جامعة حلب - كلية الهندسة المعلوماتية
سنة رابعة – كافة الأقسام

Parallel Programming Course

Lecture (8)

Fundamentals of Processes and Synchronization

قواعد التنفيذ المتواقت (Semantics of Concurrent Execution)

- إن عبارة التواقت co أو ما يكافئها من التصريح process هي عبارة تحكم. لذلك، فإن تأثيرها يوّصف بقاعدة دلالية التي تستحصل تأثير التنفيذ التفرعي.
- الإجرائيات تتألف من عبارات تسلسلية وعبارات تزامنية مثل await.
- تأثير عبارة await هو أقرب ما يكون لتعليمة if التي من أجلها الحرس B محرراً عندما يبدأ تنفيذ S.

⟨await (B) S;⟩

- وهكذا، فإن قاعدة الدلالة لتعليمة await مشابهة لقاعدة الدلالة لـ if
- Await Statement Rule: $$\frac{\{P \wedge B\}S\{Q\}}{\{P\}\langle await(B)S;\rangle\{Q\}}$$
- الفرضيات تنص أنه "إذا ما بدأ تنفيذ s في حالة يكون فيها كل من P و B محقق، وأن S تمت، عندها Q ستكون محققة".
- والنتيجة تسمح بأن يستدل بأنها الحالة التي ينتج عن تعليمة await الحالة Q إذا ما بدأت في الحالة P مفترضة أن تعليمة await قد تمت.

- قواعد الدلالة لا تنص بشيء عن التأخير الممكن، لأن التأخيرات تؤثر على خواص الحياة (liveness)، وليس على خواص الأمان (safety).
- لنعتبر تأثيرات التنفيذ المتواقت، في المثال المعرف بالتعليمات التالية:

co S1; // S2; // ... // Sn; OC;

- افترض بأن التالي محقق لكل عبارة:

$\{P_i\}\ S_i\ \{Q_i\}$

- وفقاً لتفسير الثلاثيات، هذا يعني بأنه، إذا بدأ S_i في حالة تحقيق P_i وS_i تمت، فإن الحالة ستحقق Q_i.
- لكي يتم التفسير عندما تنفذ الإجرائيات تواقتياً، على الإجرائيات أن تبدأ في حالة تحقيق تقاطعات الشروط P_i.
- إذا ما تمت جميع الإجرائيات، فإن الحالة النهائية ستحقق تقاطعات الشروط Q_i.

- عندها نحصل على قاعدة الدلالة لتعليمة co:

$$\{P_i\}\ S_i\ \{Q_i\} \text{ are free interference /}$$
$$\{P_i \wedge \ldots \wedge P_n\}$$
$$\text{co } S_1;\ //\ \ldots\ //\ S_n;\ \text{oc}$$
$$\{Q_1 \wedge \ldots \wedge Q_n\}$$

- لاحظ الفرضيات، لكي تتحقق النتيجة، يجب على الإجرائيات وبراهينها أن لا تتداخل مع بعضها البعض.
- أحدى الاجرائيات تتداخل مع الأخرى إذا ما نفذت تكليفاً يلغي توكيداً في إجرائية أخرى.
- التوكيدات توّصف ما على الإجرائية أن تفترض أنه محقق قبل وبعد كل تعليمة.
- فإذا قامت إجرائية بتكليف إلى متحول مشترك وبذلك تلغي افتراضاً لإجرائية أخرى، ويصبح برهان الإجرائية الأخرى غير صالح.

- مثال، لنعتبر البرنامج البسيط التالي:

$$\{x == 0\}$$
$$co\ \langle x = x+1;\rangle\ //\ \langle x = x+2;\ \rangle\ oc$$
$$\{x == 3\}$$

- إذا بدأ البرنامج بالحالة التي فيها x يساوي 0، فعندما يتم البرنامج، فإن x تساوي 3. لكن ما هو المحقق بالنسبة لكل إجرائية؟
  - الجواب ولا واحدة من الاجرائيتين تفترض بأن x لا تزال 0 عندما تبدأ، لأن ترتيب التنفيذ غير محدد.
- إذا افترضت إجرائية ما بأن x هي 0 عندما تبدأ، ذلك التوكيد سوف يتداخل إذا ما نفذت الإجرائية الأخرى أولاً. ما هو محقق تم التقاطه في المثال التالي:

- التوكيدات في كل إجرائية تأخذ بالاعتبار إمكانيات التنفيذ حسب الترتيب الممكن للاجرائيات
- لاحظ بأن تقاطع الشروط المسبقة هو فعلاً x==0 وأن تقاطع الشروط اللاحقة هو بالفعل x == 3.
- عرض البرنامج بهذا الشكل يدعى (proof outline) ويتضمن:
  - توكيد قبل وبعد كل تعليمة
  - نتيجة الثلاثيات محققة (ثلاث ثلاثيات واحدة لكل إجرائية وواحدة لتعليمة co)

```
{x == 0}
co {x == 0 ∨ x == 2}
    ⟨x = x+1;⟩
    {x == 1 ∨ x == 3}
// {x == 0 ∨ x == 1}
    ⟨x = x+2;⟩
    {x == 2 ∨ x == 3}
oc
{x == 3}
```

- حدث التكليف (*assignment action*) هو عبارة تكليف أو عبارة await تحوي تعليمة تكليف أو أكثر.
- التوكيد الحرج (*critical assertion*)هو شرط سابق أو لاحق غير مضمن في عبارة await.
- تحقيق عدم التداخل (*non-interference*):
  - إذا كانت a حدث تكليفي في إحدى الإجرائيات و pre(a) هو الشرط المسبق. وأن c هو التوكيد الحرج في إجرائية أخرى. وعندها لن يحدث تداخل a مع c إذا برهنا على أن التالي هو نظرية في البرمجة المنطقية.

$$\{c \wedge pre(a)\}\ a\ \{c\}$$

  - التوكيد الحرج c لامتغير بالنسبة لتنفيذ حدث التكليف a.
  - تضمين الشرط المسبق لـa لأنه يمكن تنفيذ a فقط إذا كانت الإجرائية في حالة تحقق الشرط المسبق لـa.

- وكمثال على استخدام نظرية عدم التداخل اعتبر البرنامج السابق.
  - الشرط السابق للإجرائية الأولى هو توكيد حرج.
  - لا يتداخل مع عبارة التكليف في الإجرائية الثانية لان الثلاثية التالية محققة:

$$\{(x == 0 \vee x == 2) \wedge (x == 0 \vee x == 1)\}$$

$$X = x+2;$$

$$\{x == 0 \vee x == 2\}$$

- الـpredicate الأول يُختصر إلى $x==0$، بعد إضافة 2 إلى x، فإن قيمة x تكون إما 0 أو 2.
- تُعبِّر هذه الثلاثية عن أنه إذا نفذت الإجرائية الثانية قبل الأولى، عندها فإن قيمة x ستكون 2 عندما تبدأ الإجرائية الأولى بالتنفيذ.
- هناك ثلاث توكيدات أخرى في البرنامج السابق:
  - الشرط اللاحق في الإجرائية الأولى والشرط السابق واللاحق من الثانية.

تقنيات التخلص من التداخل (avoiding interference)

- الإجرائيات في البرنامج المتواقت تعمل مع بعضها البعض لحساب النتائج.
- متطلبات بناء البرنامج المتواقت هو أن لا تتداخل إجرائيا ته مع بعضها البعض.
- تكون مجموعة من الاجرائيات متحررة من التداخل إذا لم يوجد حدث تكليفي في أحدى الإجرائيات مع توكيد حرج في الإجرائية الأخرى.
- سنتحدث عن أربعة تقنيات نتغلب فيها على التداخل وهي تقنيات يمكن استخدامها في تطوير البرمجيات المتواقتة وهي:
  - المتحولات المنفصلة (disjoint variables).
  - التوكيدات الضعيفة (weakened assertions).
  - اللامتغيرات العامة (global invariants).
  - التزامن (synchronization).
- تتضمن جميعها وضع تأكيدات وأحداث تكليفيه في شكل يؤكد أن المعادلات الغير متداخلة محققة.

المتحولات المنفصلة (Disjoint Variables)

- نكرر ما ذكرناه سابقاً "مجموعة الكتابة لإجرائية ما هي مجموعة المتحولات التي تكلفها (ويمكن أن تقرأها)، ومجموعة القراءة لإجرائية ما هي مجموعة المتحولات التي تقرأها ولكن لا تغير فيها".
- المجموعة المرجعية (*reference*) لإجرائية ما هي مجموعة المتحولات التي تظهر في التوكيدات في برهان ما لهذه الإجرائية.
- هذه المجموعة سوف تكون بالغالب نفس المجموعة الناتجة عن اجتماع مجموعة القراءة والكتابة، ويمكن أن لا تكون نفسها.
- باعتبار التداخل، المتحولات الحرجة هي المتحولات في التوكيدات.
- إذا تم فصل مجموعة الكتابة لإجرائية ما عن المجموعة المرجعية لإجرائية أخرى، والعكس بالعكس، فإن الإجرائيتين لا يمكن أن تتداخلان.

- كمثال، اعتبر مايلي:

```
co x = x+1; // y = y+1; oc
```

إذا كان كل من x و y مهيأ بقيمة 0، عندئذ من التكليف البديهي، فإن كل من العبارتين التاليتين تعتبر نظرية:

```
{x == 0} x = x+1; {x == 1}
{y == 0} y = y+1; {y == 1}
```

- كل إجرائية تحتوي على عبارة تكليف وتوكيدين؛ لذلك يوجد أربع نظريات تثبت عدم التداخل.
  - كل منها سيكون محققاً ببساطة لأن كلا الإجرائيتين ترجعان إلى متحولات مختلفة.
- مجموعات الكتابة / المرجعية المنفصلة تعتبر قاعدة لمعظم الخوارزميات التكرارية التفرعية، كما في خوارزمية جداء المصفوفات سابقاً.
- كمثال آخر، يمكن البحث في برنامج تشغيل الألعاب عن الفروع المختلفة لشجرة إمكانيات الانتقال على التوازي.

Weakened Assertions

- حتى عندما تتراكب مجموعات الكتابة والمرجعية، بإمكاننا في بعض الأحيان تجاوز التداخل بواسطة تضعيف التوكيدات لكي نأخذ بالاعتبار معظم تأثيرات التنفيذ المتواقت.
- التوكيدات الضعيفة تسمح بحالات أكبر للبرنامج من التوكيدات الأخرى التي يمكن أن تتحقق لإجرائية بالعزل كما رأينا سابقاً البرنامج التالي:

```
{x == 0}
co {x == 0 ∨ x == 2}
    ⟨x = x+1; ⟩
    {x == 1 ∨ x == 3}
// {x == 0 ∨  x == 1}
    ⟨x = x+2; ⟩
    {x == 2 ∨ x == 3}
oc
{x == 3}
```

- الشروط المسبقة واللاحقة في كل إجرائية هي أضعف مما تكون فيه في حالة العزل.
  - كل إجرائية يمكن أن تؤكد في العزل بأنه إذا تم تهيئة x بالصفر، عندها فإنه عند إتمام الإجرائية تكون قيمة x هي 1 للإجرائية الأولى أو 2 للإجرائية الثانية. لكن، هذه التوكيدات الأقوى يمكن أن تتداخل.
- التوكيدات الضعيفة تمتلك تطبيقات أكثر واقعية من المسألة البسيطة أعلاه.
- مثال إجرائية عمليات الشدولة على هارد ديسك.
  - تحشر اجرائيات أخرى عمليات ضمن الرتل؛ عندما يكون الديسك في حالة Idle، فإن المشدول يختبر الرتل، ينتخب العملية الأفضل وفقاً لقاعدة ما، ويبدأ العملية.
  - على الرغم من أن المشدول يمكن أن يكون قد اختار العملية الأفضل في الوقت الذي فحص الرتل، فإنه ليست هي الحالة دوماً بأن ينجز الهارد العملية الأفضل.
    - والسبب لان أي إجرائية يمكن أن تدرج عملية أخرى إلى الرتل تماماً بعد أن تم الاختيار. وهكذا فالأفضل في هذه الحالة تابع للوقت، وهذا كاف بالنسبة للشدولة.

- مثال آخر، هناك الكثير من الخوارزميات التفرعية للحلول التقريبية للمعادلات التفاضلية الجزئية تملك الشكل التالي:
  - فضاء المسألة يقرب بشبكة محدودة من النقاط، لنقل grid[n,n].
  - توكل كل نقطة من الشبكة أو بلوك من نقاط الشبكة إلى إجرائية،

```
double grid[n,n]
process PDE[i = 0 to n-1, j = 0 to n-1] {
    while (not converged) {
        grid[i,j] = f(neighbouring points);
    }
}
```

  - إن f المحسوب في كل تكرار يمكن، أن تكون مثلاً معدل أربع نقاط متجاورة في نفس السطر والعمود.
  - في كثير من المسائل تكون القيمة المنسوبة لـgrid[i,j] في أحدى التكرارات تعتمد على قيم الجوار من التكرارية التي تسبقها.
  - وهكذا، فالحلقة اللامتغيرة توّصف هذه العلاقة بين القيمة القديمة والجديدة لنقاط الشبكة.

- لكي نتأكد بأن الحلقة اللامتغيرة في كل إجرائية لا تتداخل، يجب على الإجرائيات أن تستخدم مصفوفتين وأن تتزامن بعد كل تكرار.
- في كل تكرار كل إجرائية PDE تقرأ قيم من أحد المصفوفات، تحسب f، بعدها تسلم النتيجة إلى المصفوفة الثانية.
- عندها تنتظر كل إجرائية حتى تحسب جميع الإجرائيات القيمة الجديدة لنقاطها الشبكية.
  - سنرى في محاضرة قادمة كيف يمكن بناء هذا التزامن باستخدام الـbarrier.
- تتبادل المصفوفتين وظيفة كل منها، وتقوم الإجرائيات بتنفيذ تكرارية جديدة.
- هناك طريقة أخرى لتحقيق هذا النوع من التزامن بواسطة تنفيذ الإجرائيات باستخدام الخطوة المقفولة lockstep، مع كل إجرائية تنفذ نفس العمل في نفس الوقت.
- تدعم المعالجات المتعددة المتزامنة هذا الشكل من التنفيذ.
- هذه التقنية تتجنب التداخل بما أن كل إجرائية تقرأ القيم القديمة من الشبكة قبل أن تقوم أي إجرائية بتجديد هذه القيم.

اللامتغيرات العامة (Global Invariants)

- هي تقنية أخرى، وفعالة جداً في تجنب التداخل باستخدام اللامتغير العام لتقصي العلاقات بين المتحولات المشتركة.
- في الحقيقة، يمكن استخدام لامتغير عام ليقود تطوير حل لأي مسألة تزامن.
- ليكن I مسبب يرجع إلى متحولات عامة.
  - يكون I لامتغير عام بالنسبة لمجموعة إجرائيات إذا حقق الشروط التالية:
    - I محقق عندما تبدأ الإجرائيات بالتنفيذ
    - I مصان من قبل كل حدث تكليفي.
  - يتحقق الشرط الثاني إذا كان، من أجل كل حدث تكليفي a، يكون I محقق بعد تنفيذ الحدث بافتراض أن I محقق قبل تنفيذ a.

التزامن (synchronization)

- التسلسلات من العبارات التكليفية التي هي ضمن عبارات await تبدو للإجرائيات الأخرى كوحدات غير قابلة للتجزئة.
- لذلك يمكن أن نهمل تأثيرات العبارات المفردة الذاتية عندما نعتبر إمكانية تداخل إجرائية مع أخرى.
- مثال، اعتبر الحدث الجزيئي التالي:

⟨x = x+1; y = y+1;⟩

- ولا واحدة من التكاليف بذاتها تسبب تداخل، لأنه لا يوجد إجرائية أخرى ترى الحالة التي فيها x تزيد و y لم تزد بعد.

- الحالات الداخلية لأجزاء البرنامج ضمن الأقواس الزاوية هي أيضاً غير مرئية.
  - لذلك لا يوجد توكيد حول الحالة الداخلية يمكن أن يتداخل مع إجرائية أخرى.
- مثال إن التوكيد في وسط الحدث الجزيئي التالي ليس توكيداً حرجاً:

$\{x == 0 \wedge y == 0\}$

$\langle x = x+1;\ \{x == 1 \wedge y == 0\}\ y = y+1 \rangle$

$\{x == 1 \wedge y == 1\}$

- هذه الخاصتين للأحداث الجزيئية تؤول إلى طريقتين في استعمال التزامن للتغلب على التداخل:

Mutual exclusion, and condition synchronization

استبعاد التشارك (*Mutual exclusion*)

- اعتبر ما يلي:

```
co P1: ... a; ...
// P2: ... S1; {c} S2; ...
oc
```

- a عبارة تكليف في الإجرائية P1، و S1 و S2 عبارات في الإجرائية P2. التوكيد الحرج هو الشرط المسبق لـS2.
- افترض أن a تتداخل مع c.
- احدى الطرق لتجنب التداخل هو باستعمال الـmutual exclusion لكي نغطي توكيد c عن a.
- هذا يتم بدمج العبارتين S1 و S2 في الإجرائية الثانية إلى حدث جزيئي مفرد بالشكل:

⟨S1; S2;⟩

- وهذا يجعل تنفيذ كل من S1 و S2 يتم جزيئياً وبالتالي يجعل الحالة c غير مرئية للاجرائيات الأخرى.

التزامن المشروط (*condition synchronization*)

- الطريقة الأخرى لتجنب التداخل هو باستعمال التزامن المشروط لتقوية الشرط المسبق لـa.
  - نستطيع استبدال a بالحدث الجزيئي الشرطي التالي:

⟨await (!c or B) a;⟩

- B هو *predicate* يوصف مجموعة من الحالات التي من أجلها تنفيذ a يجعل c محقق.
- لذلك تتجنب العبارة أعلاه التداخل إما عن طريق انتظار c لتصبح غير محققة، وبالتالي لا يمكن لـS2 أن تنفذ.
- أو عن طريق التأكيد على أن تنفيذ a سيجعل c محققة وعندها لا ضير من تنفيذ S2.

Example: Array copy problem

- معظم البرامج المتواقت تستخدم تركيب من التقنيات السابقة. وهنا سوف نستعرضها جميعاً في برنامج بسيط واحد.
  - كما وجدنا في مثال consumer/producer استخدمنا حيز ذاكري buf لنسخ محتوى المصفوفة a في إجرائية المنتج إلى مصفوفة b في إجرائية المستهلك.
  - يتناوب كل من المنتج والمستهلك الدخول إلى الـbuf. أولاً المنتج يودع العنصر الأول من a في buf، بعدها المستهلك يجلب ذلك العنصر، بعدها يودع المنتج العنصر الثاني من a، وهكذا.
  - المتحولات p و c تعد عدد الحدود التي تم ايداعها وجلبها على التوالي.
  - التعبير await تستخدم من أجل تزامن الورود إلى الـbuf.
  - عندما p==c الحيز الذاكري يكون فارغاً وعندما p>c فالحيز يكون مليئاً.

- افترض أن محتوى a[n] هو تشكيلة من القيم A[n]. وأن A[i] تدعى بالمتحولات المنطقية؛ وهي ببساطة حجرة محجوزة لمتحول مهما يكن نوعه.
- الهدف هو برهان أن، مع اتمام البرنامج السابق، فإن محتوى b[n]، نفسه A[n]، القيم الموجودة في المصفوفة a.
- يمكن البرهان باستخدام global invariant التالي:

$$PC: c <= p <= c+1 \wedge a[0:n-1] == A[0:n-1] \wedge (p == c+1) \Rightarrow (buf == A[p-1])$$

- بما أن الإجرائيات تتناوب الورود إلى الـbuf، في كل وقت p يساوي إلى c، أو p يكون أكبر بواحد من c.
- المصفوفة a لا تتغير، لذلك a[i] هو دائماً يساوي A[i].
- أخيراً، عندما يصبح الحيز الذاكري مليئاً (أي p==c+1)، فإنه يحوي A[p-1].
- نلاحظ أن الـpredicate (PC) محققة في البداية، لان كلاً من p و c تهيأ بداية بالصفر.
  - فإنها تصان مع كل عبارة تكليف، كما هو موضح في البرنامج

```
int buf, p = 0, c = 0;
{PC: c <= p <= c+1 ∧ a[0:n-1] == A[0:n-1] ∧
        (p == c+1) ⇒ (buf == A[p-1])}
Process Producer {
    Int a[n];     #assume a[i] is initialized to A[i]
    {IP: PC ∧ p <= n}
    While (p < n) {
        {PC ∧ p < n}
        ⟨await (p == c);⟩          # delay until buffer empty
        {PC ∧ p < n ∧ p == c}
        buf = a[p];
        {PC ∧ p < n ∧ p == c ∧ buf == A[p]}
        p = p+1;
        {IP}
        }
    {PC ∧ p == n}
    }
```

```
process Consumer {
    int b[n];
    {IC: PC ∧ c <= n ∧ b[0:c-1] == A[0:c-1]}
    while (c < n) {
        {IC ∧ c < n}
        ⟨await  (p > c);⟩          # delay until buffer full
        {IC ∧ c < n ∧ p > c}
        b[c] = buf;
        {IC ∧ c < n ∧ p > c ∧ b[c] == A[c]}
        c = c+1;
        {IC}
    }
    {IC ∧ c == n}
}
```

- *IP* هو اللامتغير للحلقة في اجرائية المنتج و *IC* هو اللامتغير للحلقة في إجرائية المستهلك. المسببة *IP* و *IC* تنتميان إلى المسبب PC.
- يوجد توكيد قبل وبعد كل عبارة، وكل ثلاثية هي محققة.
- الثلاثيات في كل إجرائية تتبع مباشرة من عبارات التكليف في كل إجرائية.
- بافتراض أن كل إجرائية تأخذ فرصة للتنفيذ، عبارات await تتم عندما يصبح الحرس الأول محقق، وبعدها الآخر، وهكذا.
- لذلك، كل إجرائية تتم بعد n تكرارية. عندما يتم البرنامج، الشروط المسبقة لكلا الإجرائيتين تكون محققة. لذلك الحالة النهائية للبرنامج تحقق السببية

$$PC \wedge p == n \wedge IC \wedge c == n$$

- تملك المصفوفة b تتضمن نسخة من المصفوفة a.
- التوكيدات في كلا الإجرائيتين لا تتداخل مع بعضها البعض. معظمها عبارة عن تركيب من اللامتغيرات العامة *PC* ومسببات محلية. لذلك فإنها تقابل متطلبات عدم التداخل.
- الاستثناءات الأربع هي التوكيدات التي تحدد العلاقة بين قيم المتحولات المشتركة p و c. هؤلاء لا تتداخل مع بعضها بسبب عبارات await في البرنامج.

- وظيفة عبارات *await* في برنامج نسخ المصفوفة هو التأكد بأن اجرائيات المنتج والمستهلك تتبادلان الدخول إلى الحيز الذاكري.
- يلعب هذا دوريين بالنسبة للتغلب على التداخل.
  - الأول، تتأكد بأن الإجرائيات لا تستطيع دخول buf بآن واحد؛ وهذا مثال عن الـ*mutual exclusion*.
  - الثاني، تتأكد بأن المنتج لا يكتب فوق الحدود (*overflow*) وأن المستهلك لا يقرأ حد ما أكثر من مرة (*underflow*) وهذا مثال على الـ*condition synchronization*.

الخلاصة

- على الرغم من أن البرنامج بسيط فإنه يوضح كل التقنيات الأربعة للتغلب على التداخل.
- أولاً، تم فصل عدد من العبارات وعدد من أجزاء التوكيد في كل إجرائية.
- ثانياً، استخدمنا التوكيدات الضعيفة بالنسبة إلى قيم المتحولات المشتركة؛ فمثلاً، قلنا بأن buf == A[p-1]، لكن فقط إذا كان p == c+1.
- ثالثاً، استخدمنا *PC global invariant* للتعبير عن العلاقة بين قيم المتحولات المشتركة؛ بالرغم من أن كل متحول يتغير مع تنفيذ البرنامج، هذه العلاقة لا تتغير!
- أخيراً، استخدمنا عبارة التزامن await للتأكد من أن كل من الـ mutual exclusion و condition synchronization مطلوب لهذا البرنامج.

الوظيفة