Events, permissions, and obligations

... and their refinement

Stephan Merz

(with thanks to Frédéric Cuppens and Dominique Méry)

INRIA Lorraine & LORIA, Nancy
Context

- Project on information security
  - access control
  - information flow

- Static system model: who may / must (not) do what?
  - identify organizations, roles, activities, contexts, etc
  - assign permissions / user rights and responsibilities

- Integration with dynamic system model
  - (temporal) properties of behaviors
  - stepwise refinement preserving “deontic” properties
System: Bank

Constants: Client, Loan, maxDebt

Variables: loans, clt, due, rate

Invariant: \( \land \) loans \( \subseteq \) Loan

\( \land \) clt \( \in \) [loans \( \rightarrow \) Client] \( \land \) due \( \in \) [loans \( \rightarrow \) \( \mathbb{N} \)] \( \land \) rate \( \in \) [loans \( \rightarrow \) \( \mathbb{N} \)]

\( \land \) \( \forall \) c \( \in \) Client : \( \sum \) {due(ll) : ll \( \in \) loans \( \land \) clt(ll) = c} \( \leq \) maxDebt

Initial: loans = \( \emptyset \)

Event: newLoan(c : Client, l : Loan, sum : \( \mathbb{N} \), dur : \( \mathbb{N} \)) \( \equiv \)

\( \land \) l \( \notin \) loans \( \land \) sum + \( \sum \) {due(ll) : ll \( \in \) loans \( \land \) clt(ll) = c} \( \leq \) maxDebt

\( \land \) loans' = loans \( \cup \) \{l\} \( \land \) clt' = clt \( \cup \) \{l \( \mapsto \) c\}

\( \land \) due' = due \( \cup \) \{l \( \mapsto \) sum\} \( \land \) rate' = rate \( \cup \) \{l \( \mapsto \) sum/dur\}

Event: payRate(l : Loan) \( \equiv \)

\( \land \) l \( \in \) loans

\( \land \) due' = due \( \oplus \) \{l \( \mapsto \) due(l) - rate(l)\}

\( \land \) clt' = clt \( \land \) rate' = rate
Properties (safety)

- Stable predicates

\[ P \land e(x) \Rightarrow P' \quad \text{for all events } e \]

\[ \text{stable } P \]

- Invariants

\[ Init \Rightarrow P \quad \text{stable } P \]

\[ \text{inv } P \]

\[ \text{inv } P \]

\[ \text{always } Q \]

- Proof obligation: \text{inv } Inv

for the declared system invariant \text{Inv}
Adding fairness conditions

- Event systems describe what can occur
- Fairness ensures that events do occur eventually

**event** \( \text{payRate}(l : \text{Loan}) \equiv \)
\[
\land l \in \text{loans} \\
\land \text{due}' = \text{due} \oplus \{l \mapsto \text{due}(l) - \text{rate}(l)\} \\
\land \text{clt}' = \text{clt} \land \text{rate}' = \text{rate} \\
\text{fairness} \ l \in \text{loans} \land \text{due}(l) > 0
\]

This talk: weak fairness
- if condition persists, event must eventually occur
- condition may be stronger than guard
Properties (liveness)

- $F \rightsquigarrow G$: every $F$ will be followed by $G$

Verification rules

\[
\begin{align*}
P \land a(x) \land \neg e(t) & \Rightarrow P' \lor Q' \quad \text{for all events } a \\
P & \Rightarrow \text{fair}_e(t) \\
\hline
P & \rightsquigarrow Q \lor (P \land e(t))
\end{align*}
\]

\[
\begin{align*}
P \land e(t) & \Rightarrow Q' \\
P \land e(t) & \rightsquigarrow Q \\
F & \Rightarrow G \\
\hline
\text{inv } I & \\
I \land F & \rightsquigarrow G \lor \neg I
\end{align*}
\]

\[
\begin{align*}
\forall x \in S : F(x) & \Rightarrow G \lor (\exists y \in S : y \prec x \land F(y)) \\
(\exists x \in S : F(x)) & \rightsquigarrow G \\
(x \text{ not free in } G)
\end{align*}
\]

\[
\begin{align*}
F & \rightsquigarrow G \\
G & \rightsquigarrow H \\
\hline
F & \rightsquigarrow H
\end{align*}
\]

\[
\begin{align*}
F & \rightsquigarrow H \\
G & \rightsquigarrow H \\
\hline
F \lor G & \rightsquigarrow H
\end{align*}
\]

\[
\begin{align*}
(\exists x : F) & \Rightarrow (\exists x : G)
\end{align*}
\]
Refinement: intuition

- add detail to model, but preserve properties
  - different data representation, related by linking invariant $J$
  - refine grain of atomicity of events
- map concrete events to abstract ones (maybe stutter)
Refinement: intuition

- add detail to model, but preserve properties
  - different data representation, related by linking invariant $J$
  - refine grain of atomicity of events

- map concrete events to abstract ones (maybe stutter)

$$e_{abs} \lor \tau$$

$$J$$

$$e_{ref}$$

$$J$$
Refinement: intuition

- add detail to model, but preserve properties
  - different data representation, related by linking invariant $J$
  - refine grain of atomicity of events

- map concrete events to abstract ones (maybe stutter)
  
  \[ e_{abs} \lor \tau \]

common extra conditions:
- eventually perform abstract events
- relative deadlock freedom

here: preserve fairness
Refinement : proof obligations

- simulation of initial condition

Init_{ref} \Rightarrow \exists \text{var}_{abs} : \text{Init}_{abs} \land J

- step simulation (possibly stuttering)

er(t) \land J \Rightarrow \exists u, \text{var}'_{abs} : \text{ea}(u) \land J' \quad (er \text{ refines } ea)

er(t) \land J \Rightarrow \exists \text{var}'_{abs} : \text{var}'_{abs} = \text{var}_{abs} \land J' \quad (er \text{ new event})

- refinement of fairness constraints \quad (er_1, \ldots, er_n \text{ refine } ea)

\text{true} \iff \forall \lnot (\exists \text{var}'_{abs} : \text{fair}_{ea}(u) \land J)

\quad \lor (\exists t_1 : er_1(t_1)) \lor \ldots \lor (\exists t_n : er_n(t_n))
Refinement: properties

- simulation of traces

  for every trace of the concrete system $Ref$
  there is a corresponding trace of the abstract system $Abs$
Refinement: properties

- Simulation of traces

  for every trace of the concrete system $Ref$
  there is a corresponding trace of the abstract system $Abs$

- Preservation of properties modulo linking invariant

  \[
  \begin{align*}
  Abs & \models \text{stable } P \Rightarrow Ref \models \text{stable } (\exists var_{abs} : P \land J) \\
  Abs & \models \text{inv } P \Rightarrow Ref \models \text{inv } \bar{P} \\
  Abs & \models \text{always } P \Rightarrow Ref \models \text{always } \bar{P} \\
  Abs & \models F \leadsto G \Rightarrow Ref \models \bar{F} \leadsto \bar{G}
  \end{align*}
  \]
Permissions & obligations

Who may/must do what, under what circumstances?

- static model of entities and activities (Or-BAC)
  represented as constants and events

- specify permissions / rights and obligations
  add corresponding predicates to event definitions
Permissions & obligations

- Who may/must do what, under what circumstances?
  - static model of entities and activities (Or-BAC)
    represented as constants and events
  - specify permissions / rights and obligations
    add corresponding predicates to event definitions

- Relation with system model?
  - verify “deontic” properties of model
  - and adapt refinement relation
Representing permissions

- Extend description of events

  event newLoan(c : Client, l : Loan, sum : \mathbb{N}, dur : \mathbb{N}) \equiv \ldots
  
  permission l \notin \text{loans} \land \text{risk}(c, sum) \in \{\text{low, medium}\}
  
  interdiction \text{risk}(c, sum) = \text{high}

- Verification conditions ensure that annotations hold

  - invariant and permission implies guard
  
  - invariant and interdiction implies negation of guard
    
    always \neg(e(t) \land intd_e(t))
Representing obligations

- Similarly add obligation predicates

  \[\text{event} \quad \text{payRate}(l : \text{Loan}) \equiv \ldots\]
  \[\text{oiligation} \quad l \in \text{loans} \land \text{due}(l) > 0\]

- Temporal interpretation

  \begin{align*}
  \text{strict obligation} & \quad \text{obl}_e(t) \leadsto e(t) \\
  \text{weak obligation} & \quad \text{obl}_e(t) \leadsto \lnot\text{obl}_e(t) \lor e(t) \quad \text{[this is just weak fairness!]} 
  \end{align*}

- We know how to establish these properties
Refinement: preserving properties

- Obligations & interdictions: nothing to prove
  - expressed as (linear-time) properties of traces
  - hence preserved by refinement

Permissions: more problematic
- Refinement does not preserve branching behavior
- What should be preserved across non-atomic refinement?
- Refined event won't be executable whenever abstract one is
Refinement: preserving properties

- Obligations & interdictions: nothing to prove
  - expressed as (linear-time) properties of traces
  - hence preserved by refinement

- Permissions: more problematic
  - refinement does not preserve branching behavior
  - what should be preserved across non-atomic refinement??
  - refined event won’t be executable whenever abstract one is
Refinement of permissions

- Idea: refine abstract-level permission
  - by a concrete-level permission (to start a branch)
  - and a concrete-level obligation (to simulate the event)
Refinement of permissions

- Idea: refine abstract-level permission
  - by a concrete-level permission (to start a branch)
  - and a concrete-level obligation (to simulate the event)
Refinement of permissions

Idea: refine abstract-level permission

- by a concrete-level permission (to start a branch)
- and a concrete-level obligation (to simulate the event)

Formalization (assume ea refined by er₁, . . . , erₙ)

identify “initial events” ei₁, . . . , eiₘ of refined model where

\[ \overline{perm_{ea}} \Rightarrow perm_{ei₁} \lor \ldots \lor perm_{eiₘ} \quad \text{and} \]
\[ ei_j \sim \neg perm_{ea} \lor er₁ \lor \ldots \lor erₙ \]
Example

- Refining event `newLoan`

  \[
  \text{event } \text{askLoan}(c : \text{Client}, l : \text{Loan}, \text{sum} : \mathbb{N}, \text{dur} : \mathbb{N}) \equiv \ldots
  \]

  \[
  \text{permission } l \notin \text{loans}
  \]

  \[
  \text{event } \text{approveLoan}(l : \text{Loan}, e : \text{Employee}) \equiv \ldots
  \]

  \[
  \text{permission}
  \]

  \[
  \land l \in \text{non}\_\text{approved}
  \]

  \[
  \land \lor \text{risk}(\text{clt}(l), \text{due}(l)) = \text{low} \land \text{rank}(e) \geq \text{Clerk}
  \]

  \[
  \lor \text{risk}(\text{clt}(l), \text{due}(l)) = \text{medium} \land \text{rank}(e) \geq \text{Manager}
  \]

  \[
  \text{interdiction } \text{risk}(\text{clt}(l), \text{due}(l)) = \text{high}
  \]

  \[
  \text{obligation } l \in \text{non}\_\text{approved} \land \text{risk}(\text{clt}(l), \text{due}(l)) \in \{\text{low, medium}\}
  \]
Observations

- Refinement of permissions is transitive
  - introduce explicit permission on “initial” event
  - has to be taken into account when refining further

- Weak interpretation of obligations adequate
  - consider client applying for two loans concurrently
  - no obligation to approve them both
Summing up

- slight extension of event systems
- represent permissions, interdictions, obligations
- property-preserving refinement rules
  - non-atomic refinement of events
  - inheritance of linear-time properties
  - basic branching-time properties: enabledness + liveness
- future work: controllers for security policies