Introduction to Refinement Checking and FDR

Andrzej Filinski
DIKU

Extreme Multiprogramming 12
January 7, 2014

Overview

- Brief introduction to refinement checking
  - An alternative alphabet model for CSP
    - Generalized concurrent composition
  - Refining specifications to implementations
    - trace refinements
    - failure refinements
- A look at the FDR2/FDR3 tool
  - From Oxford Computing Lab / Formal Systems Europe
  - Free academic version (for Unix-like platforms only)
- Not a requirement to use FDR for exam
  - It will not construct the required derivations for you
  - But may (or may not) be useful to sanity-check your results

Beware of bugs in the above code; I have only proved it correct, not tried it. – Donald E. Knuth

CSP as a machine-processable notation

- CSP can be used as foundation of a programming notation.
  - Often entails considerable syntactic and semantic adjustments.
    - Especially if embedding CSP inside existing language.
  - $\rightsquigarrow$ programming track of the course.
- Also a tool for machine-supported specification and verification of general concurrent systems.
  - hardware design, communication protocols, parallel-algorithm skeletons, ...
- Machine-readable notation (CSP_M) very close to CSP book
  - Plain-ASCII syntax: `P |~| Q` for $P \sqcap Q$, `P(x)` for P_x, etc.
  - Some (but surprisingly few) restrictions on allowable forms of process definitions.
  - Main difference: somewhat different formal treatment of alphabets and concurrent composition.

Alphabet models

- In traditional CSP (Hoare), all processes must have explicitly specified alphabets.
  - Alphabets don't usually matter for actual process behavior, *except* in concurrent composition ($\parallel$).
  - This formulation leads to particularly simple equational laws for $\parallel$.
- In alternative formulation (Roscoe), alphabets are only explicitly mentioned in concurrent composition (and a few other places, e.g. $CHAOS_A$)
  - More convenient for some purposes.
  - But muddies waters a bit, because concurrency can now by itself introduce some nondeterminism.
- Generalized concurrent composition: $P \underset{B}{\parallel} Q$.
  - Concrete syntax: `P [| B |] Q`, much nicer layout.
  - Processes synchronize (only) on events from set B.

Generalized concurrent composition

- $P \underset{B}{\parallel} Q$ like $P \parallel Q$ in Hoare CSP, but
  - Event in B can happen iff *both* P and Q are willing to participate
    - (and then both processes advance)
  - Event not in B can happen iff *either* P or Q willing to participate
    - (and then only the participating process advances)
    - If both are willing, the event happens *twice* (not necessarily consecutively).
- $P \parallel Q$ (in Hoare) corresponds to taking $B = \alpha P \cap \alpha Q$
- $P \mid\mid\mid Q$ (interleaving) corresponds to taking $B = \{\}$.
- Other choices for B do not necessarily correspond to anything simple in Hoare CSP.
- See full details in Roscoe: *The Theory and Practice of Concurrency*, Chapter 2.

Other significant differences between book and CSP$_M$ notation

- No explicit syntactic construct for enumerated choice; use general choice instead:

$$(x \to P \mid y \to Q) = (x \to P) \,\Box\, (y \to Q).$$

- Somewhat different treatment of divergence and *CHAOS*.
  - Shouldn't be an issue as long as all processes are guarded
- Allows indexed nondeterministic choice, potentially infinitary: $\bigsqcap_{x \in A} P(x)$.
- A few others, but mainly in parts of CSP we did not cover. See FDR manual
- [Live demo 1: HW1 in FDR; equivalence and deadlock checking]

Trace refinement

- Already have notion of satisfaction, P **sat** $\phi(tr)$.
  - Says that every (finite) trace tr of P satisfies logical formula ϕ.
  - ϕ expressed in "mathematics" (sets, sequences, functions, etc.).
  - Very general, but often hard to verify mechanically without substantial human assistance.
- A simpler notion of specification satisfaction is often sufficient: *trace refinement*.
  - Set of allowable traces expressed as *specification* process S.
  - Set of actual traces determined by *implementation* process I.
  - Relation $S \sqsubseteq_{\mathrm{T}} I$, "$I$ trace-refines S",

$$S \sqsubseteq_{\mathrm{T}} I \iff \mathit{traces}(I) \subseteq \mathit{traces}(S)$$

- Motivation: refining nondeterministic specification into (more) deterministic implementation
  - Picking one concrete behavior from space of allowable ones.

Trace refinement disallows patently incorrect behaviors

- **Ex:** Specification and implementations of vending machines

$$
\begin{aligned}
SVM &= coin \rightarrow ((sprite \rightarrow SVM) \sqcap (pepsi \rightarrow STOP)) \\
I1 &= coin \rightarrow sprite \rightarrow I1 \\
I2 &= coin \rightarrow pepsi \rightarrow STOP \\
I3 &= coin \rightarrow ((sprite \rightarrow I3) \,\Box\, (pepsi \rightarrow STOP)) \\
I4 &= coin \rightarrow I4 \\
I5 &= coin \rightarrow sprite \rightarrow pepsi \rightarrow STOP \\
I6 &= coin \rightarrow pepsi \rightarrow I6
\end{aligned}
$$

- Easy to check that for each of $I1$, $I2$, $I3$, have $SVM \sqsubseteq_{\mathrm{T}} I$.
- But each of $I4, I5, I6$ has at least one trace that is not a valid trace of SVM.
- [Live demo 2: Trace-refinement checking]

Trace refinement is not always enough

- Obvious problem: a deadlocked machine doesn't actively do anything wrong.

$$S = coin \rightarrow ((sprite \rightarrow S) \sqcap (pepsi \rightarrow STOP))$$

$$\begin{aligned} I1 &= coin \rightarrow sprite \rightarrow STOP \\ I2 &= STOP \\ I3 &= coin \rightarrow STOP \end{aligned}$$

All these I also satisfy $S \sqsubseteq_{\mathrm{T}} I$.

- More subtle problem: traces don't distinguish between internal and external choice (recall: $traces(P \sqcap Q) = traces(P \,\square\, Q)$.)

$$SE = coin \rightarrow ((sprite \rightarrow SE) \,\square\, (pepsi \rightarrow STOP))$$

$$\begin{aligned} I1 &= coin \rightarrow sprite \rightarrow I1 \\ I2 &= coin \rightarrow ((sprite \rightarrow I2) \sqcap (pepsi \rightarrow STOP)) \end{aligned}$$

Both I satisfy, $SE \sqsubseteq_{\mathrm{T}} I$, but neither actually allows customer to choose beverage.

Failures

- **Recall:** $\mathit{refusals}(P)$ = all sets B s.t. P *may* deadlock if environment only offers B.
- **Def.** A *failure* of a process P is a pair (s, B), s.t. $s \in \mathit{traces}(P)$ and $B \in \mathit{refusals}(P \;/\; s)$.
- A failure (despite the name) does not indicate an *error* in P.
  - Simply means that P may (possibly correctly) deny a specific sequence of requests from environment.
  - E.g., $(\langle\rangle, \{\mathit{sprite}\})$ *should* be a failure of $\mathit{VM} = \mathit{coin} \rightarrow \mathit{sprite} \rightarrow \mathit{VM}$ (do not dispense free sprites!).
- **Def.** *failure-refinement*: $S \sqsubseteq_{\mathrm{F}} I \Leftrightarrow \mathit{failures}(I) \subseteq \mathit{failures}(S)$.
- Stronger than trace-refinement: If $S \sqsubseteq_{\mathrm{F}} I$ then also $S \sqsubseteq_{\mathrm{T}} I$:
  1. Suppose $s \in \mathit{traces}(I)$
  2. then $(s, \{\}) \in \mathit{failures}(I)$ [since $\{\} \in \mathit{refusals}(P)$ always]
  3. hence $(s, \{\}) \in \mathit{failures}(S)$ [by assumption that $S \sqsubseteq_{\mathrm{F}} I$)
  4. so in particular $s \in \mathit{traces}(S)$.

Failure refinement and deadlocks

- Example

$$S = \mathit{coin} \rightarrow \mathit{sprite} \rightarrow S$$
$$I = \mathit{coin} \rightarrow \mathit{STOP}$$

- Traces:

$$\mathit{traces}(S) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{coin}, \mathit{sprite}\rangle, \langle \mathit{coin}, \mathit{sprite}, \mathit{coin}\rangle, ...\}$$
$$\mathit{traces}(I) = \{\langle\rangle, \langle \mathit{coin}\rangle\}$$

Have $\mathit{traces}(I) \subseteq \mathit{traces}(S)$, so $S \sqsubseteq_{\mathrm{T}} I$

- Failures:

$$\begin{aligned} \mathit{failures}(S) = \{ & (\langle\rangle, \{\}), (\langle\rangle, \{\mathit{sprite}\}), (\langle \mathit{coin}\rangle, \{\}), (\langle \mathit{coin}\rangle, \{\mathit{coin}\}), \\ & (\langle \mathit{coin}, \mathit{sprite}\rangle, \{\}), (\langle \mathit{coin}, \mathit{sprite}\rangle, \{\mathit{sprite}\}), \ldots\} \\ \mathit{failures}(I) = \{ & (\langle\rangle, \{\}), (\langle\rangle, \{\mathit{sprite}\}), (\langle \mathit{coin}\rangle, \{\}), (\langle \mathit{coin}\rangle, \{\mathit{coin}\}), \\ & (\langle \mathit{coin}\rangle, \{\mathit{sprite}\}), (\langle \mathit{coin}\rangle, \{\mathit{coin}, \mathit{sprite}\})\} \end{aligned}$$

Here, e.g., $(\langle \mathit{coin}\rangle, \{\mathit{sprite}\})$ in $\mathit{failures}(I)$, but not in $\mathit{failures}(S)$, so $S \not\sqsubseteq_{\mathrm{F}} I$.

Failure refinement and choices

- **Recall:** $\mathit{refusals}(P \sqcap Q) = \mathit{refusals}(P) \cup \mathit{refusals}(Q)$; $\mathit{refusals}(P \,\Box\, Q) = \mathit{refusals}(P) \cap \mathit{refusals}(Q)$.
- **Example**: $(A = \{\mathit{coin}, \mathit{sprite}, \mathit{pepsi}\})$

$$\begin{aligned} I1 &= \mathit{coin} \rightarrow \mathit{sprite} \rightarrow \mathit{STOP} \\ I2 &= \mathit{coin} \rightarrow ((\mathit{sprite} \rightarrow \mathit{STOP}) \sqcap (\mathit{pepsi} \rightarrow \mathit{STOP})) = \mathit{SI} \\ I3 &= \mathit{coin} \rightarrow ((\mathit{sprite} \rightarrow \mathit{STOP}) \,\Box\, (\mathit{pepsi} \rightarrow \mathit{STOP})) = \mathit{SE} \end{aligned}$$

$$\begin{aligned} \mathit{failures}(I1) &= \{(\langle\rangle, B) \mid B \subseteq \{s,p\}\} \cup \{(\langle c\rangle, B) \mid B \subseteq \{c,p\}\} \cup \\ &\quad \{(\langle c,s\rangle, B) \mid B \subseteq A\} \\ \mathit{failures}(I2) &= \{(\langle\rangle, B) \mid B \subseteq \{s,p\}\} \cup \{(\langle c\rangle, B) \mid B \subseteq \{c,p\}\} \cup \\ &\quad \{(\langle c\rangle, B) \mid B \subseteq \{c,s\}\} \cup \{(\langle c,x\rangle, B) \mid x \in \{s,p\}, B \subseteq A\} \\ \mathit{failures}(I3) &= \{(\langle\rangle, B) \mid B \subseteq \{s,p\}\} \cup \{(\langle c\rangle, B) \mid B \subseteq \{c\}\} \cup \\ &\quad \{(\langle c,x\rangle, B) \mid x \in \{s,p\}, B \subseteq A\} \end{aligned}$$

So for all I, have $\mathit{SI} \sqsubseteq_{\mathrm{F}} I$; but $\mathit{SE} \not\sqsubseteq_{\mathrm{F}} I1$, $\mathit{SE} \not\sqsubseteq_{\mathrm{F}} I2$.

- [Live demo 3: Failure-refinement checking]

FDR

- Under continuous development since 1990; lots of other features.
  - "Concurrency workbench" for experimental development.
- Also includes systematic treatment of *divergences*
  - Name from *Failures–Divergence Refinement*
  - Implementation may be allowed to diverge, if spec does
  - Formal treatment is a bit nastier, when recursion isn't necessarily guarded.
- Can also have parameterized processes, channel communications.
  - For small alphabets/state sets, works as expected.
  - For large or infinite, need to be careful.
    - Verification may "time out" because it needs to check a huge (or infinite) number of cases
- Details in FDR manual.

Conclusion

- No more theory lectures!
  - Last programming lecture on Friday
- Course evaluation (handled outside of Absalon now): please complete!
- Final exam: will be out on Monday.