程序验证方法
研究生课程
Chapter 12 (12.1-12.6)
Fairness

朱惠彪
华东师范大学 软件工程学院

Outline

- **12.1 The Concept of Fairness**
- **12.2 Transformational Semantics**
- **12.3 Well-Founded Structures**
- **12.4 Random Assignment**
- **12.5 Schedulers**
- **12.6 Transformation**

12.1 The Concept of Fairness

Fairness models the idea of "true parallelism", where every component of a parallel program progresses with unknown, but positive speed.

In other words, every component eventually executes its next enabled atomic instruction.

$$
\begin{aligned}
PU1 \equiv\ & signal := \textbf{false}; \\
& \textbf{do}\ \neg signal \rightarrow \text{"print next line"} \\
& \square\ \ \neg signal \rightarrow signal := \textbf{true} \\
& \textbf{od}.
\end{aligned}
$$

Assuming that the $signal := true$ is eventually executed, the program $PU1$ terminates.

To enforce termination one has to assume fairness.

Two variants of fairness

- **Weak fairness：requires that every guarded command of a do loop, which is from some moment on continuously enabled, is activated infinitely often.**
- **Strong fairness: requires that every guarded command that is enabled infinitely often is also activated infinitely often.**

$$
\begin{aligned}
PU2 \equiv\ & signal := \textbf{false};\ \textit{full-page} := \textbf{false};\ \ell := 0; \\
& \textbf{do}\ \neg signal \rightarrow\ \text{“print next line”}; \\
& \qquad\qquad\qquad\quad \ell := (\ell + 1)\ \textbf{mod}\ 30; \\
& \qquad\qquad\qquad\quad \textit{full-page} := \ell = 0 \\
& \square\ \ \neg signal \wedge \textit{full-page} \rightarrow signal := \textbf{true} \\
& \textbf{od}.
\end{aligned}
$$

In this book, we understand by *fairness* the notion of strong fairness.

Selections and Runs

- A selection (of n components) is a pair $(\boldsymbol{E}, \boldsymbol{i})$
  - $\boldsymbol{E} \subseteq \{\mathbf{1}, \dots, \boldsymbol{n}\}$ of enabled components
  - an activated component $\boldsymbol{i} \in \boldsymbol{E}$
- A run (of n components) is a finite or infinite sequence of selections.

$$(E_0, i_0). . .(E_j , i_j).$$

Example

$$
\begin{aligned}
PU1 \ \equiv\ & signal := \mathbf{false}; \\
& \mathbf{do}\ \neg signal \rightarrow \text{“print next line”} \\
& \square\ \ \neg signal \rightarrow signal := \mathbf{true} \\
& \mathbf{od}.
\end{aligned}
$$

A computation of PU1 that exclusively activates the first component yields the run:

$$(\{1,2\},1)(\{1,2\},1)\ldots(\{1,2\},1)\ldots .$$

Since the index 2 is never activated, the run and hence the computation is **not fair.**

Every fair computation of PU1 is finite, yielding a run of the form:

$$(\{1,2\},1)\ldots(\{1,2\},1)(\{1,2\},2).$$

Fair Nondeterminism Semantics

$$\begin{aligned}\mathcal{M}_{fair}[\![S]\!](\sigma) = \quad & \{\tau \mid < S, \sigma > \ \rightarrow^* \ < E, \tau >\} \\ \cup \ & \{\perp \mid S \text{ can diverge from } \sigma \ \boxed{\text{in a fair computation}}\} \\ \cup \ & \{\mathbf{fail} \mid S \text{ can fail from } \sigma\}.\end{aligned}$$

$$\begin{aligned}\mathcal{M}_{tot}[\![S]\!](\sigma) = \quad & \mathcal{M}[\![S]\!](\sigma) \\ \cup \ & \{\perp \mid S \text{ can diverge from } \sigma\} \\ \cup \ & \{\mathbf{fail} \mid S \text{ can fail from } \sigma\}.\end{aligned}$$

Example

$$
\begin{aligned}
PU3 \equiv\ & signal := \textbf{false};\ count := 0; \\
& \textbf{do}\ \neg signal \rightarrow \text{“print next line”}; \\
& \qquad\qquad\qquad count := count + 1 \\
& \square\ \ \neg signal \rightarrow signal := \textbf{true} \\
& \textbf{od}.
\end{aligned}
$$

count: counts the number of lines printed. For $i \geq 0$, $\sigma_i(count) = i$

we obtain

$$\mathcal{M}_{tot}[\![PU3]\!](\sigma) = \{\sigma_i \mid i \geq 0\} \cup \{\perp\}$$

but

$$\boxed{\mathcal{M}_{fair}[\![PU3]\!](\sigma) = \{\sigma_i \mid i \geq 0\}.}$$

Under the **assumption of fairness**, **PU3** **always terminates** but still there are **infinitely many final states** **possible: $\boldsymbol{\sigma_i\ with\ i \geq 0}$.**
This differs from the bounded nondeterminism (Bounded Nondeterminism Lemma 10.1, P353), it is called unbounded nondeterminism.

12.2 Transformational Semantics

- **We are looking for a transformation T_{fair} which transforms each nondeterministic program S into another nondeterministic program $T_{fair}(S)$ satisfying the semantic equation**

$$\mathcal{M}_{fair}[\![S]\!] = \mathcal{M}_{tot}[\![T_{fair}(S)]\!]. \qquad (12.1)$$

T_{fair} provides us with information on how to implement fairness.

Conclusion of (12.1)

$$\models_{fair} \{p\}\ S\ \{q\} \text{ iff } \models_{tot} \{p\}\ T_{fair}(S)\ \{q\}, \qquad (12.2)$$

A program S is correct in the sense of fair total correctness if and only if its transformed version $T_{fair}(S)$ is correct in the sense of usual total correctness.

12.3 Well-Founded Structures

Definition 12.1. **Let (P, <) be an *irreflexive partial order*; that is, let P be a set and < an irreflexive transitive relation on P. We say that < is *well-founded on a subset* W ⊆ P if there is no infinite descending chain**

$$\ldots < w_2 < w_1 < w_0$$

of elements $w_i \in W$. The pair (W, <) is then called a *well-founded structure*. If w < w′ for some w, w′ ∈ W we say that w is *less than* w′ or w′ is *greater than* w.

Of course, the natural numbers form a well-founded structure (N,<) under the usual relation <. But also the extension (N ∪ {ω},<), with ω denoting an "unbounded value" satisfying

$$n < \omega$$

for all n ∈ N, is well-founded.

12.4 Random Assignment

- $\boldsymbol{M_{tot}}$ **yields bounded nondeterminism (Lemma 10.1),** $\boldsymbol{M_{fair}}$ **yields unbounded nondeterminism (Example 12.2).**
- **The transformed program** $\boldsymbol{T_{fair}(S)}$ **uses an additional language construct : the random assignment.**

X:=?

It assigns an **arbitrary nonnegative integer** to the **integer variable x**.

The **random assignment** is an explicit form of **unbounded nondeterminism**.

The **random assignments** will enable us to reason about programs **under fairness assumptions.**

Random Assignment: Semantics

(xxvi) $\langle x := ?, \sigma\rangle \rightarrow \langle E, \sigma[x := d]\rangle$

for every natural number $d \geq 0$.

- The random assignment terminates for any initial state, but there are infinitely many possibilities for the final state.

- $\mathcal{N}[[x := ?]](\sigma) = \{\sigma[x := d] \mid d \geq 0\}$

  for a proper state σ and $\mathcal{N} = \mathcal{M}$ or $\mathcal{N} = \mathcal{M}_{tot}$.

Random Assignment: Verification

AXIOM 37: RANDOM ASSIGNMENT

$$\{\forall x \geq 0 : p\}\ x :=?\ \{p\}$$

PROOF SYSTEM *PNR* :

This system consists of the proof system *PN* augmented with axiom 37.

Random Assignment---Verification

RULE 38: REPETITIVE COMMAND III

$$\frac{\begin{array}{l}\{p \wedge B_i\}\ S_i\ \{p\}, i \in \{1,\ldots,n\},\\ \{p \wedge B_i \wedge t = \alpha\}\ S_i\ \{t < \alpha\}, i \in \{1,\ldots,n\},\\ p \rightarrow t \in W\end{array}}{\{p\}\ \mathrm{do}\ \square_{i=1}^{n}\ B_i \rightarrow S_i\ \mathrm{od}\ \{p \wedge \bigwedge_{i=1}^{n}\ \neg B_i\}}$$

Guarantees the termination of
the whole repetitive command

where
(i) t is an expression which takes values in an irreflexive partial order (P,<) that is well-founded on the subset W ⊆ P,
(ii) α is a simple variable ranging over P that does not occur in p, t, B_i or S_i for i ∈ {1, . . . , n}.

RULE 33: REPETITIVE COMMAND II

$$\frac{\begin{array}{l}\{p \wedge B_i\}\ S_i\ \{p\}, i \in \{1,\ldots,n\},\\ \{p \wedge B_i \wedge t = z\}\ S_i\ \{t < z\}, i \in \{1,\ldots,n\},\\ p \rightarrow t \geq 0\end{array}}{\{p\}\ \mathrm{do}\ \square_{i=1}^{n}\ B_i \rightarrow S_i\ \mathrm{od}\ \{p \wedge \bigwedge_{i=1}^{n}\ \neg B_i\}}$$

where t is an integer expression and z is an integer variable not occurring in p, t, B_i or S_i for i ∈ {1, . . . , n}.

PROOF SYSTEM *TNR* :

This system is obtained from the proof system *TN* by adding axiom 37 and replacing rule 33 by rule 38.

12.5 Scheduler

- **Schedulers explain how to implement fairness.**
- **Develop a transformation T_{fair}**

➢ The **development of a scheduler** that enforces fairness in runs,

➢ The **embedding of the schedulers** into nondeterministic programs.

The Scheduler *FAIR*

For n components it is defined as follows:

- **The scheduler state is given by n integer variables $z_1, \ldots, z_n$,**

Represent priorities assigned to the n components.
A component i has higher priority than a component j if $z_i < z_j$

- **This state is initialized nondeterministically by the random assignments**

$$\mathrm{INIT} \equiv z_1 :=?; \ldots; z_n :=?$$

The Scheduler *FAIR*

A component i has higher priority than a component j if $z_i < z_j$

- this state is initialized nondeterministically by the random assignments

$$INIT \equiv z_1 :=?; \ldots; z_n :=?,$$

- the scheduling relation $sch(\sigma, (E, i), \sigma')$ holds iff σ, E, i, σ' are as follows:

(i) σ is given by the current values of $z_1, \ldots, z_n$,

(ii) E and i satisfy the condition

$$SCH_i \equiv z_i = \min \{z_k \mid k \in E\},$$

If during a run FAIR is presented with a set E of enabled components, it selects a component $i \in E$ that has maximal priority

(iii) σ' is obtained from σ by executing

$UPDATE_i \equiv z_i :=?;$
for all $j \in \{1, \ldots, n\} - \{i\}$ do
if $j \in E$ then $z_j := z_j - 1$ fi
od,

Reset the priority of the selected component i

Guarantees that the priorities of all enabled but not selected components j get increased.

12.6 Transformation

- **Given a nondeterministic program**

$$S \equiv S_0;\ \mathbf{do}\ \square_{i=1}^{n}\ B_i \rightarrow S_i\ \mathbf{od},$$

$$\boxed{SCH_i \equiv z_i = min\ \{z_k \mid k \in E\}}$$

- **The transformed program $T_{fair}(S)$ is obtained by embedding the scheduler *FAIR* into S:**

The guard of the ith component can be passed only if it is enabled and selected by FAIR

$$T_{fair}(S) \equiv S_0;\ INIT;$$
$$\mathbf{do}\ \square_{i=1}^{n}\ \boxed{B_i \wedge SCH_i} \rightarrow UPDATE_i;\ S_i\ \mathbf{od},$$

where we interpret E as the set of indices k ∈ {1, . . . , n} for which B_k holds: E = {k | 1 ≤ k ≤ n ∧ B_k}.

Transformation

- **Expanding the abbreviations $INIT, SCH_i, UPDATE_i$ from FAIR yields:**

$$
\begin{array}{l}
T_{fair}(S) \equiv S_0;\ z_1 :=?;\ \ldots;\ z_n :=?; \\
\quad \mathbf{do}\ \square_{i=1}^{n}\ B_i \wedge \boxed{z_i = min\ \{z_k \mid 1 \leq k \leq n \wedge B_k\}} \rightarrow \\
\qquad z_i :=?; \\
\qquad \mathbf{for\ all}\ j \in \{1, \ldots, n\} - \{i\}\ \mathbf{do} \\
\qquad\quad \boxed{\mathbf{if}\ B_j\ \mathbf{then}\ z_j := z_j - 1}\ \mathbf{fi} \\
\qquad \mathbf{od}; \\
\qquad S_i \\
\quad \mathbf{od}.
\end{array}
$$

- **In case of identical guards $B_1 \equiv \cdots \equiv B_n$, the transformation simplifies** to

$$
\begin{array}{l}
T_{fair}(S) \equiv S_0;\ z_1 :=?;\ \ldots;\ z_n :=?; \\
\quad \mathbf{do}\ \square_{i=1}^{n}\ B_i \wedge \boxed{z_i = min\ \{z_1, \ldots, z_n\}} \rightarrow \\
\qquad z_i :=?; \\
\qquad \mathbf{for\ all}\ j \in \{1, \ldots, n\} - \{i\}\ \mathbf{do} \\
\qquad\quad \boxed{z_j := z_j - 1} \\
\qquad \mathbf{od}; \\
\qquad S_i \\
\quad \mathbf{od}.
\end{array}
$$

Example

Example 12.4. The printer-user program

$$
\begin{aligned}
PU1 \equiv\ & signal := \mathbf{false};\\
& \mathbf{do}\ \neg signal \rightarrow \text{"print next line"}\\
& \square\ \ \neg signal \rightarrow signal := \mathbf{true}\\
& \mathbf{od}
\end{aligned}
$$

discussed in Section 12.1 is transformed into

$$
\begin{aligned}
T_{fair}(PU1) \equiv\ & signal := \mathbf{false};\ z_1 :=?;\ z_2 :=?;\\
& \mathbf{do}\ \neg signal \wedge z_1 \le z_2 \rightarrow\ z_1 :=?;\ z_2 := z_2 - 1;\\
& \qquad\qquad\qquad\qquad\quad \text{"print next line"}\\
& \square\ \ \neg signal \wedge z_2 \le z_1 \rightarrow\ z_2 :=?;\ z_1 := z_1 - 1;\\
& \qquad\qquad\qquad\qquad\quad signal := \mathbf{true}\\
& \mathbf{od}.
\end{aligned}
$$

With the help of the scheduling variables $\boldsymbol{z_1}$ **and** $\boldsymbol{z_2}$, the transformed program $T_{fair}(PU1)$ generates exactly the **fair computations** of the original program PU1

Transformation

Theorem 12.3. (Embedding) *For every one-level nondeterministic program S and every proper state σ*

$$\mathcal{M}_{fair}[\![S]\!](\sigma) = \mathcal{M}_{tot}[\![T_{fair}(S)]\!](\sigma) \ \mathbf{mod}\ Z,$$

where Z is the set of scheduling variables z_i used in T_{fair}.

Due to the presence of the scheduling variables $z_1, \ldots, z_n$ in $T_{fair}(S)$, the best we can prove is that the semantics $\mathcal{M}_{fair}[\![S]\!]$ and $\mathcal{M}_{tot}[\![T_{fair}(S)]\!]$ agree *modulo* $z_1, \ldots, z_n$; that is, the final states agree on all variables except $z_1, \ldots, z_n$. To express this we use the **mod** notation introduced in Section 2.3.

We say that two sets of states X and Y *agree modulo* Z, and write

$$X = Y \ \mathbf{mod}\ Z,$$

if

$$\{\sigma[Var - Z] \mid \sigma \in X\} = \{\sigma[Var - Z] \mid \sigma \in Y\}.$$

Thank You