Finite State Machines

- **The system is characterised by *explicitly* depicting its states as well as the transitions from one state to another.**
- **One particular state is specified as the initial one**
- **States and transitions are in a finite number.**
- **Transitions are triggered by input events.**
- **Transitions generate outputs.**
- **FSMs are suitable for modeling control dominated reactive systems (react on inputs with specific outputs)**

خصائص آلة الحالة المحدودة(FSMs)

- "The system is characterised by *explicitly* depicting its states as well as the transitions from one state to another."
  - الفكرة الأساسية هنا هي الوضوح التام. نحن نحدد مسبقاً وبشكل صريح جميع "الأوضاع" أو "الحالات (states) " الممكنة التي يمكن أن يكون عليها نظامنا.
  - مثال بسيط :آلة بيع المشروبات. حالاتها يمكن أن تكون "خاملة(idle) "، "تنتظر المزيد من النقود (waiting for money)"، "جاهزة لاختيار المشروب (ready to select) "
  - بالإضافة إلى الحالات، نحدد بشكل صريح "الانتقالات (transitions) "، وهي القواعد التي تحكم كيفية الانتقال من حالة إلى أخرى. "إذا كنا في الحالة الخاملة وأدخلنا قطعة نقدية، ننتقل إلى حالة انتظار المزيد من النقود"
- "One particular state is specified as the initial one"يتم تحديد حالة معينة على أنها الحالة الأولية
  - يجب أن يكون هناك دائماً نقطة بداية واضحة. عندما نقوم بتشغيل النظام، فإنه يبدأ دائماً في هذه الحالة الأولية (initial state) المحددة. في مثال آلة البيع، الحالة الأولية هي "خاملة".
- "States and transitions are in a finite number."عدد الحالات والانتقالات محدود.
  - محدودة(Finite) : عدد الحالات وعدد الانتقالات الممكنة معروف ومحدود. لا يمكن للنظام أن يدخل في حالة غير متوقعة لم نقم بتعريفها مسبقاً. هذا ما يجعل هذا النموذج بسيطاً وقابلاً للتحليل.
- "Transitions are triggered by input events." يتم إطلاق الانتقالات بواسطة أحداث الإدخال.

- آلة الحالة هي نظام "تفاعلي (reactive) "إنها لا تفعل شيئاً من تلقاء نفسها، بل تنتظر. ما الذي تنتظره؟ تنتظر حدث إدخال (input event) .هذا الحدث يمكن أن يكون ضغطة زر، وصول بيانات من حساس، أو انتهاء مؤقت زمني.
- هذا الحدث هو الذي "يطلق" أو "يقدح (triggers) "الانتقال.

- "Transitions generate outputs." الانتقالات تولد مخرجات.
- عندما يتم إطلاق انتقال ما، فإنه عادة ما يقوم بشيئين :
  1. ينقل النظام إلى حالة جديدة (أو يبقيه في نفس الحالة).
  2. يولد مخرجاً (output)
- في مثال آلة البيع، عند إدخال نقود كافية (input event) والانتقال إلى حالة الاختيار، فإن اختيار مشروب (input event آخر) سيؤدي إلى انتقال يولد مخرجاً وهو "إسقاط علبة المشروب"

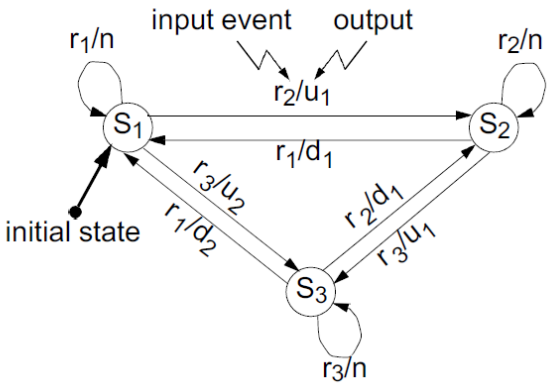
- "FSMs are suitable for modeling control dominated reactive systems (react on inputs with specific outputs)" آلات الحالة المحدودة مناسبة لنمذجة الأنظمة التفاعلية التي يهيمن عليها التحكم (تتفاعل مع المدخلات بمخرجات محددة)
- هذه النقطة تلخص مجال قوة FSMs . هي ليست مصممة للمهام التي تتطلب حسابات بيانات معقدة (مثل معالجة الصور). بدلاً من ذلك، هي تتألق في الأنظمة التي يهيمن عليها منطق التحكم(control logic)
- "النظام التفاعلي الذي يهيمن عليه التحكم "هو نظام وظيفته الرئيسية هي إدارة تدفق العمل والتفاعل مع الأحداث. مثل :
  - وحدة التحكم في إشارة المرور
  - لوحة التحكم في المايكروويف.
  - منطق اتخاذ القرار البسيط في روبوت (مثلاً: "إذا كان الحساس الأمامي يرى عائقاً، توقف ودر لليسار").

باختصار، آلة الحالة المحدودة هي طريقة منظمة وبسيطة لوصف سلوك نظام تفاعلي من خلال تحديد حالاته، نقطة بدايته، والانتقالات بين هذه الحالات التي تسببها مدخلات معينة وتنتج عنها مخرجات محددة.

FSM Example-1

Elevator controller

- **Input events: {r_1, r_2, r_3}**
  - **r_i: request from floor i.**
- **Outputs: {d_2, d_1, n, u_1, u_2}**
  - **d_i: go down *i* floors**
  - **u_i: go up *i* floors**
  - ***n*: stay idle**
- **States: {S_1, S_2, S_3}**
  - **S_i: elevator is at floor *i*.**



مثال 1 على آلة الحالة المحدودة

بعد أن استعرضنا المفاهيم النظرية لآلة الحالة المحدودة، سوف نطبقها الآن على مثال عملي وملموس لنرى كيف تعمل على أرض الواقع.
هذا المثال هو **وحدة تحكم لمصعد (Elevator controller)** في مبنى مكون من ثلاثة طوابق.

هذا مثال ممتاز لأنه يوضح كيف يمكن استخدام FSM لإدارة منطق التحكم.

لنبني نموذج FSM لهذا المصعد، علينا أولاً أن نحدد مكوناته الأساسية:

- **الحالات:{S1, S2, S3}**
  - كل حالة تمثل بوضوح الموقع الحالي للمصعد.
  - S1: المصعد في الطابق الأول.
  - S2:المصعد في الطابق الثاني.
  - S3:المصعد في الطابق الثالث.
- **أحداث الإدخال:{r1, r2, r3}**
  - هذه هي الأحداث التي يستجيب لها النظام. في هذه الحالة، هي طلبات استدعاء المصعد من الطوابق المختلفة.
  - r1: طلب من الطابق الأول (شخص في الطابق الأول ضغط على زر استدعاء المصعد).
  - r2: طلب من الطابق الثاني.

- r3 :طلب من الطابق الثالث.

- **المخرجات:{d2, d1, n, u1, u2}**

  - هذه هي الأوامر التي تصدرها وحدة التحكم إلى محرك المصعد.
  - d_i :اذهب إلى الأسفل i من الطوابق (مثلاً d1 تعني انزل طابقاً واحداً).
  - u_i :اذهب إلى الأعلى i من الطوابق (مثلاً u2 تعني اصعد طابقين).
  - n :لا تفعل شيئاً / ابق في مكانك(stay idle)

تحليل المخطط البياني:

الآن سوف نتتبع سلوك المصعد من خلال المخطط البياني على اليمين، الذي يربط كل هذه المكونات معاً:

- **نقطة البداية :**السهم الذي لا يأتي من أي حالة ويشير إلى S1 يخبرنا أن S1 (الطابق الأول) هي **الحالة الأولية (initial state)** . عند تشغيل النظام، يكون المصعد دائماً في الطابق الأول.

- **سيناريو1: نحن في الطابق الأول، وشخص يطلب المصعد من الطابق الثاني.**

1. الحالة الحالية هي S1
2. يحدث الإدخال r2 طلب من الطابق 2.
3. نبحث عن سهم يخرج من S1 ومكتوب عليه r2 كمدخل.
4. نجد السهم الذي ينتقل من S1 إلى S2 .التسمية على السهم هي r2 / u1
5. **التفسير :**عندما يكون المصعد في الطابق الأول (S1) ويأتيه طلب من الطابق الثاني(r2) ، فإنه يُصدر الأمر u1 (اصعد طابقاً واحداً)، وينتقل إلى حالة جديدة هي S2 (المصعد الآن في الطابق الثاني).

- **سيناريو2: نحن في الطابق الأول، وشخص في نفس الطابق يضغط الزر.**

1. الحالة الحالية هي S1
2. يحدث الإدخال r1
3. نبحث عن سهم يخرج من S1 ومكتوب عليه r1 كمدخل.
4. نجد سهماً يلتف ويعود إلى S1 التسمية عليه هي r1 / n
5. **التفسير :**إذا كان المصعد موجوداً بالفعل في الطابق المطلوب، فلا داعي للحركة. النظام يستقبل الإدخال r1 ولكنه يصدر الأمر n(ابق في مكانك)، وتبقى الحالة S1 كما هي.

- **سيناريو3: نحن في الطابق الثالث، وشخص يطلب المصعد من الطابق الأول.**

1. الحالة الحالية هي S3

2. يحدث الإدخال r1
3. نبحث عن سهم يخرج من S3 و يذهب إلى S1
4. نجد أن التسمية عليه هي r1 / d2
5. **التفسير** :عندما يكون المصعد في الطابق الثالث (S3) ويأتيه طلب من الطابق الأول (r1) ، فإنه يصدر الأمر d2 (انزل طابقين)، وينتقل إلى الحالة S1 (المصعد الآن في الطابق الأول).

هذا المثال يجسد بشكل مثالي كل الخصائص التي ذكرناها سابقاً: حالات وانتقالات محدودة وواضحة، حالة بداية محددة، وانتقالات يتم إطلاقها بواسطة أحداث إدخال لتوليد مخرجات محددة. إنه مثال نموذجي لنظام تفاعلي يهيمن عليه منطق التحكم.

Extended Finite State Machines

- **Variables can be associated to the FSM.**
  - **Changes to variables specified as actions associated to transitions.**
  - **Extended FSMs are suitable for systems which are both control and computation intensive.**

- ***Guards* (expressed as conditions) may be specified for transitions: The transition is performed when the associated event(s) occur and if the associated guard is true**

لقد رأينا في المثال السابق كيف أن آلة الحالة المحدودة (FSM) رائعة لنمذجة منطق التحكم البسيط. لكن ماذا لو أصبح نظامنا أكثر تعقيداً؟ ماذا لو كان على المصعد أن يحسب عدد الرحلات التي قام بها؟ أو أن يتوقف عن الحركة إذا كان وزنه زائداً؟

لنمذجة مثل هذه السلوكيات، قد نحتاج إلى عدد هائل من الحالات (حالة لكل عدد ممكن من الرحلات مثلاً)، وهذا غير عملي. هنا يأتي دور **"آلات الحالة المحدودة الموسعة(Extended Finite State Machines - EFSMs) "**، التي تضيف طبقتين من القوة على النموذج الأساسي.

آلات الحالة المحدودة الموسعة(Extended Finite State Machines)

الفكرة بسيطة: نأخذ آلة الحالة المحدودة ونضيف إليها "ذاكرة" و"شروطاً ذكية"

- **"Variables can be associated to the FSM."يمكن ربط المتغيرات بآلة الحالة المحدودة.**

- هذه هي الإضافة الأولى والكبرى :**المتغيرات (Variables)** . لم يعد النظام يعتمد على حالته الحالية فقط، بل أصبح لديه ذاكرة يمكنه تخزين قيم فيها.
- يمكن أن تكون هذه المتغيرات أي شيء: عدّاد لعدد الركاب، متغير لتخزين الطابق المطلوب، أو متغير منطقي (true/false) لتحديد ما إذا كان المصعد في وضع الصيانة.
- **"Changes to variables specified as actions associated to transitions."يتم تحديد التغييرات على المتغيرات كإجراءات مرتبطة بالانتقالات**
  - الآن، عندما يحدث انتقال، فإنه بالإضافة إلى توليد مخرج، يمكنه أيضاً القيام بـ**"إجراء(action) "**، وهذا الإجراء عادة ما يكون تعديل قيمة أحد المتغيرات.
  - **مثال :**عندما ينتقل المصعد من طابق لآخر، يمكن أن يقوم الإجراء trip_counter := trip_counter + 1 بتحديث عداد الرحلات.
- **"Extended FSMs are suitable for systems which are both control and computation intensive." الآلات الموسعة مناسبة للأنظمة التي تكون كثيفة التحكم والحسابات معاً .**
  - بفضل المتغيرات والإجراءات، أصبح بإمكاننا نمذجة أنظمة لا يقتصر عملها على التحكم فقط، بل يشمل أيضاً بعض **الحسابات ومعالجة البيانات**، مما يجعلها أقرب بكثير للأنظمة الواقعية.

- **"*Guards* (expressed as conditions) may be specified for transitions: The transition is performed when the associated event(s) occur and if the associated guard is true." (يمكن تحديد الحراس (المعبر عنها كشروط للانتقالات: يتم تنفيذ الانتقال عند وقوع الحدث (الأحداث) المرتبطة به وإذا كان الحارس المرتبط به صحيحاً(true)**
  - هذه هي الإضافة الثانية القوية :**الحراس (Guards)** . الحارس هو شرط منطقي إضافي يوضع على الانتقال.
  - لكي يحدث انتقال معين، لم يعد كافياً أن يقع حدث الإدخال فقط. الآن يجب أن يتحقق شرطان معاً :

1. أن يقع **حدث الإدخال** المرتبط بالانتقال (مثلاً، ضغط زر الطلب r3)
2. **وأن يكون الشرط الحارس (guard condition)** صحيحاً (true) . هذا الشرط عادة ما يتحقق من قيم المتغيرات.

  - **مثال للمصعد :**
    - لدينا انتقال من S1إلى S2 بسبب الحدث r2
    - يمكننا أن نضع عليه حارساً: [weight < max_weight]
    - **ماذا يعني هذا؟** لن يتحرك المصعد من الطابق الأول إلى الثاني حتى لو ضغط أحدهم الزر(r2) ، إلا إذا كان الشرط [weight < max_weight]صحيحاً. إذا كان المصعد محملاً بوزن زائد (الحارس قيمتهfalse)، فلن يتم الانتقال، وسيبقى المصعد في مكانه، ربما مع إصدار صوت إنذار (كمخرج آخر).

الخلاصة:

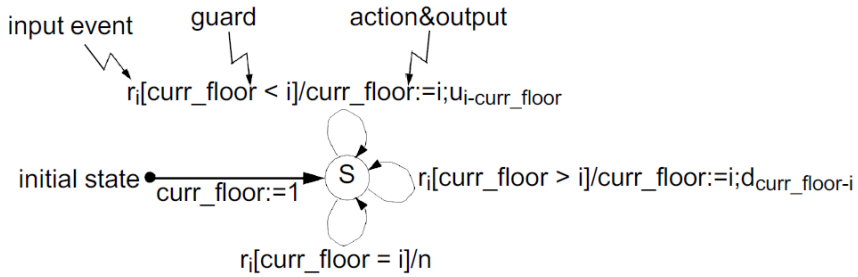
آلة الحالة المحدودة الموسعة (EFSM) هي ببساطة : **EFSM = FSM الأساسية + متغيرات (ذاكرة) + حراس (شروط ذكية)**

هذا النموذج أقوى وأكثر كفاءة بكثير من FSM العادية، لأنه يسمح لنا بفصل منطق التحكم (الموجود في الحالات والانتقالات) عن منطق البيانات والحسابات (الموجود في المتغيرات والحراس). هذا يساعدنا بشكل كبير في تجنب "مشكلة انفجار الحالات" التي سنتحدث عنها لاحقاً، حيث لم نعد بحاجة لإنشاء حالة جديدة لكل قيمة ممكنة لمتغير ما.

FSM Example-1 Modified

Elevator controller with extended FSM

- **We associate to the FSM a variable storing the current floor.**



- **You might wonder:** ***Do we really have one single state of the system? Of course not!***

  The global system state is now encoded in the FSM state and the value of the associated variable.

هذه الشريحة تعرض لنا نفس مثال المصعد، ولكن هذه المرة باستخدام **آلة الحالة المحدودة الممتدة(EFSM)**

- **"We associate to the FSM a variable storing the current floor." نربط بآلة الحالة متغيراً يخزن الطابق الحالي.**
  - هذا هو التغيير الجذري الأول. بدلاً من أن يكون لدينا ثلاث حالات منفصلة (S1, S2, S3) لتمثيل موقع المصعد، سنستخدم الآن **حالة تحكم واحدة فقط** سنسميها S، و **متغير (variable)** اسمه curr_floor لتخزين رقم الطابق الذي يوجد فيه المصعد حالياً.

نلاحظ أن المخطط الآن يحتوي على دائرة واحدة فقط للحالة S، وكل الانتقالات تعود إليها. هذا ممكن بسبب قوة المتغيرات والحراس.

- **الحالة الأولية (Initial State):** عند بدء تشغيل النظام، فإنه يدخل الحالة S ويقوم فوراً بتنفيذ **إجراء (action)** وهو : curr_floor := 1. هذا يضمن أن المصعد يبدأ دائماً من الطابق الأول، تماماً كما في المثال السابق.
- **الانتقالات (Transitions):** نلاحظ الصيغة الجديدة للانتقالات :حدث_الإدخال [الحارس] / الإجراء; المخرج.
  - **سيناريو 1: طلب المصعد إلى طابق أعلى.**
    - لنتتبع الانتقال العلوي: ri [curr_floor < i] / curr_floor := i; ui-curr_floor

- **شرح :** لنفترض أن المصعد في الطابق الأول (أي أن المتغير curr_floor = 1)، ويأتي طلب من الطابق الثالث (أي أن i = 3 والحدث هو r3).

1. **يتم فحص الحارس (Guard) :** هل الشرط [curr_floor < i]صحيح؟ هل 3 > 1؟ نعم، الشرط صحيح. بما أن الحدث وقع والحارس صحيح، يتم تنفيذ الانتقال.
2. **يتم تنفيذ الإجراء(Action) :** curr_floor := i ، أي curr_floor := 3. الآن، "يعرف" النظام أن وجهته الجديدة هي الطابق الثالث، فيقوم بتحديث قيمة المتغير.
3. **يتم توليد المخرج(Output):** ui-curr_floor ، أي u(3-1) وهو u2 . هذا هو الأمر لمحرك المصعد: "اصعد طابقين"

- **سيناريو 2: طلب المصعد إلى طابق أدنى.**
  - الانتقال السفلي... / ri [curr_floor > i] :
  - **شرح :**بالمثل، إذا كان المصعد في الطابق الثالث (curr_floor = 3) وجاء طلب من الطابق الأول(i = 1) ، فإن الحارس [1 > 3]يكون صحيحاً. يتم تحديث المتغير curr_floor := 1، ويتم إصدار الأمر d(3-1) أي d2(انزل طابقين).
- **سيناريو 3: طلب المصعد من نفس الطابق.**
  - الانتقال الأوسط / ri [curr_floor = i] :
  - **شرح :**إذا جاء طلب من الطابق الذي يوجد فيه المصعد حالياً (curr_floor = i) ، فإن الحارس [curr_floor = i] يكون صحيحاً. لا يوجد أي إجراء لتغيير المتغير، والمخرج هو n (ابق في مكانك).

- **"You might wonder: Do we really have one single state of the system? Of course not!"**
  - هذا سؤال جوهري. قد يبدو أننا بسطنا النموذج من ثلاث حالات إلى حالة واحدة فقط. لكن في الحقيقة، التعقيد لم يختف، بل **تمت إعادة تنظيمه**.
  - **"The global system state is now encoded in the FSM state *and* the value of the associated variable." حالة النظام الكلية الآن مشفرة في حالة الآلة FSM وقيمة المتغير المرتبط بها .**
    - "الحالة الكلية" للنظام لم تعد فقط حالة التحكم(S) ، بل هي **مزيج من حالة التحكم وقيمة جميع المتغيرات**.
    - إذاً، الحالات "الفعلية" لنظامنا هي :
      - (S, curr_floor = 1) ← هذه تعادل حالتنا القديمة S1
      - (S, curr_floor = 2) ← هذه تعادل حالتنا القديمة S2
      - (S, curr_floor = 3) ← هذه تعادل حالتنا القديمة S3

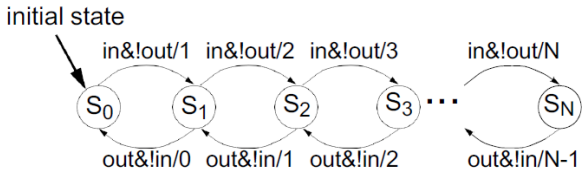
لماذا هذا أفضل؟

"إذا كانت الحالات الفعلية هي نفسها، فلماذا كل هذا العناء؟" لأن هذه الطريقة **أكثر قابلية للتوسع والنظافة في التصميم**. لنتخيل مصعداً في مبنى من 100 طابق. هل من الأفضل أن نرسم 100 حالة ومئات الانتقالات بينها؟ أم من الأفضل أن يكون لدينا حالة تحكم واحدة ومتغير واحد يمكنه أن يأخذ قيماً من 1 إلى 100؟ إن مبدأ الآلة الموسعة (EFSM) أفضل بكثير لإدارة التعقيد وفصل منطق التحكم عن البيانات، وهو ما يمهد لنا الطريق لفهم "مشكلة انفجار الحالات"

FSM Example-2

Parking counter

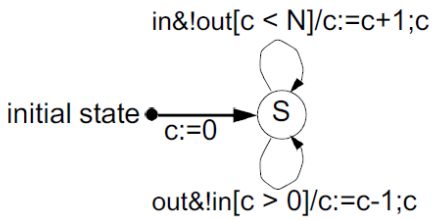
- **Input events: {in, out}**
  - ***in*: car enters;**
  - ***out*: car leaves.**
- **Outputs: {1, 2, 3, ... N}**
  - ***i: display value i***
- **States: {S_0, S_1, S_2, ... S_N}**
  - **S_i: *i* cars in the parking.**



FSM Example-2

Parking counter with extended FSM

- **We associate to the FSM a variable *c* storing the number of cars.**



بعد أن رأينا كيف حسّنت الآلة الموسعة (EFSM) نموذج المصعد، لنرى مثالاً آخر يوضح هذا الفارق بشكل أكثر وضوحاً.

هاتان الشريحتان تعرضان نفس النظام، وهو **عدّاد لساحة انتظار سيارات(Parking counter)** ، لكنهما يمثلانه بطريقتين مختلفتين تماماً.

الشريحة الأولى: النموذج باستخدام FSM التقليدية

هنا نصف النظام باستخدام آلة الحالة المحدودة الكلاسيكية.

- **الحالات: $\{S_0, S_1, S_2, \dots S_n\}$**
  - كل حالة تمثل عدد السيارات الموجودة في الساحة بدقة. S_0 تعني أن الساحة فارغة، S_1 تعني وجود سيارة واحدة، وهكذا حتى S_n التي تمثل الساحة ممتلئة تماماً.
  - نحتاج إلى N+1 حالة لتمثيل عداد يتسع لـ N من السيارات.
- **أحداث الإدخال{in, out} :**
  - in :سيارة تدخل الساحة.
  - out :سيارة تغادر الساحة.
  - ملاحظة على الترميز in&!out :تعني أن حدث الدخول in وقع، وحدث الخروج out لم يقع في نفس اللحظة.
- **المخرجات{1, 2, 3, ... N} :**
  - المخرج هو ببساطة عرض العدد الحالي للسيارات.

تحليل المخطط(FSM) :

- نبدأ من الحالة الأولية S_0 (الساحة فارغة).
- عندما تدخل سيارة (حدث in)، ننتقل من S_0 إلى S_1 والمخرج هو 1
- عندما تدخل سيارة أخرى، ننتقل من S_1 إلى S_2 والمخرج هو 2
- وهكذا، كل سيارة تدخل تنقلنا إلى الحالة التالية في السلسلة.
- وعندما تغادر سيارة حدث (out)، ننتقل إلى الحالة السابقة. مثلاً، من S_2 إلى S_1 والمخرج هو 1

المشكلة الواضحة هنا :إذا كانت سعة ساحة الانتظار كبيرة، مثلاً N = 500 سيارة، فسنحتاج إلى **501 حالة**. مع حجم هذا المخطط الضخم سيكون من المستحيل رسمه أو التعامل معه. هذا يقودنا مباشرة إلى النموذج الموسع.

الشريحة الثانية: نفس النموذج باستخدامEFSM

- بدلاً من N+1 حالة، لدينا الآن **حالة تحكم واحدة فقط** (S)
- ولدينا **متغير (variable)** اسمه c لتخزين عدد السيارات.

تحليل المخطط (EFSM) :

- **الحالة الأولية :**ندخل الحالة S ونقوم فوراً بتنفيذ إجراء c:=0، أي نقوم بتصفير العدّاد.
- **الانتقالات (الحلقات على الحالة S):**
  - **دخول سيارة (الحلقة العلوية) :** in&!out [c < N] / c:=c+1; c
    - **الحدث :**سيارة تدخل (in&!out)
    - **الحارس (Guard) :** [c < N] هذا هو الشرط الذكي. لن يتم هذا الانتقال إلا إذا كان عدد السيارات الحالي **أقل من السعة القصوى N** . هذا يمنع دخول سيارات إذا كانت الساحة ممتلئة.
    - **الإجراء (Action) :** c:=c+1 نقوم بزيادة قيمة العدّاد c بالقيمة واحد.
    - **المخرج (Output) :** c نعرض القيمة الجديدة للعدّاد.
  - **خروج سيارة (الحلقة السفلية) :** out&!in [c > 0] / c:=c-1; c
    - **الحدث :**سيارة تغادر(out&!in)
    - **الحارس (Guard) :** [c > 0] لن يتم هذا الانتقال إلا إذا كان عدد السيارات **أكبر من صفر** .هذا يمنع النظام من محاولة إنقاص العداد إذا كانت الساحة فارغة أصلاً.
    - **الإجراء (Action) :** c:=c-1 نقوم بإنقاص قيمة العدّاد c بواحد.
    - **المخرج (Output) :** c نعرض القيمة الجديدة للعدّاد.

المقارنة والخلاصة:

- **نموذج FSM** : بسيط نظرياً، ولكنه ضخم وغير عملي إطلاقاً للأنظمة التي تحتوي على أي نوع من العد أو الذاكرة. يتضخم حجمه بشكل مباشر مع حجم البيانات.
- **نموذج EFSM** : أنيق، مختصر، وقابل للتوسع. سواء كانت سعة الساحة 10 سيارات أو 1000 سيارة، يبقى المخطط كما هو: حالة واحدة، متغير واحد، وانتقالان.
- لقد قمنا بفصل **منطق التحكم** (لدينا حالة واحدة فقط هي "التشغيل") عن **البيانات** (المخزنة في المتغير c)

هذا المثال هو أفضل توضيح لمفهوم **"مشكلة انفجار الحالات"** وكيف أن الآلات الممتدة (EFSMs) هي الحل المباشر والفعال لها، وهو ما سنتحدث عنه بالتفصيل في الشريحة القادمة.

State Explosion

- **Complex systems tend to have very large number of states. This particularly is the case in the presence of concurrency.**
  The phenomenon is called *state explosion*.

- **Every global state of a concurrent system must be represented individually**
  ⇒ interleaving of independent actions leads to exponential number of states.

- **Expressing such a system as a FSM (or extended FSM) is very difficult.**

لقد أشرنا في الأمثلة السابقة إلى "مشكلة انفجار الحالات" و التي تعتبر واحدة من أكبر العقبات في تصميم الأنظمة المعقدة.، والآن سوف نتعرف على هذه المشكلة بشكل مفصل.

- **"Complex systems tend to have very large number of states. This particularly is the case in the presence of concurrency. The phenomenon is called *state explosion*."**
  تميل الأنظمة المعقدة إلى امتلاك عدد كبير جداً من الحالات. ويحدث هذا بشكل خاص في وجود التزامن. تسمى هذه الظاهرة بانفجار الحالات.
  - في مثال عدّاد السيارات، رأينا كيف أن عدد الحالات يزداد بشكل خطي مع سعة الموقف. كان ذلك نمواً كبيراً ولكنه يمكن التحكم فيه.
  - المشكلة الحقيقية، أو "الانفجار"، تحدث عندما يكون لدينا **التزامن(concurrency)** ، أي عندما يكون هناك عدة أجزاء أو عمليات في النظام تعمل بشكل متوازٍ ومستقل عن بعضها البعض.

- **"Every global state of a concurrent system must be represented individually ⇒ interleaving of independent actions leads to exponential number of states."**
  يجب تمثيل كل حالة كلية لنظام متزامن بشكل فردي --> تداخل الإجراءات المستقلة يؤدي إلى عدد هائل (أسيّ) من الحالات.
  - هذه هي النقطة الجوهرية التي تفسر سبب المشكلة.
  - **ما هي "الحالة الكلية(Global State)"؟** إنها لقطة (snapshot) لحالة جميع الأجزاء المكونة للنظام في لحظة معينة.
  - **ما هو "التداخل(Interleaving)"؟** هو جميع الطرق الممكنة التي يمكن أن تتشابك بها إجراءات الأجزاء المستقلة.
  - **مثال بسيط جداً لتوضيح الفكرة:**
    - لنتخيل أن نظامنا يتكون من **مصباحين** فقط، وكل مصباح مستقل عن الآخر.
    - المصباح (أ) له حالتان: {مضاء، مطفأ}.

- المصباح (ب) له حالتان أيضاً: {مضاء، مطفأ}.
- إذا أردنا وصف "الحالة الكلية" لنظام المصباحين معاً، فما هي كل الاحتمالات الممكنة؟
  1. أ مطفأ، ب مطفأ
  2. أ مطفأ، ب مضاء
  3. أ مضاء، ب مطفأ
  4. أ مضاء، ب مضاء
- نظامان بسيطان، كل منهما له حالتان، نتج عنهما نظام كلي له $2 \times 2 = 4$ حالات.
- الآن، لنضف مصباحاً ثالثاً (ج). سيصبح عدد الحالات الكلية $2 \times 2 \times 2 = 8$ حالات.
- إذا كان لدينا **10 مصابيح**، سيكون لدينا $2^{10} = 1024$ حالة كلية
- هذا هو **النمو الأسيّ (exponential growth)** . كلما أضفنا مكوناً متزامناً جديداً، فإن عدد الحالات الكلية للنظام يُضرب في عدد حالات هذا المكون الجديد.

- **"Expressing such a system as a FSM (or extended FSM) is very difficult." التعبير عن مثل هذا النظام كآلة حالة محدودة (أو آلة موسعة) صعب للغاية.**
  - محاولة بناء آلة حالة محدودة واحدة لتمثيل نظام العشرة مصابيح ستكون صعبة جداً. سيتعين علينا رسم 1024 حالة وآلاف الانتقالات بينها.
  - حتى الآلة الممتدة (EFSM) ستواجه صعوبة. صحيح أنه يمكننا استخدام 10 متغيرات منطقية (boolean) بدلاً من 1024 حالة، لكن التعقيد لم يختف، بل انتقل إلى "الحراس (Guards)" . ستصبح الشروط في الحراس معقدة جداً للتعامل مع كل التوافيق الممكنة لهذه المتغيرات العشرة.

الخلاصة:

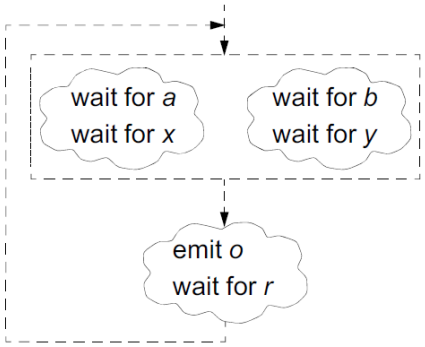
مشكلة انفجار الحالات هي عقبة أساسية تمنعنا من استخدام نموذج آلة الحالة المحدودة الواحدة لنمذجة الأنظمة الواقعية التي تحتوي على مكونات متعددة تعمل بالتوازي.

State Explosion

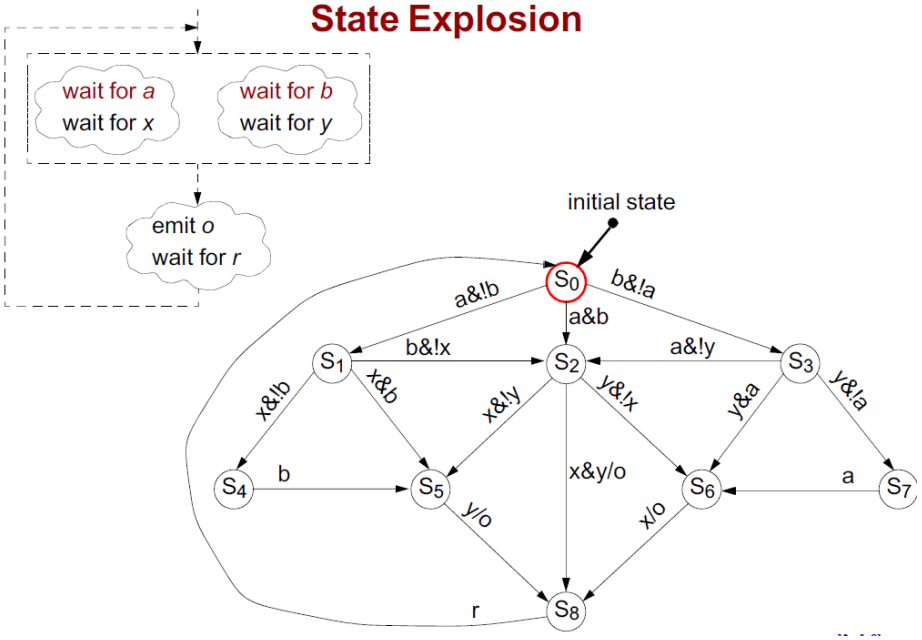
Example

After starting the system, it waits simultaneously for event *a* followed by *x*, and event *b* followed by *y*. Events can arrive in any order, except that *x* follows *a* and *y* follows *b*. Once the events are received, output *o* is emitted. Then the system waits for the reset signal *r* to return into the initial state.

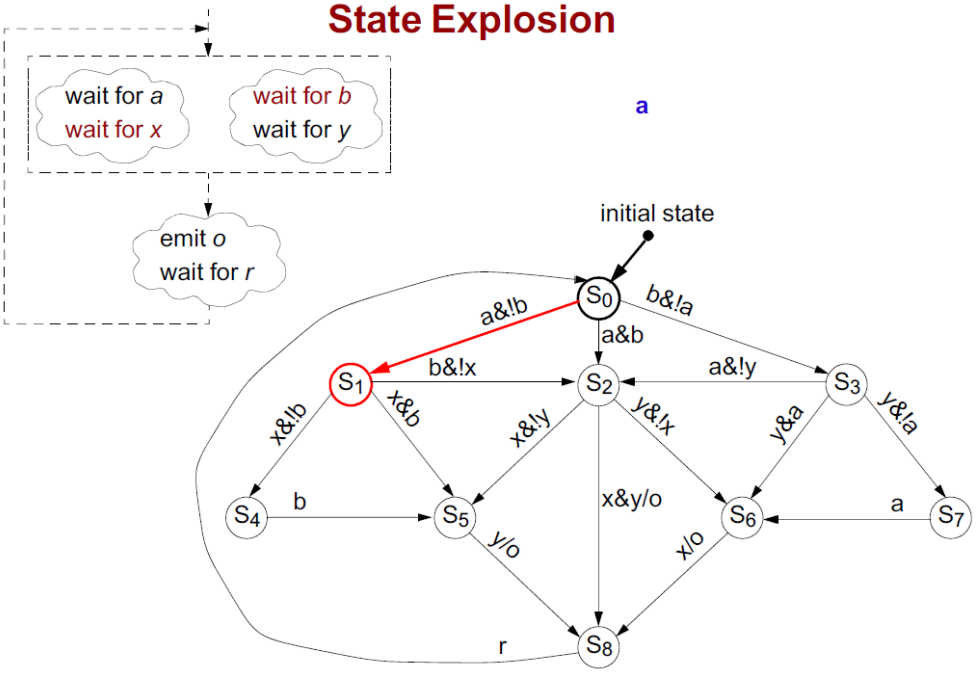
- **Input events: {a, b, x, y, r}**
- **Output: {o}**
- **States: {S_0, S_1, ..., S_8}**



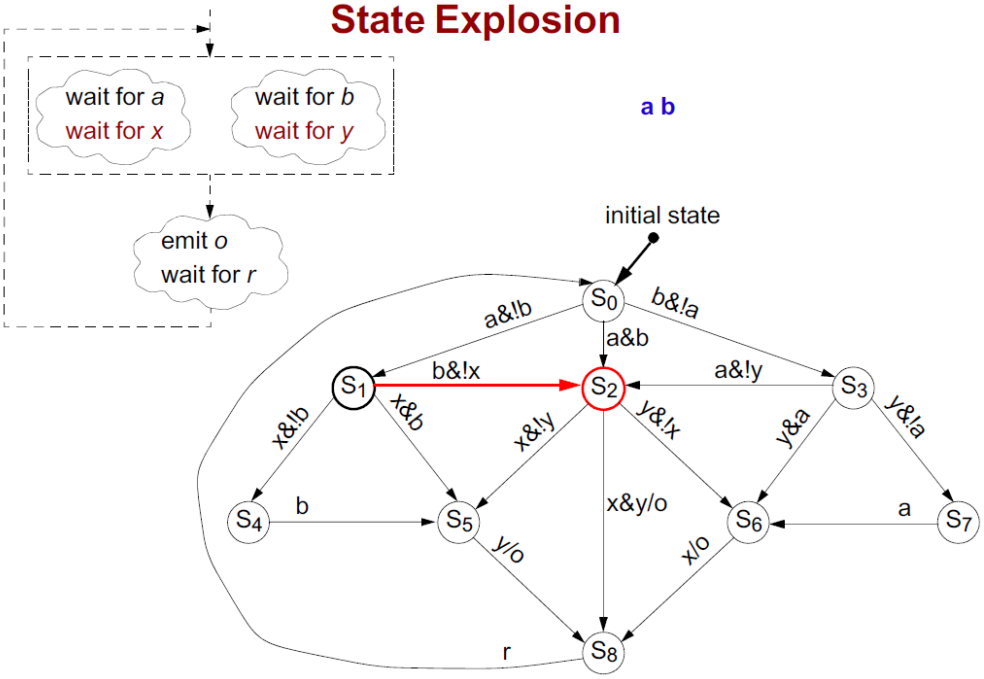
State Explosion



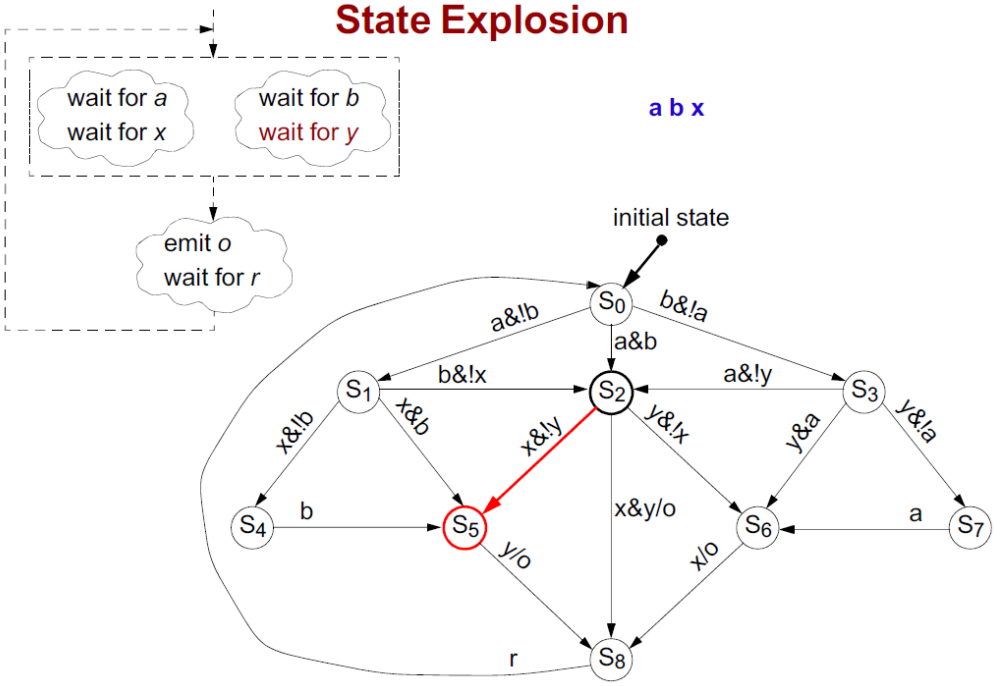
State Explosion



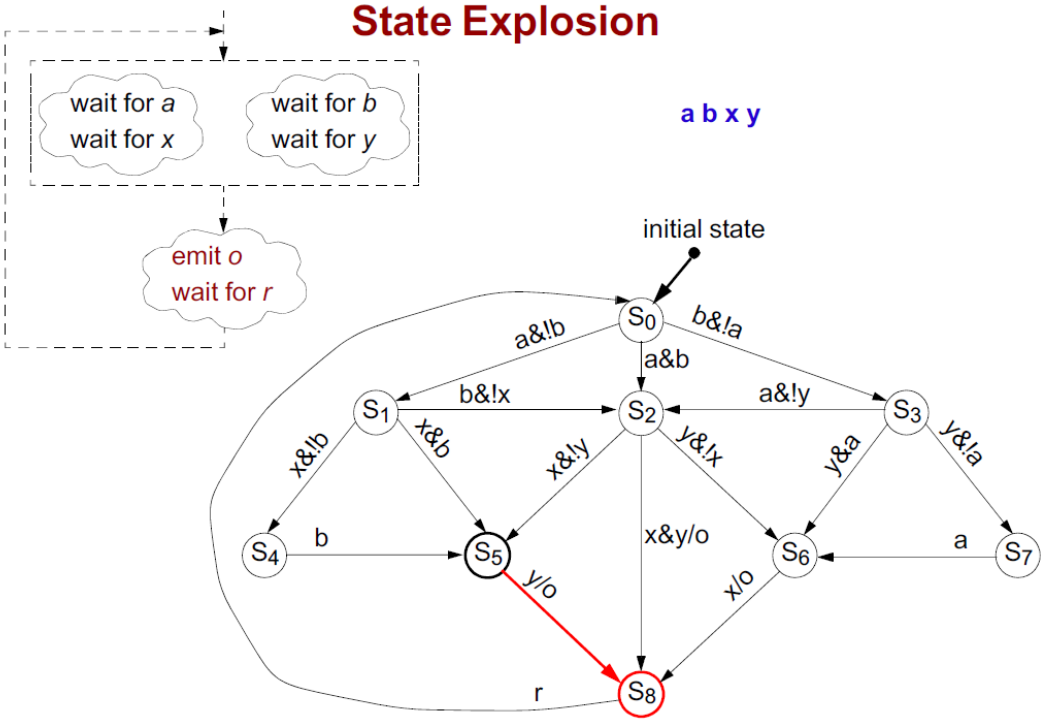
State Explosion



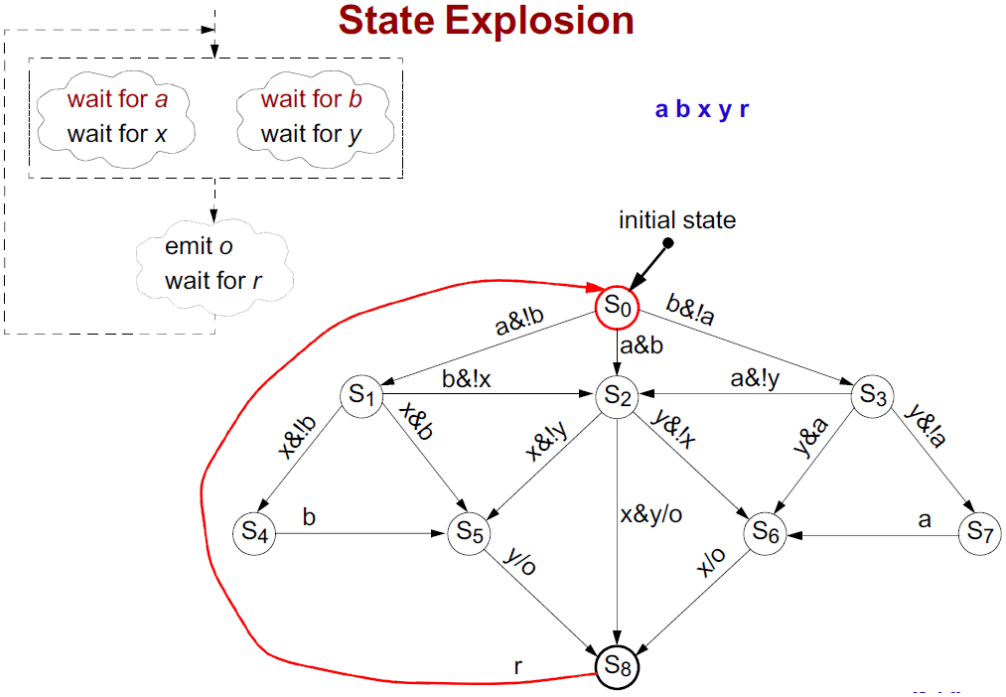
State Explosion



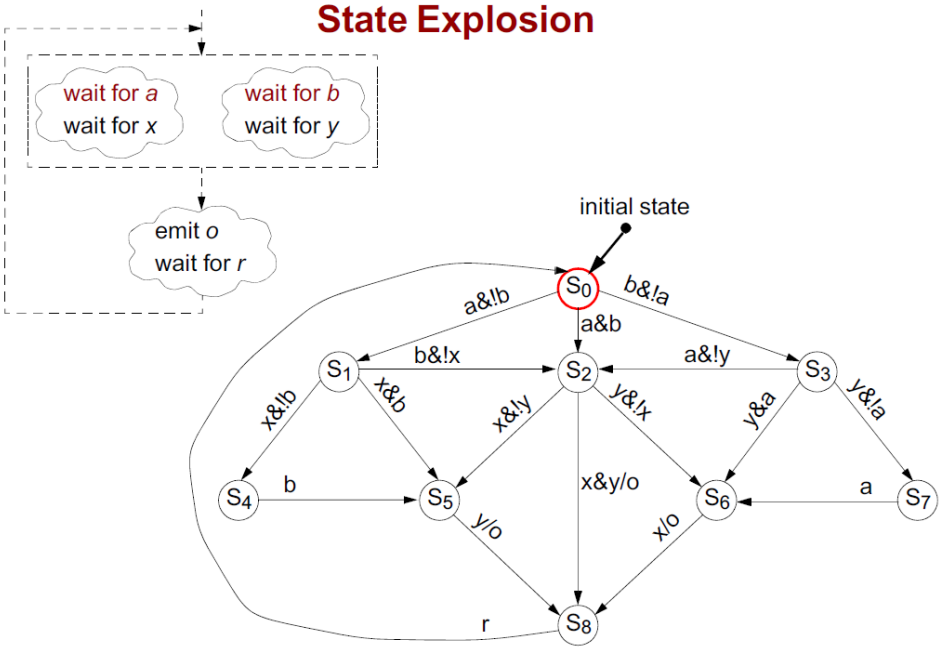
State Explosion



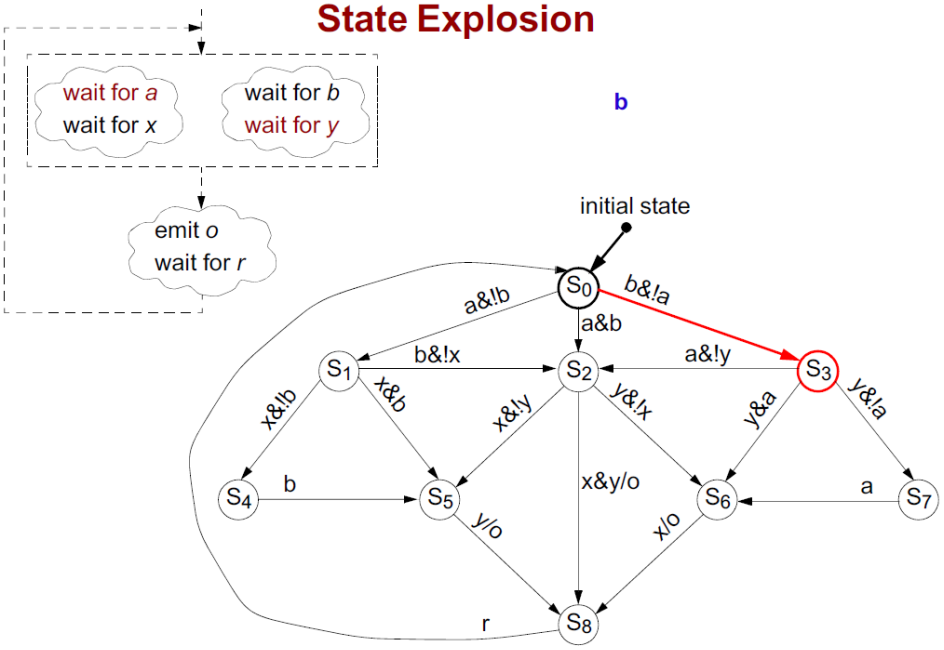
State Explosion



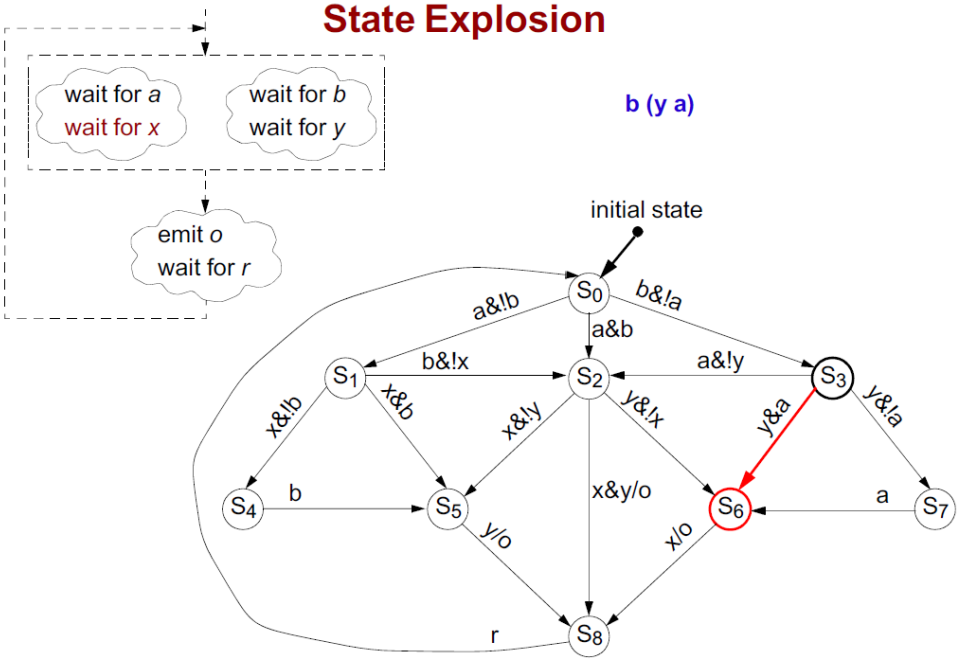
State Explosion

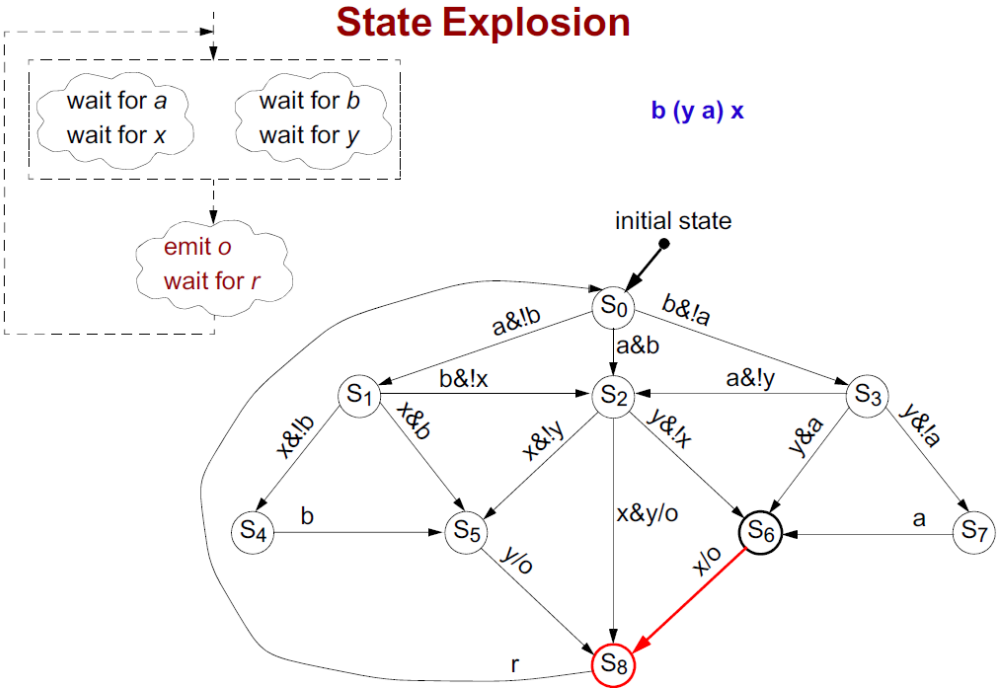


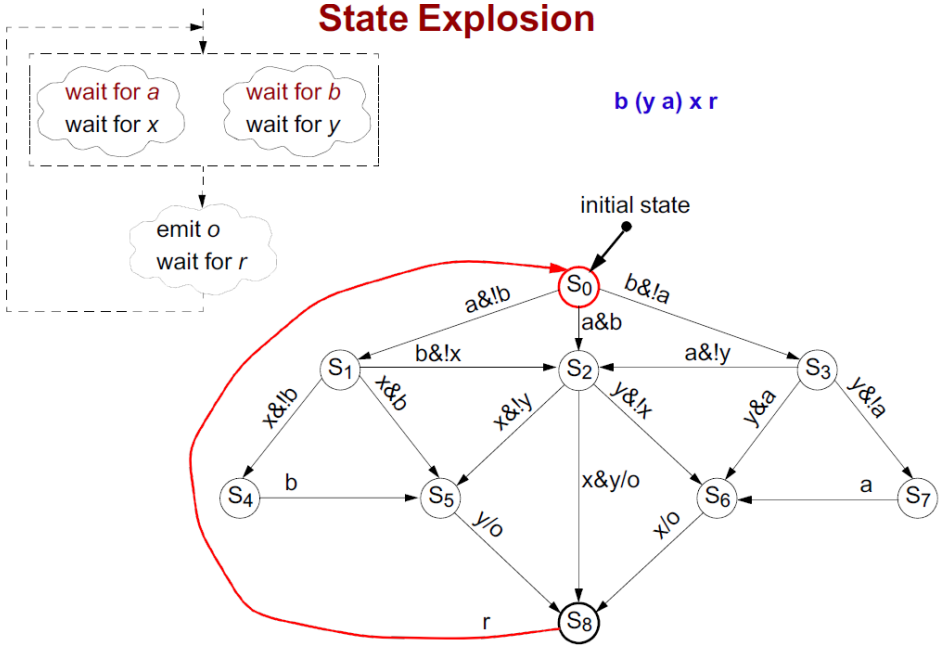
State Explosion



State Explosion







هذا المثال يوضح بشكل ملموس كيف يمكن لمواصفات نظام تبدو بسيطة أن تتحول إلى آلة حالة معقدة جداً بسبب التزامن.

سوف نتتبع المثال الذي تعرضه هذه الشرائح خطوة بخطوة.

وصف المشكلة

أولاً، لنفهم ما هو المطلوب من النظام:

- بعد بدء تشغيل النظام، فإنه ينتظر **بشكل متزامن (simultaneously)** وقوع سلسلتين من الأحداث :
  1. الحدث a متبوعاً بالحدث x
  2. الحدث b متبوعاً بالحدث y
- يمكن للأحداث أن تصل بأي ترتيب (هذا هو **التداخل (interleaving)** الذي تحدثنا عنه)، بشرط أن xدائماً يأتي بعد a، و y دائماً يأتي بعد b
- بمجرد استلام السلسلتين كلتيهما (a-x و b-y)، يتم إصدار المخرج o
- ثم ينتظر النظام إشارة إعادة الضبط r للعودة إلى الحالة الأولية.
- (wait for a, wait for x) و (wait for b, wait for y) هما عمليتان بسيطتان تعملان بالتوازي.

نموذج FSM

الآن، لننظر إلى المخطط الكبير في الأسفل. هذا هو ما يحدث عندما نحاول تحويل هذين المفهومين البسيطين المتوازيين إلى **آلة حالة محدودة واحدة** . لماذا هي معقدة جداً؟ لأن كل حالة (S_0 إلى S_8) يجب أن "تتذكر" التقدم المحرز في **كلا المسارين** في نفس الوقت.

- S_0: الحالة الأولية، لم نستلم شيئاً بعد.
- S_1: استلمنا a فقط.
- S_3: استلمنا b فقط.
- S_2: استلمنا a و b، وننتظر x و y
- وهكذا... كل حالة هي ذاكرة فريدة لجميع التوافيق الممكنة للأحداث التي وردت حتى الآن.

تتبع مسار الأحداث الأول(a ⟶ b ⟶ x ⟶ y ⟶ r)

لنتتبع المسار الأحمر لنرى كيف يتنقل النظام عبر هذه الحالات المعقدة بناءً على تسلسل معين من المدخلات:

- **a يصل (الشريحة 3) :**نبدأ من S_0. عند وصول الحدث a، ننتقل إلى **S_1**
  - **المعنى:**" لقد استلمنا a، وننتظر الآن x لإكمال المهمة الأولى، وما زلنا ننتظر b لبدء المهمة الثانية"
- **b يصل (الشريحة 4) :**نحن في S_1 عند وصول b، ننتقل إلى **S_2**
  - **المعنى:**" لقد استلمنا a و b كليهما. الآن ننتظر x و y "
- **x يصل (الشريحة 5) :**نحن في S_2. عند وصول x، ننتقل إلى **S_5**
  - **المعنى:**" لقد أكملنا الآن سلسلة a-x بالكامل. لم يتبق سوى انتظار y لإكمال المهمة الثانية"
- **y يصل (الشريحة 6) :**نحن في S_5. عند وصول y، ننتقل إلى **S_8** ويتم **إصدار المخرج o**

- **المعنى:**" نجاح. لقد استلمنا الآن كلتا السلسلتين (a-x) و (b-y) النظام يصدر المخرج o كما هو مطلوب.

- **r يصل (الشريحة 7)** :نحن في S_8. عند وصول إشارة إعادة الضبط r، نعود عبر السهم الطويل إلى الحالة الأولية **S_0**
  - **المعنى:**" تمت إعادة ضبط النظام وهو جاهز لبدء الدورة من جديد."

الشرائح 9 إلى 12: تتبع مسار أحداث مختلف(r ⟶ x ⟶ a ⟶ y ⟶ b)

هذا المسار يوضح كيف أن ترتيباً مختلفاً للمدخلات (بسبب التداخل) يأخذنا عبر مسار مختلف تماماً في آلة الحالة، ولكنه يؤدي إلى نفس النتيجة النهائية.

- **b يصل (الشريحة 9)** :نبدأ من S_0 هذه المرة، الحدث bمن المهمة الثانية يصل أولاً. ننتقل إلى **S_3**
  - **المعنى:**" لقد استلمنا b، وننتظر الآن y و a "

- **(y a)يصلان (الشريحة 10)** :نحن في S_3 الشريحة تظهر أن y و a يصلان معاً (أو في تتابع سريع جداً). ننتقل من S_3 عبر y&a إلى **S_6**
  - **المعنى:**" كنا قد استلمنا b. والآن استلمنا y و a هذا يعني أننا أكملنا سلسلة b-y بالكامل، واستلمنا a من السلسلة الأولى. لم يتبق سوى انتظار x "

- **x يصل (الشريحة 11)** :نحن في S_6 عند وصول x، ننتقل إلى **S_8** ويتم **إصدار المخرج o**
  - **المعنى:**" وصل الحدث الأخير x. اكتملت كلتا السلسلتين، والنظام يصدر المخرج"

- **r يصل (الشريحة 12)** :نحن في S_8، والإشارة r تعيدنا إلى **S_0**

الخلاصة

كانت لدينا مهمة تبدو بسيطة: ننتظر سلسلتين من الأحداث بالتوازي. ولكن لتمثيل جميع **التداخلات الممكنة** بين أحداث هاتين السلسلتين، احتجنا إلى هذه الشبكة المعقدة المكونة من **9حالات** S_0 إلى S_8

هذا هو مثال حي وملموس على **"مشكلة انفجار الحالات** ."وصف بسيط ومتزامن "ينفجر" ليولد آلة حالة واحدة ضخمة ومعقدة. هذا يجعل التصميم صعباً، والفهم أصعب، والتحقق من صحة النظام كابوساً.

هذه المشكلة هي الدافع الرئيسي الذي سيقودنا إلى الحل في الشريحة التالية: بدلاً من محاولة حشر كل شيء في آلة واحدة ضخمة، سنتعلم كيف ندير هذا التعقيد باستخدام آلات الحالة المحدودة الهرمية والمتزامنة (Hierarchical and Concurrent FSMs)