Z specification language

Quantifers

- ∀ Universal Quantifier, e.g. $\forall n : \mathbb{N} \bullet n^2 > n$
- ∃ Existential Quantifier, e.g. $\exists n : \mathbb{N} \bullet n > 0$

Sets

- ∈ is a predicate. Use n : N to declare a new variable n of type N
- $\{ x : X \mid P(x) \}$: All elements of X for which predicate P is true
- $a \,..\, b = \{ n : \mathbb{N} \mid a \le n \le b \}$
- ⊆ Subset : $S \subseteq T \Leftrightarrow \forall s : S \bullet s \in \mathrm{T}$
- ⊂ Proper subset : $S \subset T \Leftrightarrow S \subseteq T \wedge S \neq T$
- ***P*** Power set: $P(X)$ = set of all subsets of X
  - $S \subseteq T \Leftrightarrow S \in \mathrm{P}(T)$
  - $\#X = k \Leftrightarrow \#P(X) = 2^k$
- ∪ Union : Suppose $\mathrm{S}, T: P(X),\ S \cup T \Leftrightarrow \{ x : X \mid x \in \mathrm{S} \vee x \in \mathrm{T} \}$
- ∩ Intersection : Suppose $S, T: P(X),\ S \cap T \Leftrightarrow \{ x : X \mid x \in S \wedge x \in T \}$
- − Difference: Suppose $S, P(X),\ S - T \Leftrightarrow \{ x : X \mid x \in S \wedge x \notin T \}$
- × Cartesian Product: Set of all ordered pairs (s, t) where $s \in S \wedge t \in T$
  - The i-th value (1-indexed) of a tuple t is referred to as $t.i$

Relations

- ↔ Relation $A \leftrightarrow B$ = Set of all possible relations from A to B
  - $A \leftrightarrow B = P(A \times B)$; Suppose $R: A \leftrightarrow B, R \subseteq A \times B$
  - $(\mathrm{a}, \mathrm{b}) \in R \Leftrightarrow a \to b \in R \Leftrightarrow a\ R\ b$
- **dom**(R) : Domain of $R = \{ a : A \mid \exists \mathrm{b} : B \bullet a\ R\ b \}$
- **ran**(R) : Range of $R = \{ b : B \mid \exists a : A \bullet a\ R\ b \}$
- Suppose $R : A \leftrightarrow B$ and $S \subseteq A$ and $T \subseteq B$
  - ◁ Domain restriction: $S \lhd R \Leftrightarrow \{ (a, b) : R \mid a \in S \}$
    - $S \lhd R \subseteq \mathrm{R}\,; S \lhd R \in A \leftrightarrow B$
    - E.g. Supose has_sibling: People ↔ People, then female ◁ has_sibiling = is_sister_of
  - ▷ Range restriction: $R \rhd T \Leftrightarrow \{ (a, b) : R \mid b \in T \}$
    - $R \rhd T \subseteq R\,; R \rhd T \in A \leftrightarrow B$
    - E.g. Suppose has_sibling: People ↔ People, then has_sibiling ▷ female = has_sister
  - ⩤ Domain subtraction: $S ⩤ R \Leftrightarrow \{ (a, b) : R \mid a \notin S \}$
    - $S ⩤ R = R - (S \lhd R) = (A - S) \lhd R$
  - ⩥ Range subtraction: $R ⩥ T \Leftrightarrow \{ (a, b) : R \mid a \notin T \}$
    - $R ⩥ T = R - (R \rhd T) = R \rhd (B - T)$
- Relational Image: Suppose R : $A \leftrightarrow B$ and $S \subseteq A$, then $R(|S|) = \{ b : B \mid \exists a : S \bullet a\ R\ b \} \subseteq B$
  - E.g. has_sibling(| male |) = { people who have a brother }
  - E.g. divides(|{8, 9}|) = { numbers divisible by 8 or 9 }
- Relational Image: Suppose R : $A \leftrightarrow B$ and $S \subseteq A$, then $R(|S|) = \{ b : B \mid \exists a : S \bullet a\ R\ b \} \subseteq B$
- Inverse: Suppose $R : A \leftrightarrow B$, then $R^{-1} = \{ (b, a) : B \times \mathrm{A} \mid (a, b) \in R \}$
- ⨾ Relational composition: Suppose $R : A \leftrightarrow B$ and $S : B \leftrightarrow C$, then $R ⨾ \mathrm{S} = \{ (a, c) : A \times C \mid \exists b \in \mathrm{B} \bullet (a\ R\ b) \wedge (b\ S\ c) \}$
  - $R^k = R ⨾ \mathrm{R} ⨾ \mathrm{R} \ldots ⨾ \mathrm{R}$, k times

Functions

- ⇸ **Partial** function: $f : A \nrightarrow B \subseteq A \times B$, maps each $a \in A$ to **at most one** $b \in B$
- **dom**(f) : Domain of $f = \{ a : A \mid \exists b : B \bullet (a, b) \in f \}$
- **ran**(f) : Range of $f = \{ b : B \mid \exists a : A \bullet (a, b) \in f \}$
- → **Total** function: $f : A \to B$, where dom(f) = A
- ⊕ Function overriding: Suppose $f, g : A \nrightarrow B$
  - $f \oplus g = (dom(g) ⩤ f) \cup g\,; f \oplus g \in A \nrightarrow B$
  - $\mathrm{dom}(f \oplus g) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$
  - $\forall a : \mathrm{dom}(g) \bullet (f \oplus g)(a) = g(a)$
  - $\forall a : \mathrm{dom}(g) - \mathrm{dom}(f) \bullet (f \oplus g)(a) = f(a)$

Sequences

- Sequence s : seq A is a special type of function $s : \mathbb{N} \nrightarrow A$
  - seq₁ A = set of all non-empty sequences of A
- ⁀ Concatenation, e.g. $\langle a, b \rangle ⁀ \langle b, c \rangle = \langle a, b, b, c \rangle$
- head : $\mathrm{seq}_1\ A \to A$, returns the first item in the sequence
- tail : $\mathrm{seq}_1\ A \to \mathrm{seq}\ A$, returns the sequence without the head, i.e. $\forall s : \mathrm{seq}_1\ A \bullet \langle \mathrm{head}(s) \rangle ⁀ \mathrm{tail}(s) = s$
- $s(i)$ = i-th (1-indexed) element of sequence s
- Filter ↾, e.g $\langle a, b, c, d, e, d, c, b, a \rangle \upharpoonright \{a, d\} = \langle a, d, d, a \rangle$

State Schema

- State schema: 'snapshot' of a system
- State variables declared and types on top; predicates listed at the bottom
- An instance = assignment of (valid) values to state variables

Operation Schema

- An instance of a state schema to another instance, modelling how the system can change
- Pre-state variables: unprimed; post-state variables: primed
- "?" denotes an input; "!" denotes an output

```
| - Join ------------------------------------------------
| items, items' : seq MSG
| msg? : MSG
|-------------------------------------------------------
| #items ≤ max
| #items' ≤ max
| #items < max
| items' = items⁀<msg?>
| ------------------------------------------------------
```

Schema Inclusion

Schema inclusion to simplify the syntax:

- Δ required to use post state variables

```
| - Buffer ----------------------      | - Join -------------------------
| items : seq MSG                      | Δ Buffer
|--------------------------------      | msg? : MSG
| #items ≤ max                         |--------------------------------
| -------------------------------      | #items < max
                                       | items' = items⁀<msg?>
| - Δ Buffer --------------------      |--------------------------------
| items, items' : seq MSG
|--------------------------------
| #items ≤ max
| #items' ≤ max
| -------------------------------
```

Merging Schemas/Schema conjunction

- C ≙ A ∧ B
  - Declaration part: Union of declarations from A and B
  - Predicate part: Conjunction of predicates from A and B

```
| - A ----------    | - B ----------    | - C -------------    | - C -------------
| x : T1            | y : T2            | x : T1               | A
| y : T2            | z : T3            | y : T2               | B
|---------------    |---------------    | z : T3               |------------------
| P(x, y)           | P(y, z)           |------------------
| --------------    | --------------    | P(x, y) ∧ P(y, z)
                                        | -----------------
```

Schema disjunction ≙

- C ≙ A ∨ B
  - Declaration part: Union of declarations from A and B
  - Predicate part: Disjunction of predicates from A and B

```
| - A ----------    | - B ----------    | - C -------------    | - C -------------
| x : T1            | y : T2            | x : T1               | A
| y : T2            | z : T3            | y : T2               | B
|---------------    |---------------    | z : T3               |------------------
| P(x, y)           | P(y, z)           |------------------    | P(x, y) ∨ P(y, z)
| --------------    | --------------    | P(x, y) ∨ P(y, z)    | -----------------
                                        | -----------------
```

Schema composition ⨾

- Applies to operation schemas to indicate an operation immediately followed by another (atomically)
- C ≙ A ⨾ B
  - Declaration part: Union of declarations from A and B
  - Predicate part:
    - Predicates from A (Pre-state of C = pre-state of A)
    - Pre-state of B = post-state of A and post-state of C = post-state of B (can be represented by ∃state'' : X • post-state of A ∧ post-state of B)

Communicating Sequential Process (CSP)

- Process: defined by its behaviour
- Events: A process engages in events; e.g. events for a chocolate vending machine are (i) coin – insert a coin and (ii) choc – extract a chocolate
- Alphabet: Set of events a process can engage in e.g. Alphabet of the chocolate vending machine = { coin, choc }
- Trace: finite sequence of events e.g. <coin, choc>
- STOP_A: process with alphabet A that can do nothing, i.e. traces(STOP_A) = {<>}
- SKIP: A process that can only terminate, i.e. traces(SKIP) = {<✓>}
- CLOCK: process with αCLOCK={tick}, i.e. traces(CLOCK) = tick^n
- Event Prefix: A process which may participate in event a then act according to process description P is denoted as $a \to P$

CSP Primitives

- Sequential Composition (P;Q)
  - Acts as P until P terminates by communicating ✓ and then proceeds to act as Q
- Parallel Composition (P **|[X]|** Q, or P **||** Q)

- No event from set X may occur unless enabled by both P and Q
  - When events from X occur, they occur in both P and Q in parallel; events not from X occur in P or Q separately
  - E.g.(a -> P) |[a]| (c -> (a -> Q)): Event c must happen before event a for event a to run in parallel
  - If P || Q, we can infer X = αP ∩ αQ
  - Laws: suppose a ∈ (αP – αQ), b ∈ (αQ – αP), { c, d } ⊆ (αP ∩ αQ),
    - P || Q = Q || P
    - P || (Q || R) = (P || Q) || R
    - P || $STOP_{\alpha P}$ = $STOP_{\alpha P}$ (deadlock)
    - (c -> P) || (c -> Q) = c -> (P || Q)
    - (c -> P) || (d -> Q) = STOP if c ≠ d
    - (a -> P) || (c -> Q) = a -> (P || c -> Q))
    - (c -> P) || (b -> Q) = b -> ((c -> P || Q))
    - (a -> P) || (b -> Q) = a -> (P || (b -> Q)) [] b -> ((a -> P) || Q)
- Interleaving (P **|||** Q)
  - P and Q execute concurrently without any synchronisations
- Choice ((a -> P) **[]** (b -> Q))
  - Either event a or event b occurs. If event a occurs, then the process acts as P afterwards, otherwise is acts as Q
- Channel event
  - c**!**n: A process writes value n to the tail of channel c's buffer
  - c**?**n: A process reads a value from the head of the channel c's buffer to a local variable n
  - c**.**n: Channel output and its matching channel input are engaged together by two processes
  - Each channel has a capacity. If a channel is full, only a read event (c?n) can occur. If a channel is empty, only a write event (c!n) can occur
  - Synchronous channel: a channel with buffer size of 0
    - c!n and c?n **must happen at the same time** (denoted as c.n)
- Interrupt (P ∇ e -> Q)
  - Process behaves as P until first occurrence of event e, then the control passes to Q
- <op>x:<dom>@P(x) = P(x_1) <op> P(x_2) … <op> P(x_n) for each x_k in dom

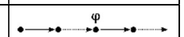
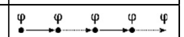
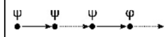
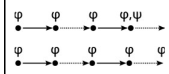
<u>Process Analysis Toolkit (PAT)</u>

- **Linear Temporal Logic (LTL) assertion**
  - var name = [1, 2, 3] or var name[n]; values defaulted to 0
  - Initialization with range:
    - e.g. var name: {0..} = …, var name: {1..9} = […]
  - Fast array initialization:
    - e.g. Suppose #define N 2. Then [1(2), 3..6, 7(N*2), 12..10] = [1, 2, 3, 4, 5, 6, 7, 7, 7, 7, 12, 11, 10]
  - Multi-dimensional arrays are internally converted into a 1d array
- Macro: **#define** name pred, e.g. #define goal x == 0; often used with #assert to define system property
- Event compound forms: x.exp1.exp2, to uniquely identify events with same event name but different states (e.g. process params, global vars)

- Statement block inside events: Run piece of code when the event is engaged, e.g. P = { C# code } -> SKIP
  - If accompanied with another event, eg P = a{ C# code } -> SKIP, it means that the C# code runs atomically with event a
- **Conditional choice**: e.g. P = if (exp) { a -> SKIP } else { b -> SKIP }
- **Guarded** process: Guarded processes only execute when their guard condition is satisfied e.g. P = [x == 1] a -> STOP [] [x != 1] b -> STOP
- **Atomic** process: Use atomic keyword to associate higher priority with process e.g. P = atomic { a -> … }
  - An atomic process will execute it's next event before any non-atomic processes
- **Assertions**: **#assert** process name
  - Deadlock-freeness: #assert P <u>deadlockfree</u>
  - Reachability: #assert P <u>reache</u>s cond [with exp]

Linear Temporal Logic (LTL) assertion

- A LTL assertion is true iif **<u>every trace</u>** of the system satisfies F:
  - #assert P |= F, where F = LTL formula = event, proposition or any of the operators below (F = event, just check initial event)

Op	Explanation	Diagram
X φ, X	ne**X**t: φ must hold at the next state	
F φ, <>	**F**inally: φ must eventually hold	
G φ, []	**G**lobally: φ must always hold	
ψ **U** φ, U	**U**ntil: ψ must hold at least until φ becomes true	
ψ **R** φ, R	**R**elease: φ must hold until and including but not beyond the first position in which ψ is true, or forever if such position does not exist	

Timed Process Definition

- Wait[t] <u>process</u>
  - Process Wait[t]; P delays the starting time of P by t time units
- Timeout (P timeout[t] Q)
  - Engage Q if no visible event has occurred in process P before t time units have elapsed
- Timed Interrupt (P interrupt[t] Q)
  - Process behaves as P until t time units elapse and then switches to Q
- Deadline[t]
  - Process P deadline[t] must terminate within t time units
- Within[t]
  - In Process P within[t], the first visible event in P must be engaged within t time units