

Verifying Properties of Processes, Data, and Time

Linking Counterexamples to High-Level Specifications

Ulrich Hobelmann

April 24, 2007

Motivation

- Software is everywhere
- Software makes important decisions
→ Software needs to be correct
- *Testing* can only prove the *presence* of bugs,
not their *absence*
→ specification, verification, model-checking

Challenges in software specification

How to express...

- concurrency and communication
- Structured data
- real-time

Specifying concurrency and communication

CSP (Communicating sequential processes)

- process naming, recursion

$$P := a \rightarrow P$$

- semantics: processes reduce to other processes:

$$a \rightarrow P \xrightarrow{a} P$$

- *STOP* can not reduce (deadlock)

Specifying structured data

Z, Object-Z

- Schemas
- Classes
- Operations

Specifying real-time behavior

Duration Calculus

- too powerful (undecidable)
- counterexample formulae (example...)
 - subclass of implementables

CSP-OZ-DC

- combination of CSP, Object-Z, and Duration Calculus
- channels for each method
- event = Object-Z method invocation

Syspect

System Specification Tool, UML modeling

- Class diagrams: classes, interfaces, dependencies
- State Machines: simple states, complex states
- Annotations: Object-Z, Duration Calculus
- Component diagrams
- export: \LaTeX markup, CSP-OZ-DC, PEAs (later)

Model-checking

- Verify certain properties of a specification
- reachability
- halting problem!

Abstraction-refinement model-checker

- states are *abstract*, i.e., not simple reachability
- Transition Constraint System

Transition Constraint System

TCS: $(st, v, init, tr, final)$

- states, variables, initial, transitions, final

Transition: $(s, d, s', d', pre, post, n)$

- states, variables, pre-, post-conditions

```
r (p (pc (source) , data (x, y) ) ,  
  p (pc (dest) , data (xP, yP) ) ,  
  [x >= 2] , [xP = x + y, yP = y + 1] , 27) .
```

ARMC behavior

- infinite loop
- program is correct
- spurious counterexample
- counterexample trace: transition list

Model-checking in Syspect

Test Formulae:

- unlike counterexample formulae: can be fulfilled
- syntax:

$$\begin{aligned}
 \textit{Phase} &:= \ell > 0 \wedge \ell \sim k \mid Ph \wedge [\varphi] \mid Ph \wedge \boxplus \mathcal{E} \\
 \textit{Trace} &:= Ph \mid \downarrow \mathcal{E} \mid \uparrow \mathcal{E} \mid T_1 ; T_2 \\
 \textit{BForm} &:= T \mid \neg F \mid F_1 \wedge F_2 \\
 \textit{Testform} &:= F \mid TF_1 ; TF_2 \mid TF_1 \wedge TF_2 \mid TF_1 \vee TF_2
 \end{aligned}$$

- first element has to be a phase; = is not allowed
- syntax ambiguous

Translation to ARMC

- CSP-OZ-DC
- Phase Event Automata
- combine PEAs
- Transition Constraint System

Phase Event Automaton

PEA: $(P, V, A, C, E, s, I, E_0)$

- phases, variables, event variables, clock variables, edges, state invariants, clock invariants, initial edges
- edge: (p, g, X, p')
- guards are terms using events, clocks, and variables from V
- can only traverse one edge at once; takes non-zero time
- parallelization: product PEA

Stuttering edges

- parallel PEAs synchronize transitions
- stuttering edge has no effect
- helps with product construction

High-Level Translation

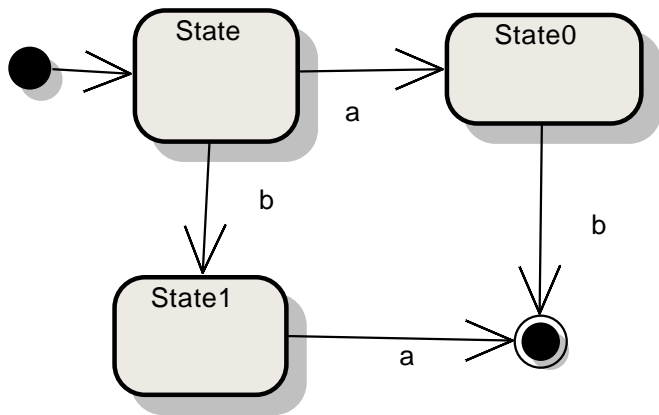
- Object-Z
- State Machines to CSP
- CSP to PEA
- Duration Calculus
- Test Formulae

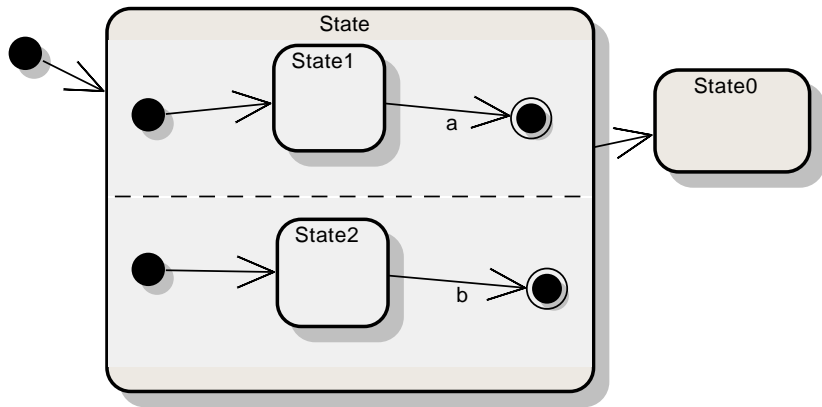
Object-Z

- two phases
- one edge per operation + *init*
- stuttering edges

State Machines

- one process equation per state
- final states: *SKIP*
- triggered transitions: $\square (e \rightarrow P_i)$
- direct transitions: $\square P_i$
- complex state: *Regions ; STOP* or *Regions ; $\square P_i$*
- regions: $R_1 ||| \dots ||| R_n$





CSP

- one phase per process
- transitions \rightarrow edges
- negate events
- no extra rule for parallel compositions needed

Duration Calculus

- separate counterexample formula into DC phases
- $\neg (\text{true}; \uparrow e; \lceil n > 3 \rceil \wedge \ell < 5; \text{true})$
- enumerate phases
- phase 0: $n' \leq 3 \vee \not\downarrow e$

Test Formulae

PEA Test Automata

- Test Formulae can be fulfilled \rightarrow final states
- model-checker can look for final state
- $(true; \uparrow e; [n > 3] \wedge \ell < 5)$
- e will transition into phase 01
- invariant $n > 3$, clock invariant

Model-checking

- program is correct \rightarrow spec can not fulfill test formulae
- counterexample trace \rightarrow transition list leading to error state
- What happened?

Translation PEAs to ARMC

- specification = list of PEAs
- parallel PEA
- Transition Constraint System can not express disjunction
- DNF

Linking ARMC Transitions to PEA edges

- give each PEA an id number i
- give each edge in each PEA an id number p_i
→ combined PEA will have the number combination Πp_i
- represented as `_pea_trans_i_` variable
- assignment conjuncted with each PEA edge

Linking ARMC Transitions to PEA edges

- PEA edge id present in all ARMC transitions
- expansion of edges with disjunction works too
- each ARMC transition has one unique PEA edge
→ has a unique product of PEA edges
- realized by parsing ARMC file

Linking PEA edges to High-Level elements

- `PEAInfo` interface
- generated by conversion algorithms
- `getName()`: describes the PEA
- `getInfoForTransition(t)`: describes the transition's high-level action

ObjectZPEAInfo

- per operation one entry: PEA edge \rightarrow operation
- one entry for initialization edge
- return operation name, enable condition, and effect
- “consume: $work = 1 \wedge fuel > 5 \wedge fuel' = fuel - 1$ ”

StateMachinePEAInfo

- two levels: PEA-CSP and CSP-SM
- PEA-CSP is easy: all processes map to one phase
→ three maps: source, target, event
- CSP-SM is harder: complex states
- StateMachineConfiguration: State **or** RegionSet

Linking processes with State Machine Configurations

- parallel processes do not persist: `getAcceptSet ()` generates new ones
- Problem: need to compare processes for equivalence
- `equals ()` is no solution: processes in different contexts need to be distinct

Linking processes with State Machine Configurations

Solution: on-demand linking

- Link all simple states and complex initial configurations
- using the event and source config, create new configuration (target)
- enter new configuration in map
- `map processes` for basic processes;
`map configs` for configurations
- “ $Par(Init2, Init1) \xrightarrow{a} Par(F2, Init1)$ ”

DCPEAInfo

- To what extent is a counterexample fulfilled?
→ split into DC phases
- DC to PEA conversion is implemented in
`Trace2PEACompiler`
- register DC phase set for every generated PEA transition
- very little modification necessary
- “In Phases: $\lceil TRUE \rceil, a ; \lceil TRUE \rceil \wedge \ell \leq 3, \lceil !n \leq 2 \rceil$ ”

Visualization

ARMC transitions, PEAIInfo objects: table

Transition	TF: abb	OZ: TestCap	SM: TestCap
89			Par (Init2, Init1)
88	In Phases: [TRUE] a ; [TRUE]	a	Par (Init2, Init1) -- a --> Par (F2, Init1)
37			

Demonstration and Questions