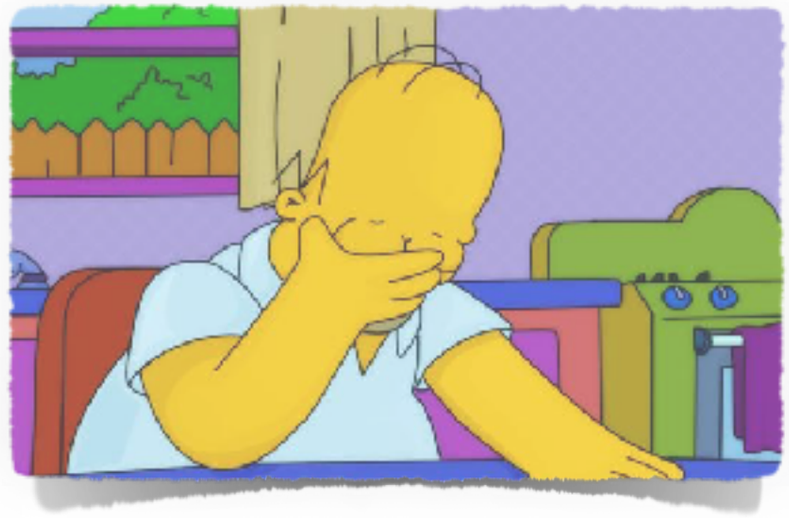


LTL CON FLUENTES PROPOSICIONALES NO ES SUFICIENTE

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

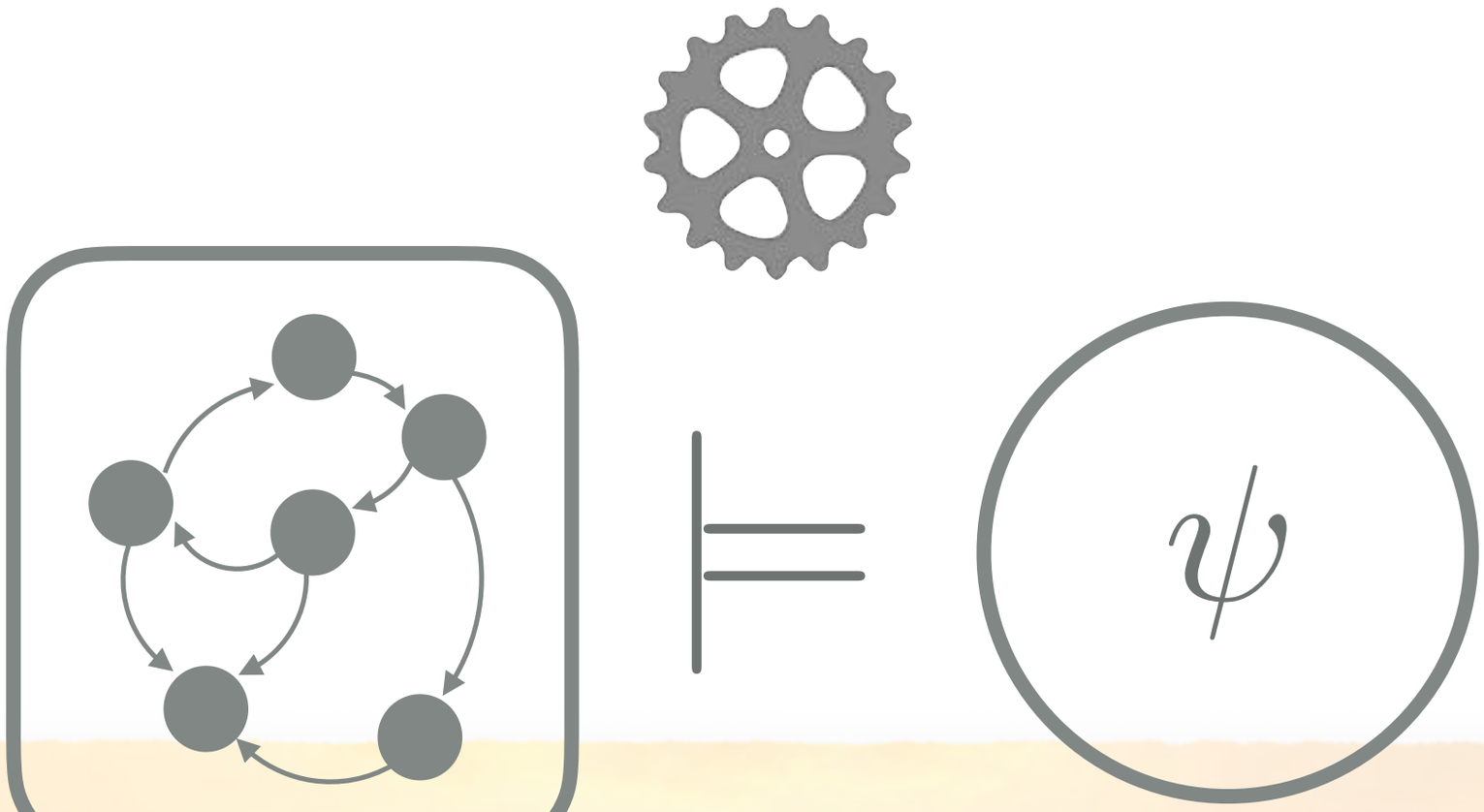
a b c a b d a b d d a c b d a d c d d

¿ ψ **...(#a≥# c)...** ?

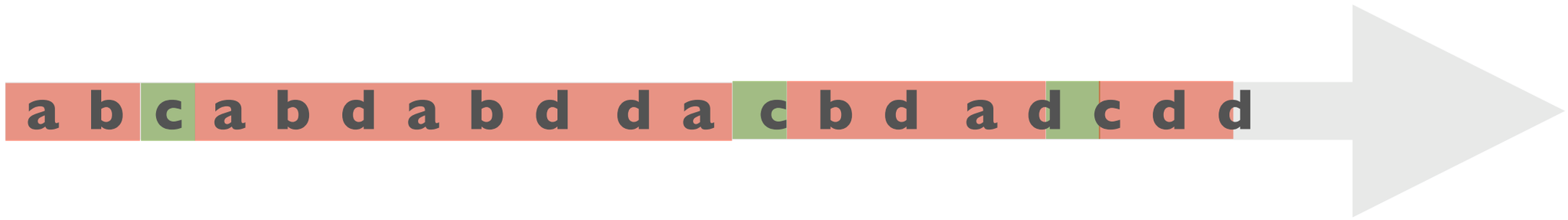


ψ **...(1a∧0c)∨(1a∧1c)∨(2a∧0c)∨(2a∧ 1c)∨(2a ...) ...**

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
```

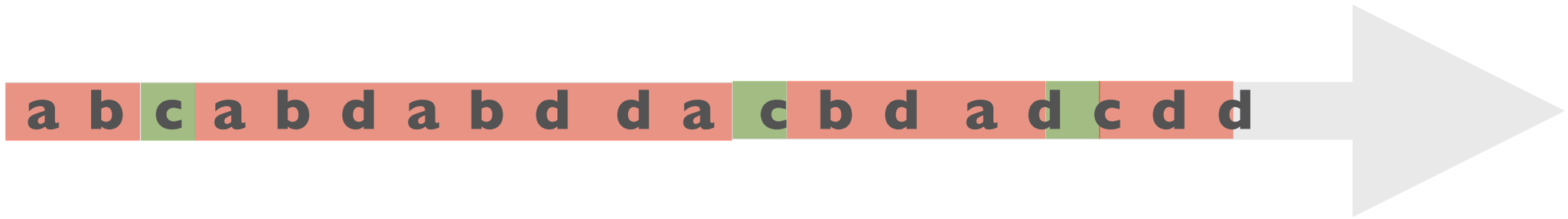
Conjunto de **incremento**

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

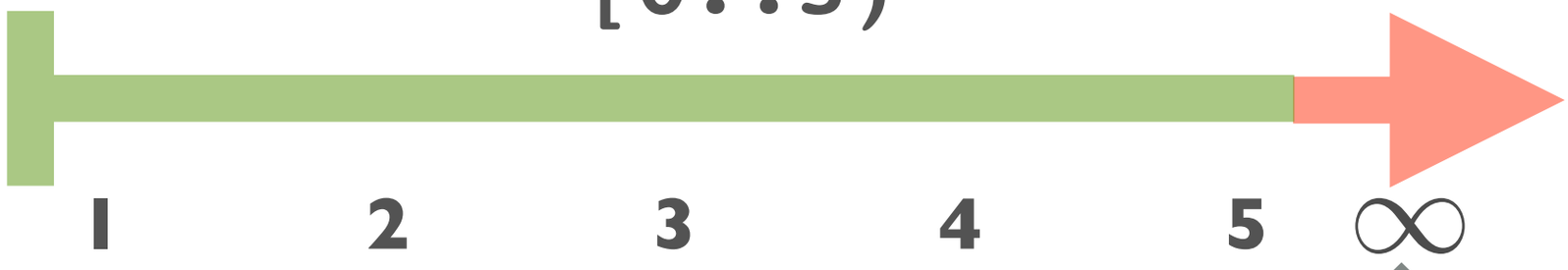
FLUENTES CONTADORES LÍMITES

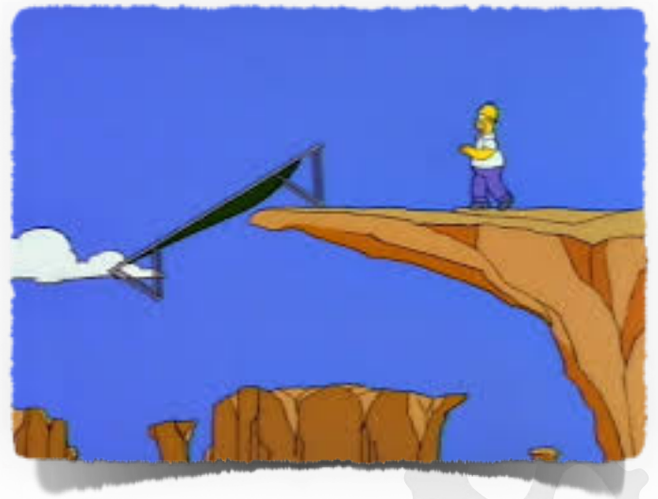
Dado el potencial infinito en los rangos de fluentes contadores (espacio infinito), para poder aplicar la técnica de Model Checking, los **limitamos**

Límite **estricto** **[0..5]**

Límite **NO estricto**

[0..5)



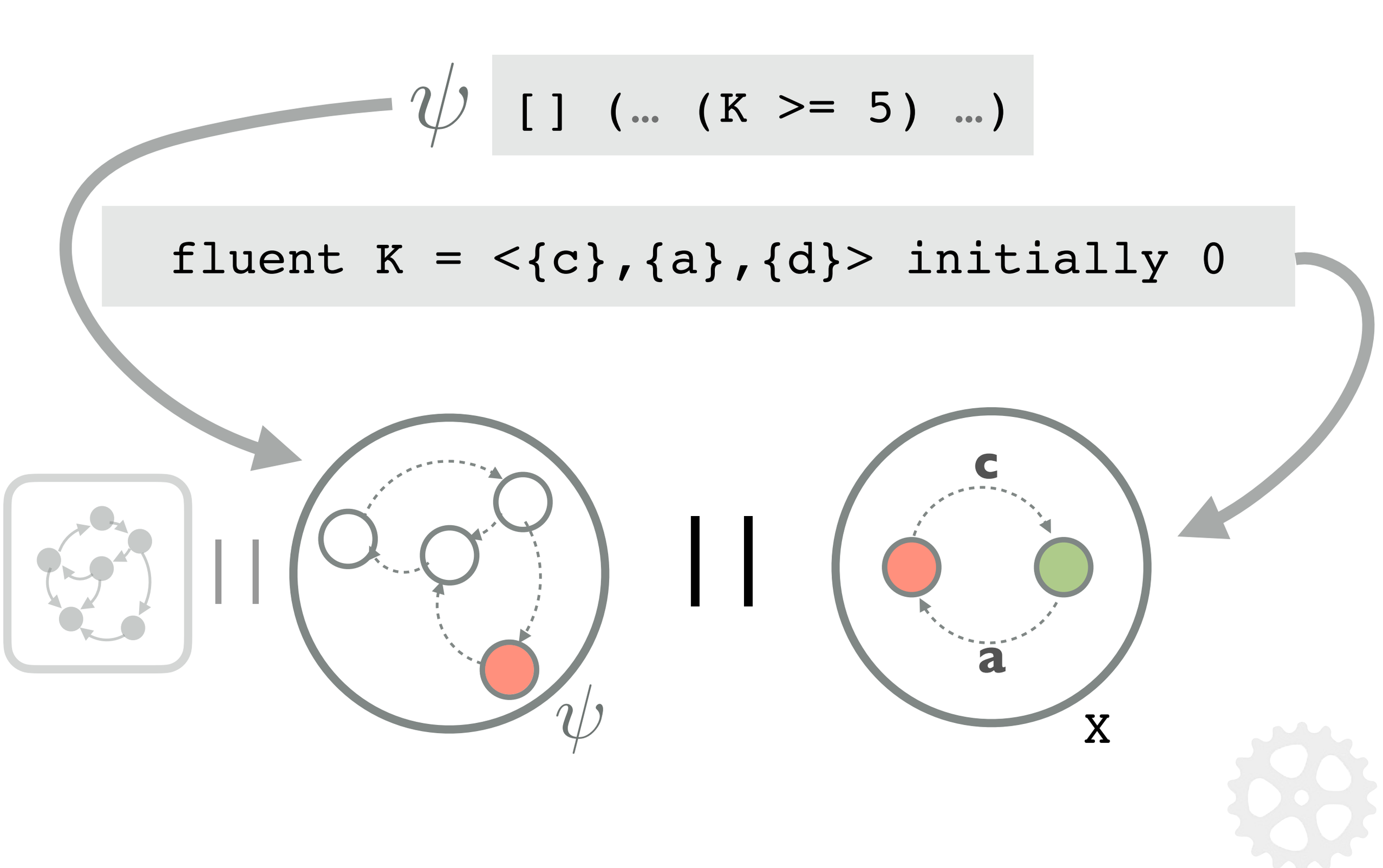


Si en algún momento del proceso de verificación alcanzamos **overflow** el resultado es

CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON FLUENTES CONTADORES**

ψ `[] (… (K >= 5) …)`

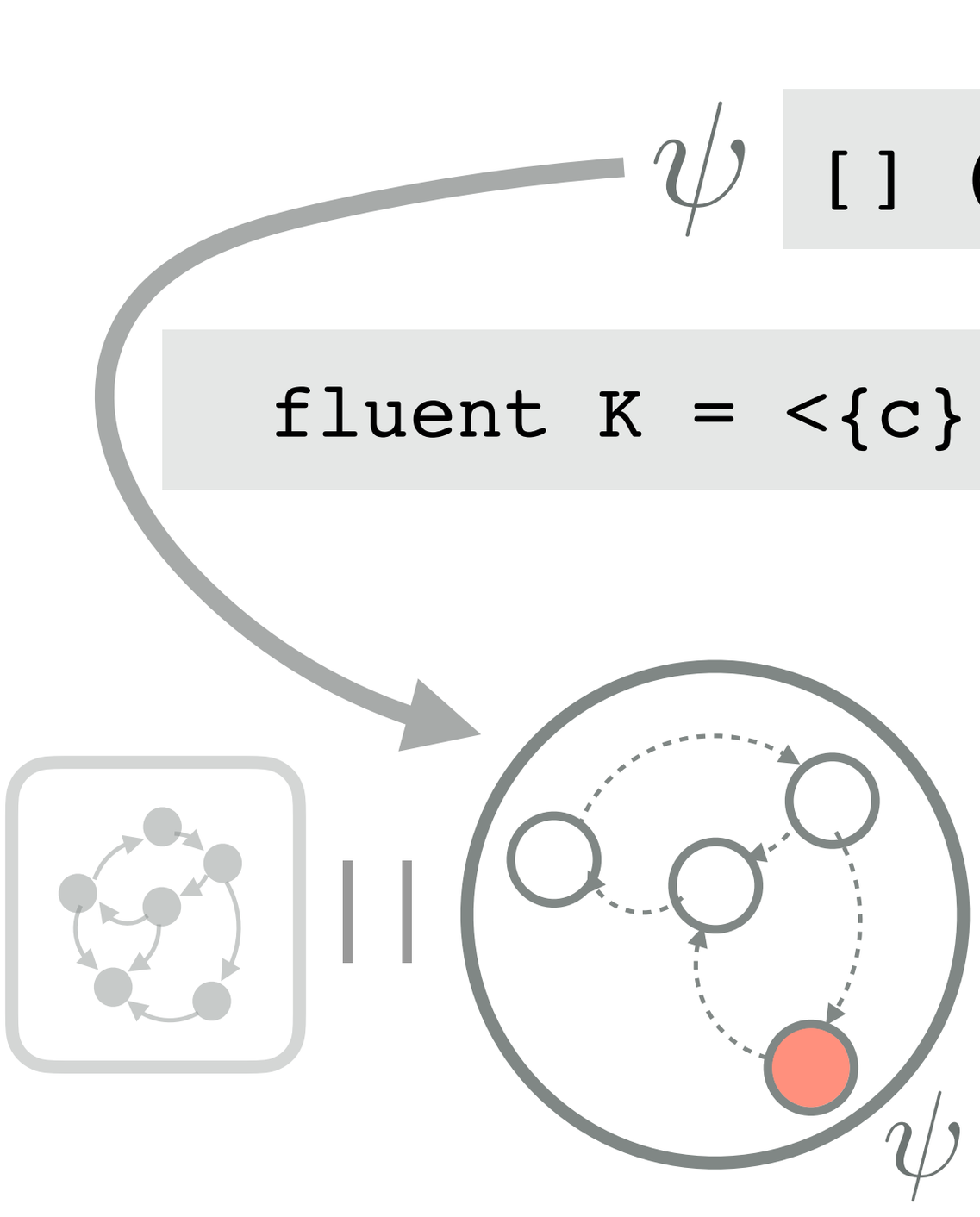
```
fluent K = <{c},{a},{d}> initially 0
```



CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON FLUENTES CONTADORES**

ψ `[] (… (K >= 5) …)`

```
fluent K = <{c},{a},{d}> initially 0
```

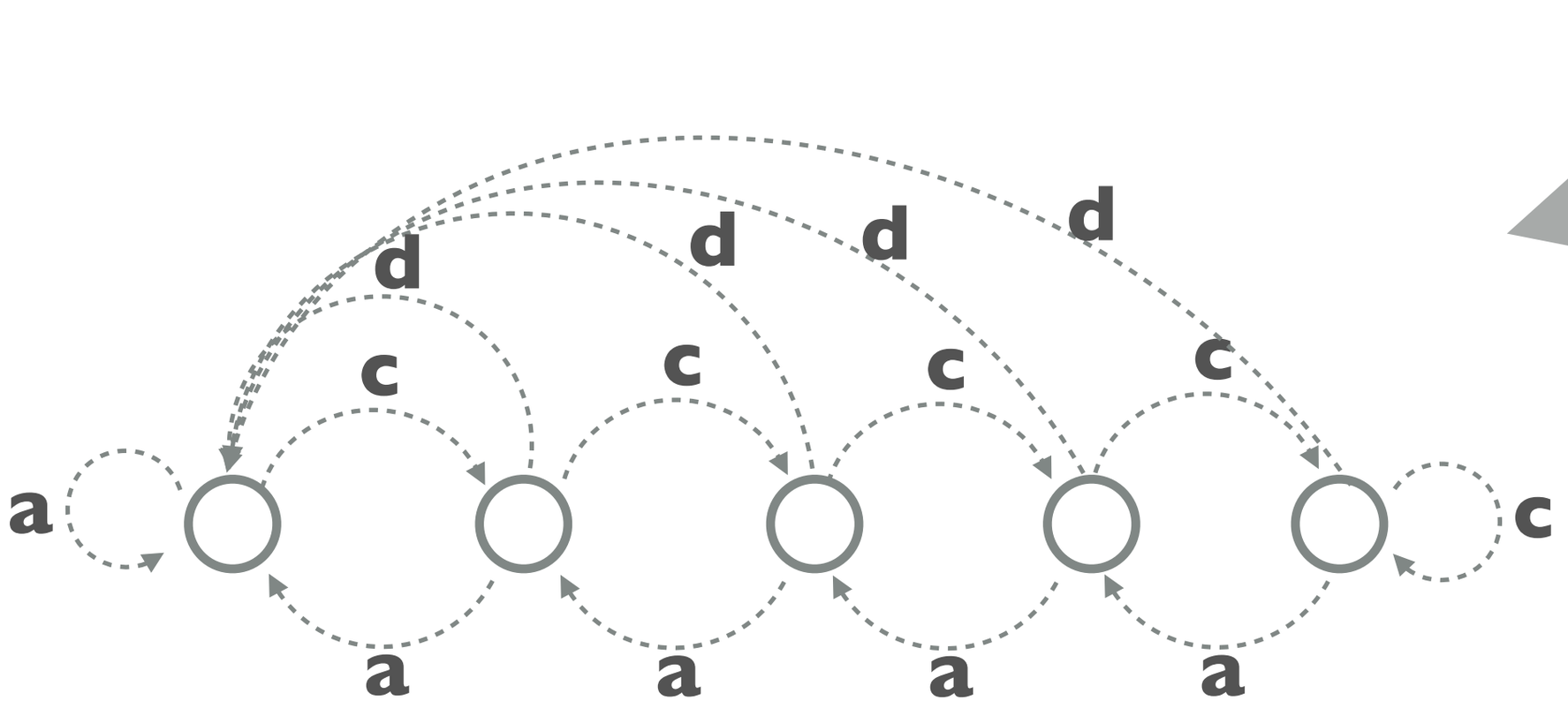


?

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

ψ `[] (… (K >= 3) …)`

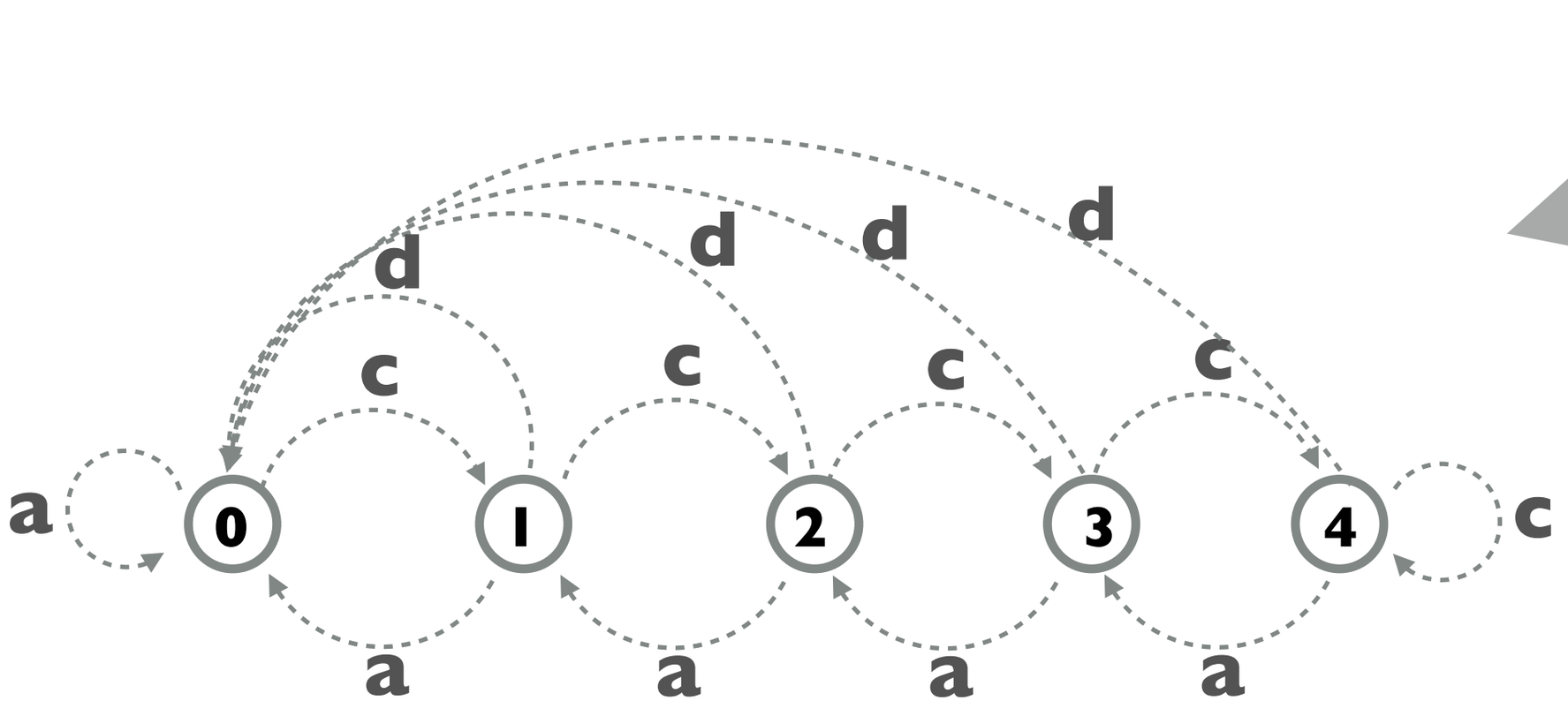
```
fluent K = <{c},{a},{d}> initially 0 apply [0..4]
```



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

ψ `[] (… (K >= 3) …)`

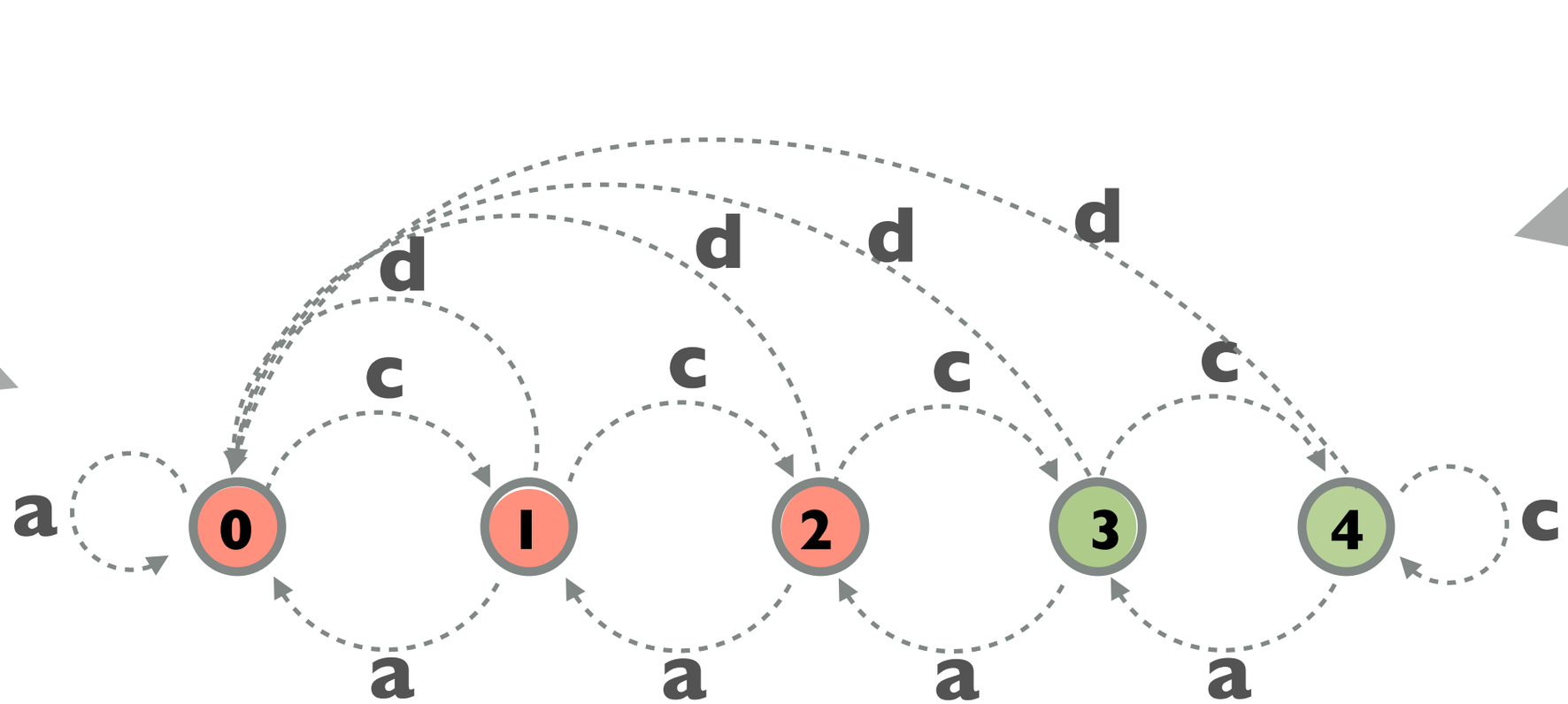
```
fluent K = <{c},{a},{d}> initially 0 apply [0..4]
```



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

ψ `[] (… (K >= 3) …)`

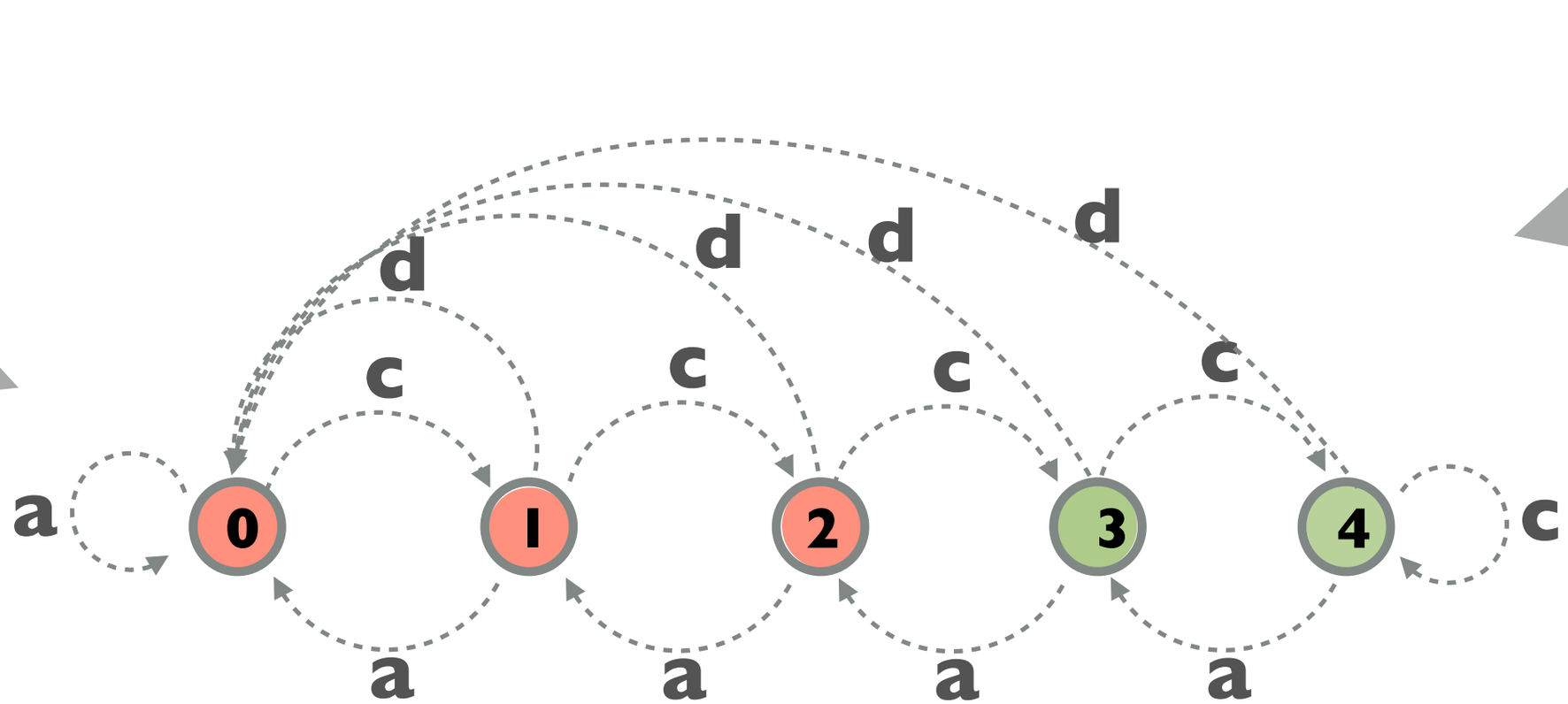
```
fluent K = <{c},{a},{d}> initially 0 apply [0..4]
```



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

ψ `[] (… (K >= 3) …)`

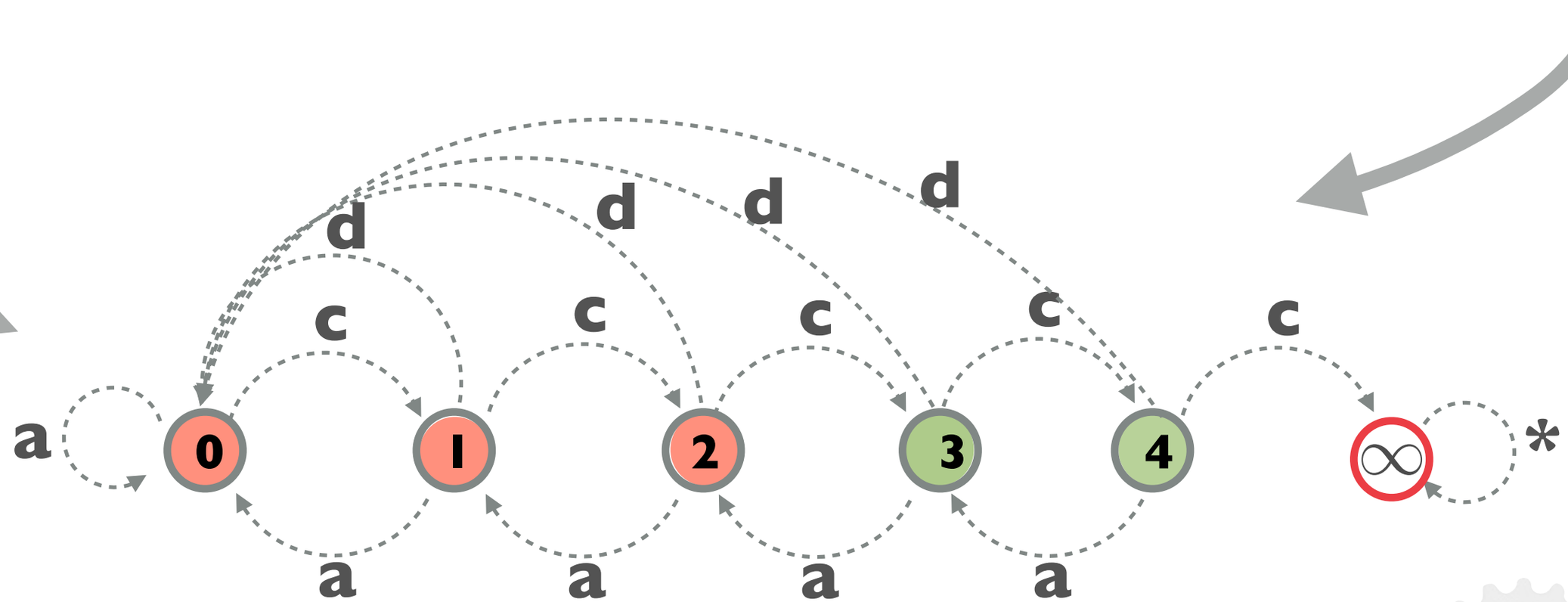
```
fluent K = <{c},{a},{d}> initially 0 apply [0..4)
```



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

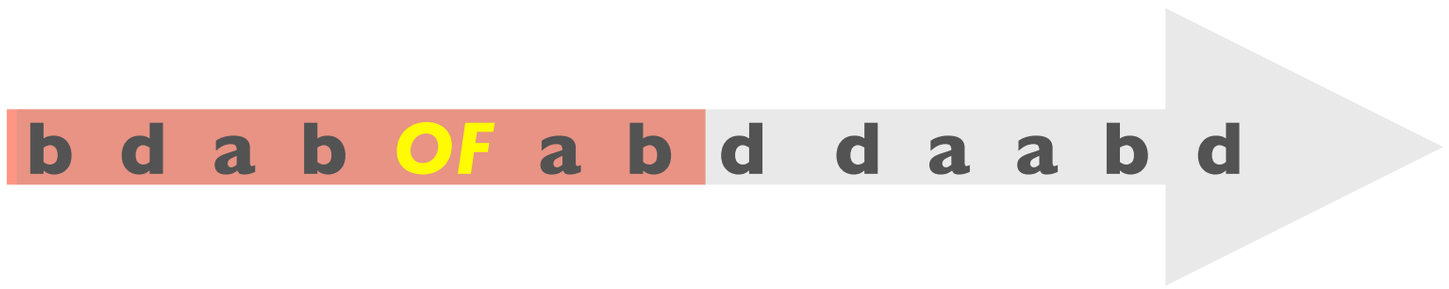
ψ `[] (… (K >= 3) …)`

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



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

Safety:

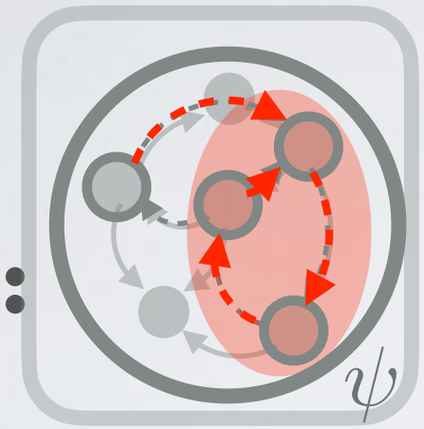


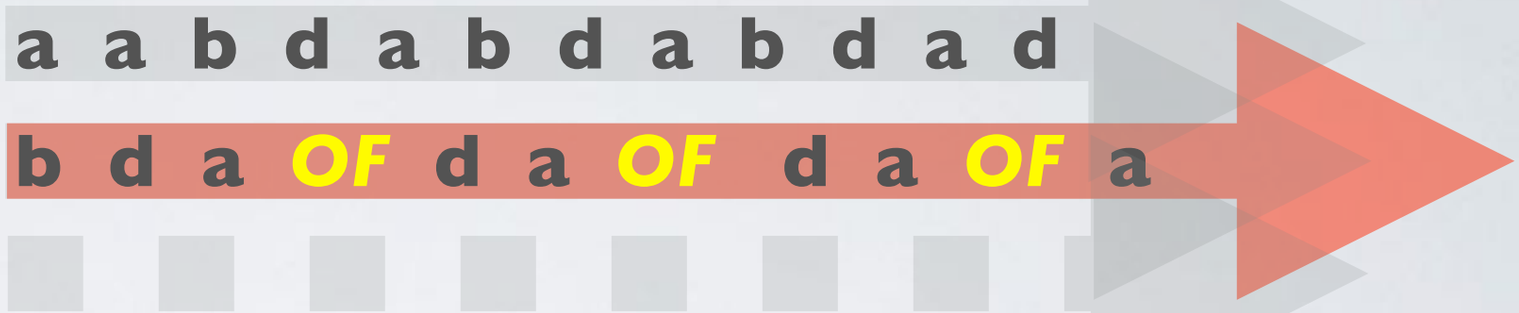
Si la traza encontrada presenta un evento de **overflow** no se tiene en cuenta.

- 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:



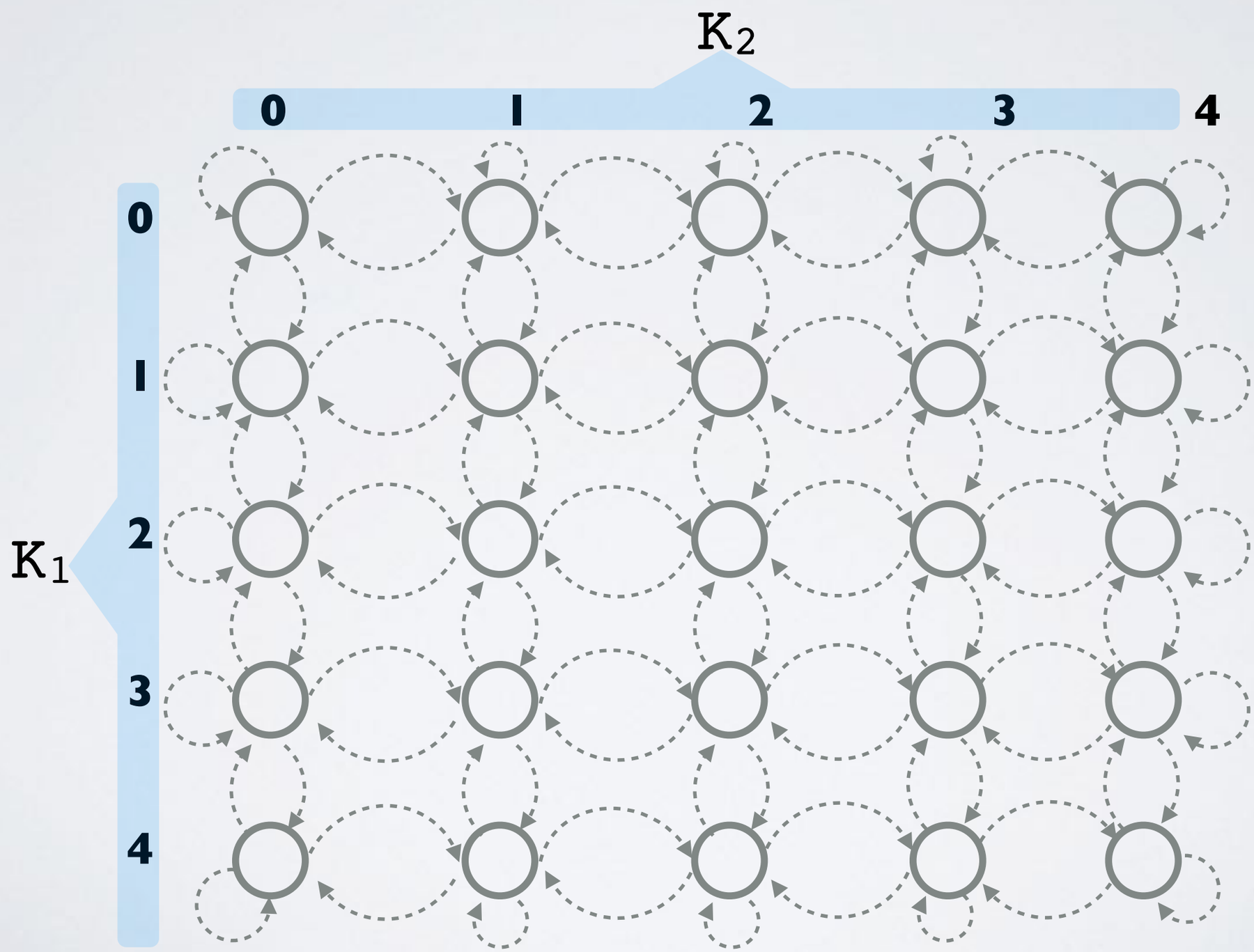


Si el componente conexo presenta un evento de **overflow** no se tiene en cuenta.

- 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

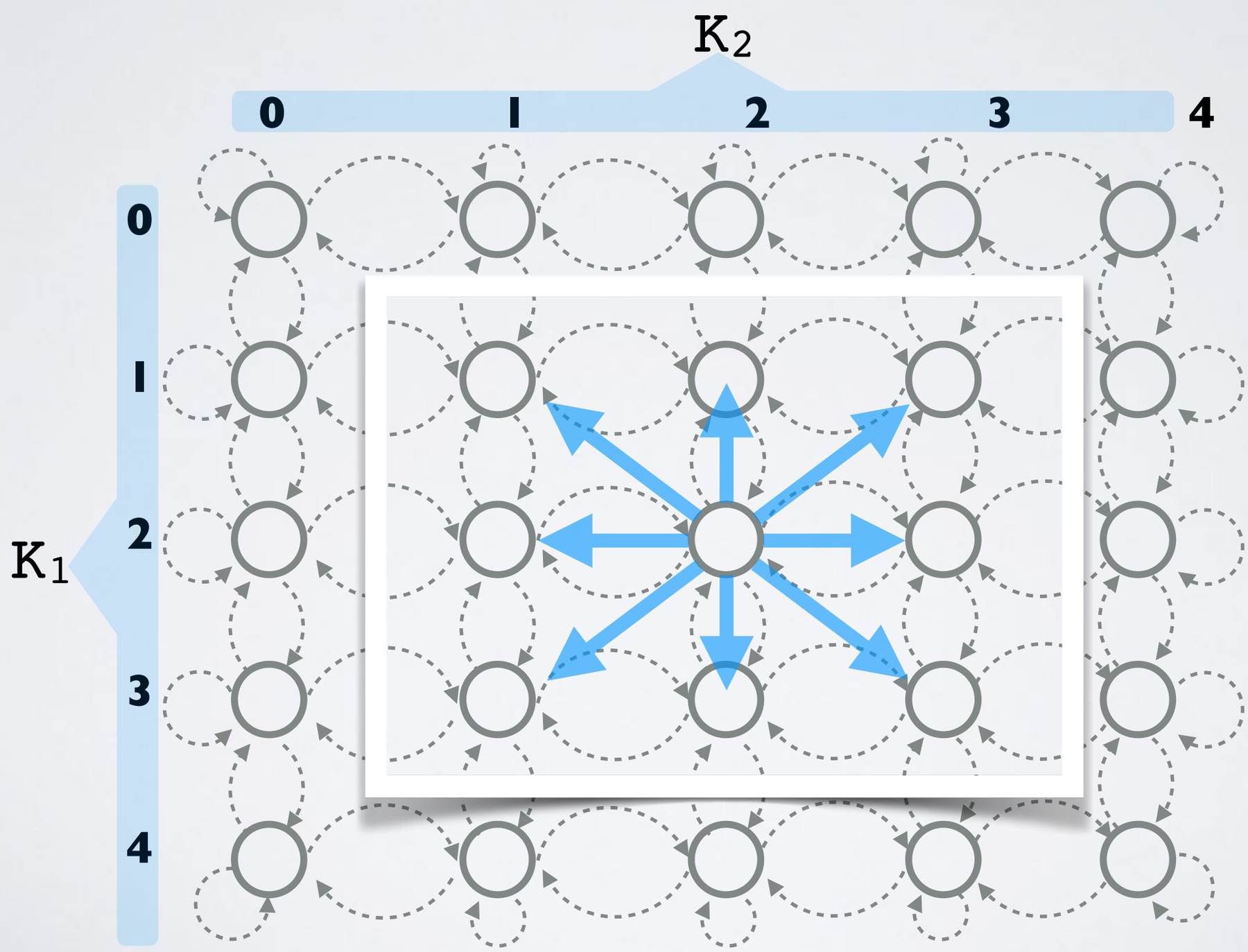
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON EXPRESIONES CONTADORAS COMPLEJAS**

ψ `[] (… (K1 >= K2) …)`



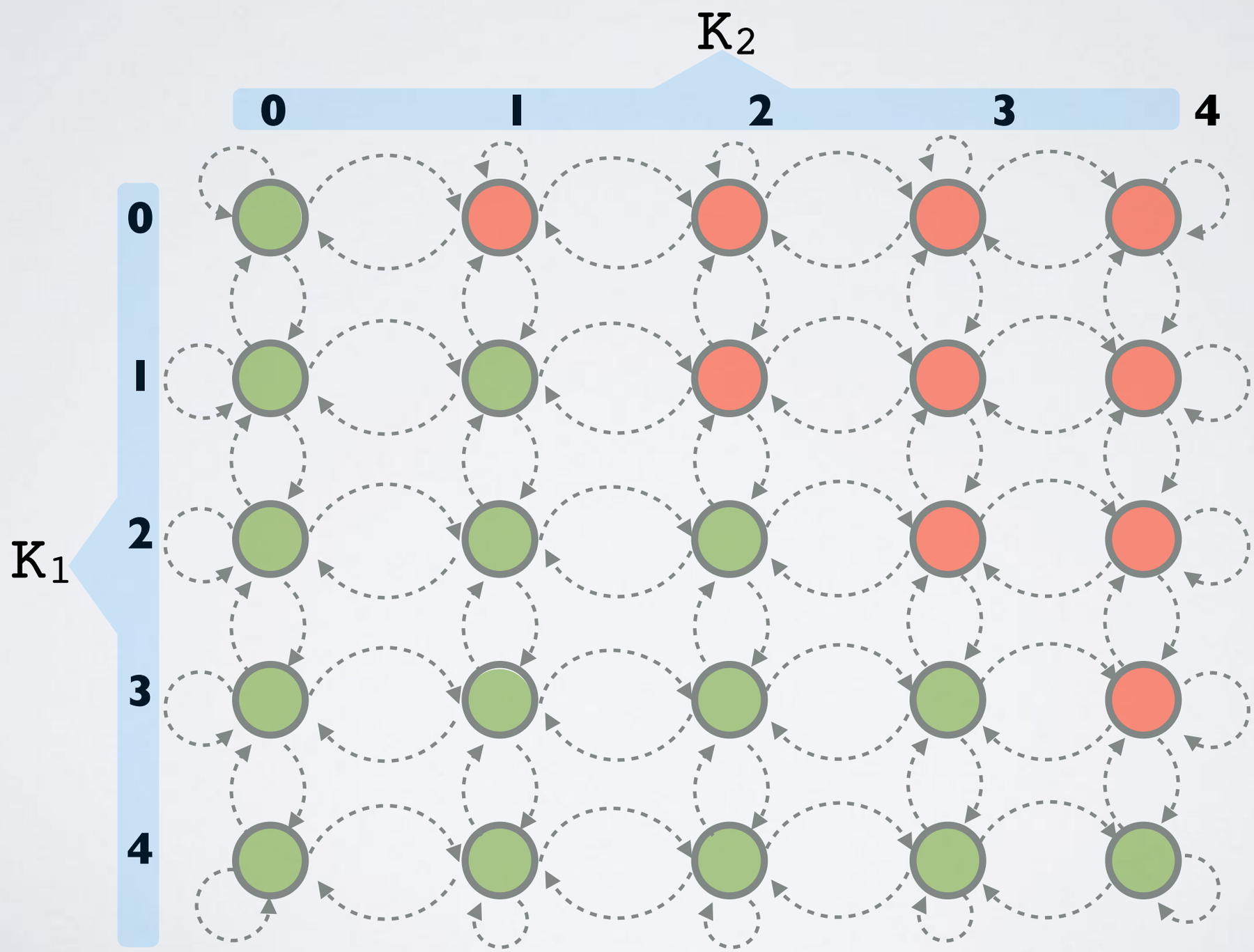
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON EXPRESIONES CONTADORAS COMPLEJAS**

ψ `[] (… (K1 >= K2) …)`



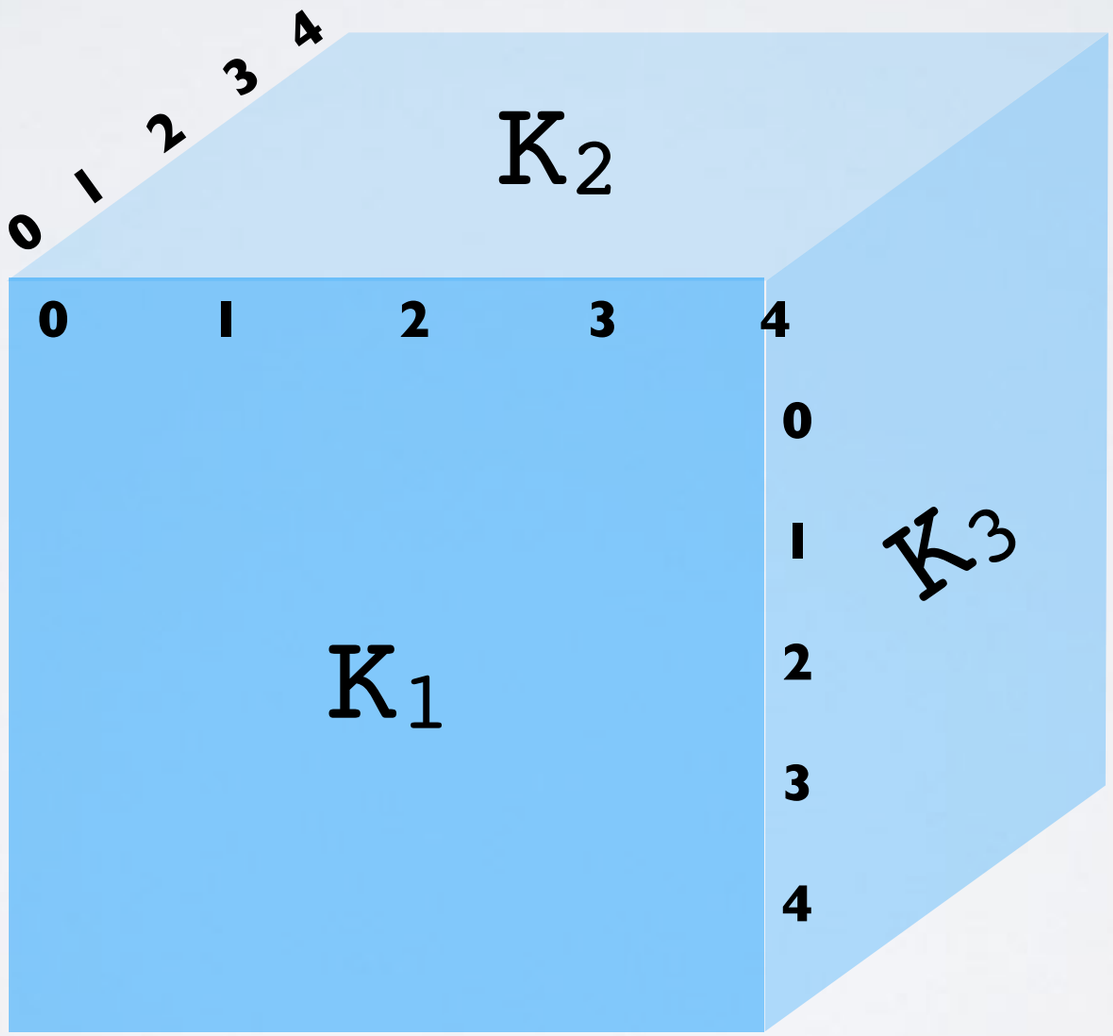
CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON EXPRESIONES CONTADORAS COMPLEJAS**

ψ `[] (… (K1 >= K2) …)`

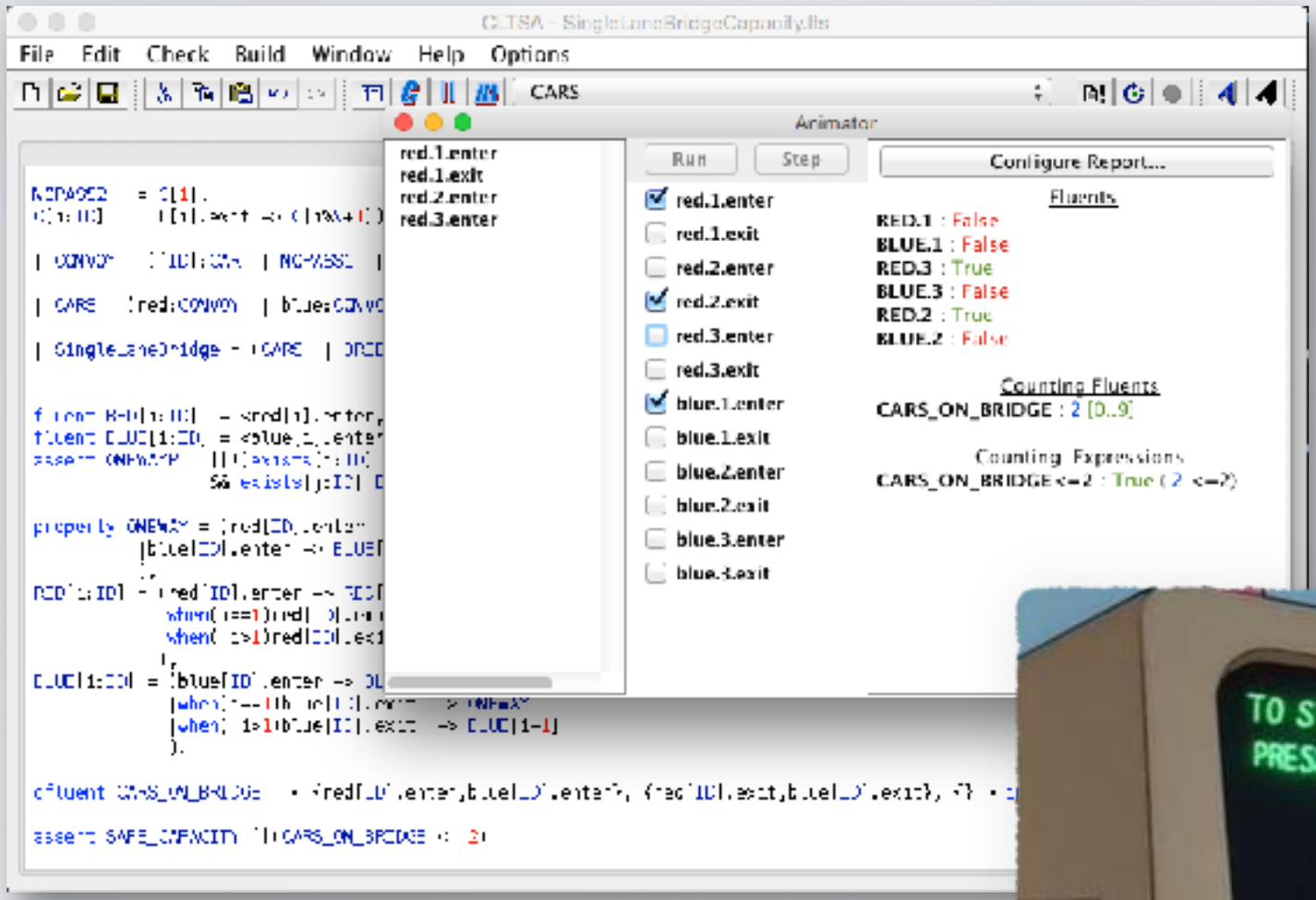


CÓMO VERIFICAR AUTOMÁTICAMENTE PROPIEDADES **CON EXPRESIONES CONTADORAS COMPLEJAS**

ψ `[] (… (K1 >= K2 + K3) …)`



COUNTING FLUENT LABELLED TRANSITION SYSTEM ANALYZER



WEB

http://countingfluents.weebly.com

countingfluents.weebly.com

CLTSA: LABELLED TRANSITION SYSTEM ANALYSER WITH COUNTING FLUENT SUPPORT

Home Model Checking Examples Download Publications About Contact

The CLTSA allows to verify system properties containing counting expressions. A system in CLTSA is modelled as a set of interacting finite state machines. CLTSA supports Finite State Process notation (FSP) for concise description of component behaviour. In order to use CLTSA to analyse a system modelled as FSP process, the counting fluents and counting expressions can be defined as follows:

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

Abrir "countingfluents.weebly.com" en una pestaña nueva one. And *res_set* is the reset