

Chapter 14: Formal Specification and Enactment

Service-Oriented Computing: Semantics, Processes, Agents
Munindar P. Singh and Michael N. Huhns, Wiley, 2005

Formal Specification and Enactment

Declarative representations based on logic

- Contrast with procedural flow specifications
 - Branch and join primitives
 - Central execution engine
- Capture the essence of what is required
 - Minimally constrain the execution of services
 - Accommodate greater efficiencies
 - Accommodate better handling of exceptions and opportunities
 - Support naturally distributed enactment

Temporal Logic

Logic of time

- Based on *significant events*: events that matter to others
 - Start: s
 - Commit: c
 - Abort: a or rather \bar{c}
- Declaratively specify *dependencies*, i.e., constraints
- Maximum flexibility bring about the right events to satisfy the stated constraints
- Would support a high-level reasoner

Example Dependencies

- If T_1 starts then T_2 starts: $\bar{s}_1 \vee s_2$
- If air ticket transaction starts then hotel booking transaction starts: $\bar{s}_A \vee s_H$
- If order (O) is canceled and payment (P) is made then refund (R) is initiated:
 $c_O \vee \bar{s}_P \vee \bar{c}_P \vee s_R$
- If refund is initiated then payment must *previously* have been made: $\bar{s}_R \vee c_P \cdot s_R$

Notice events are the atoms, \bar{e} is the complement of e , and the dot operator \cdot indicates temporal order

Specification Syntax

- The center dot (\cdot) orders events
- *Complementation* means hard opposite: commit versus abort
 - Used in specifications
- *Negation* means soft opposite: commit versus not commit
 - *Not* used in specifications

$L_1. I \longrightarrow dep \mid dep \wedge I \ll\text{interleaving}\gg$

$L_2. dep \longrightarrow seq \mid seq \vee dep \ll\text{choice}\gg$

$L_3. seq \longrightarrow bool \mid event \mid event \cdot event$
 $\ll\text{ordering}\gg$

$L_4. bool \longrightarrow 0 \mid \top$

Specification Semantics

Identify the desirable “runs” or computations

- Universe consists of *legal* runs:
 - Event instances and their complements are mutually exclusive
 - Event instances don't repeat (transaction identifiers can ensure uniqueness)

$$M_1. \tau \models e \text{ iff } (\exists i : \tau_i = e)$$

$$M_2. \tau \models I_1 \vee I_2 \text{ iff } \tau \models I_1 \text{ or } \tau \models I_2$$

$$M_3. \tau \models I_1 \wedge I_2 \text{ iff } \tau \models I_1 \text{ and } \tau \models I_2$$

$$M_4. \tau \models I_1 \cdot I_2 \text{ iff } (\exists i : \tau_{[0,i]} \models I_1 \text{ and } \tau_{[i+1,|\tau|]} \models I_2)$$

Example Coordination Relationships

- $D_{<} = \bar{e} \vee \bar{f} \vee e \cdot f$
 - If both e and f occur, then e precedes f
 - If e and f occur on τ , neither \bar{e} nor \bar{f} can occur on τ . So τ must satisfy $e \cdot f$, which means that an initial part of τ satisfies e and the remainder satisfies f
- $(\bar{e} \vee f \vee g) \wedge (\bar{g} \vee e) \wedge (\bar{g} \vee \bar{f})$
 - If e happens and f does not, then and only then do g
 - Typical with data updates, where g restores consistency (potentially) violated by the success of e and the failure of f

Enactment

Control execution of tasks to meet the specifications

- Allow, delay, deny, or trigger events to satisfy dependencies stated
 - A realized run is in each of their denotations
- System state = the runs that are allowed
 - Initially, given by the stated dependencies
 - Narrows down as events occur
- Key requirements
 - Maximal set of allowed runs (flexibility)
 - Compute symbolically and modularly

Residuation

$$E_1. 0/e \doteq 0$$

$$E_2. \top/e \doteq \top$$

$$E_3. (D \wedge F)/e \doteq (D/e \wedge F/e)$$

$$E_4. (D \vee F)/e \doteq (D/e \vee F/e)$$

$$E_5. e/e \doteq \top$$

$$E_6. \bar{e}/e \doteq 0$$

$$E_7. (e \cdot f)/e \doteq f$$

$$E_8. (\bar{e} \cdot f)/e \doteq 0$$

$$E_9. (d \cdot e)/e \doteq 0$$

$$E_{10}. (d \cdot \bar{e})/e \doteq 0$$

$$E_{11}. (d \cdot f)/e \doteq d \cdot f$$

$$E_{12}. d/e \doteq d$$

The above rules apply if we swap e and \bar{e}

Example of Residuation

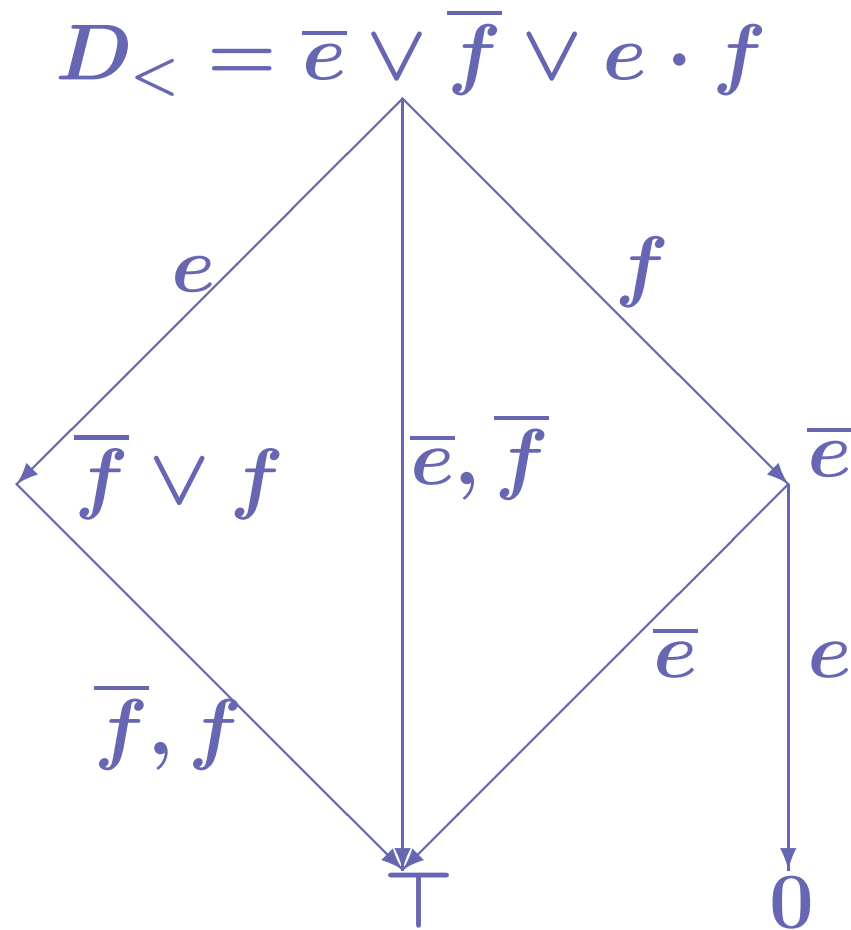


Figure 1: Scheduler states and transitions for $D_{<}$

Distributed Enactment

- Constrain autonomy based only on dependencies
 - Local decisions
- Place a *guard* on each event
 - When true, the event can safely happen
 - Modified as relevant events occur (messages arrive)
- Challenges
 - Representing them
 - Reasoning with them in a distributed manner

Guard Syntax

Enables stating whether an event can occur *now*

L₅. $T \longrightarrow conj \mid conj \wedge T$

L₆. $conj \longrightarrow disj \mid disj \vee conj$

L₇. $disj \longrightarrow bool \mid \square seq \mid \diamond seq \mid \neg event$

- Events are *stable* or durable
- $\square e$ means e has occurred
- $\diamond e$ means e has occurred or will occur eventually
- $\neg e$ means e has *not yet* occurred

Guard Semantics

- Universe consists of maximal runs (either an event or its complement occurs)

$$M_5. \quad u \models_k E \text{ iff } u \models_{0,k} E$$

$$M_6. \quad u \models_{i,k} f \text{ iff } (\exists j : i \leq j \leq k \text{ and } u_j = f)$$

$$M_7. \quad u \models_{i,k} E \vee F \text{ iff } u \models_{i,k} E \text{ or } u \models_{i,k} F$$

$$M_8. \quad u \models_{i,k} E \wedge F \text{ iff } u \models_{i,k} E \text{ and } u \models_{i,k} F$$

$$M_9. \quad u \models_{i,k} E \cdot F \text{ iff } (\exists j : i \leq j \leq k \text{ and } u \models_{i,j} E \text{ and } u \models_{j+1,k} F)$$

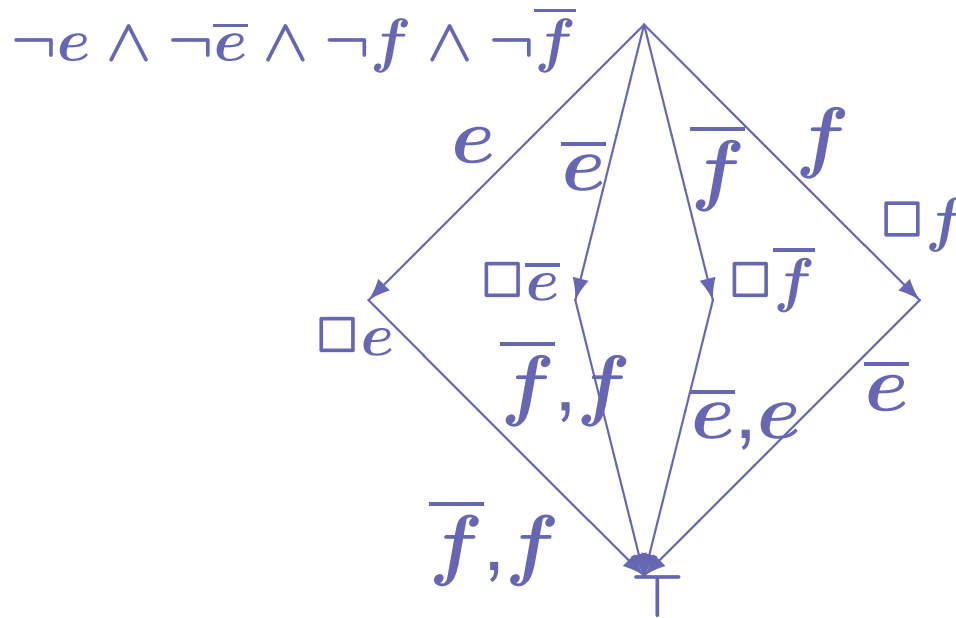
$$M_{10}. \quad u \models_{i,k} \top$$

$$M_{11}. \quad u \models_{i,k} \neg E \text{ iff } u \not\models_{i,k} E$$

$$M_{12}. \quad u \models_{i,k} \Box E \text{ iff } (\forall j : k \leq j \Rightarrow u \models_{i,j} E)$$

$$M_{13}. \quad u \models_{i,k} \Diamond E \text{ iff } (\exists j : k \leq j \text{ and } u \models_{i,j} E)$$

Guards for $D_{<} = \bar{e} \vee \bar{f} \vee e \cdot f$



- $G_b(D_{<}, e) = (\neg f \wedge \neg \bar{f} \wedge \diamond(\bar{f} \vee f)) \vee (\square \bar{f} \wedge \top) = [(\neg f \wedge \neg \bar{f}) \vee \square \bar{f}] = \neg f \vee \square \bar{f} = \neg f$
- $G_b(D_{<}, \bar{e}) = \top$
- $G_b(D_{<}, \bar{f}) = \top$
- $G_b(D_{<}, f) = (\neg e \wedge \neg \bar{e} \wedge \diamond \bar{e}) \vee \square e \vee \square \bar{e} \cong \diamond \bar{e} \vee \square e$

Scheduling with Guards: Example

- If e is attempted first
 - $G(e) = \top$: e executes and notifies
 - Notification $\square e$ changes
 $G(f) = \diamond \bar{e} \vee \square e = \top$, enabling f
- If f is attempted first
 - $G(f) = (\diamond \bar{e} \vee \square e) \neq \top$, so it waits
 - Notification of $\square \bar{e}$ or $\square e$ changes $G(f)$ to \top , thus enabling f
- $G(\bar{e}) = \top$ and $G(\bar{f}) = \top$, so they can happen any time

Motivations for Formalization

- Proving correctness when
 - Guards are created by compiling the dependencies
 - Guards are preprocessed
 - Events are executed and guards updated
- Justifying improvements in efficiency
 - Simplifying guards prior to execution
 - Updating guards incrementally
 - Skipping some steps

Formalization Sketch: 1

- Evaluation strategy: a function that captures
 - Evolution of guards
 - Execution of events
- An evaluation strategy *generates* a run u if
 - For each event e that occurs on u ,
 - u satisfies e 's current guard due to the strategy
 - At the index *preceding* e 's occurrence
- Generation is more abstract than execution:
 - A true guard may involve \diamond expressions

Formalization Sketch: 2

- Begin with trivial strategy
 - Easily correct, but useless
- Replace with better strategies
 - Symbolically calculate guards from dependencies
 - Safely discard certain terms
 - Process messages symbolically

Symbolically Calculating Guards

- $G(0, e) \triangleq 0$
- $G(\top, e) \triangleq \top$
- $G(D \vee F, e) \triangleq G(D, e) \vee G(F, e)$
- $G(D \wedge F, e) \triangleq G(D, e) \wedge G(F, e)$
- $G(e, e) \triangleq \top$
- $G(\bar{e}, e) \triangleq 0$
- $G(d \cdot e, e) \triangleq \Box d$
- $G(d \cdot \bar{e}, e) \triangleq 0$
- $G(e \cdot f, e) \triangleq \neg f \wedge \Diamond f$
- $G(\bar{e} \cdot f, e) \triangleq 0$
- $G(d, e) \triangleq \Diamond d$
- $G(d \cdot f, e) \triangleq \Diamond(d \cdot f)$

The above rules apply if we swap e and \bar{e}

Calculating Guards: Example

For $D_{<} = \bar{e} \vee \bar{f} \vee e \cdot f$:

- $G(D_{<}, e) = (\diamond \bar{f} \vee (\neg f \wedge \diamond f)) \cong \neg f$
- $G(D_{<}, \bar{e}) = \top$
- $G(D_{<}, f) = \diamond \bar{e} \vee \square e$
- $G(D_{<}, \bar{f}) = \top$

Assimilating Messages

Old: G	Message: M	New: $G \div M$
$G_1 \vee G_2$	M	$G_1 \div M \vee G_2 \div M$
$G_1 \wedge G_2$	M	$G_1 \div M \wedge G_2 \div M$
$\Box e$	$\Box e$	\top
$\Diamond e$	$\Box e$ or $\Diamond e$	\top
$\Box \bar{e}$ or $\Diamond \bar{e}$	$\Box e$ or $\Diamond e$	0
$\Diamond(e \cdot f)$	$\Box e$	$\Diamond f$
$\Diamond(e \cdot f)$	$\Diamond(e \cdot f)$	\top
$\Diamond(e \cdot f)$	$\Box(f \cdot e)$ or $\Diamond(f \cdot e)$ or $\Box \bar{e}_i$ or $\Diamond \bar{e}_i$	0
$\neg e$	$\Box e$	0
$\neg \bar{e}$	$\Box e$ or $\Diamond e$	\top
G	M	G , otherwise

Summary

- Generic approach to describe processes and extended transactions
 - Hides low-level details
 - Combines declarative specifications and operational decision procedures
- Directions
 - Refining methodologies, based on assessment of scenarios
 - Accommodating richer heuristics for distributed evaluations