Mechanizing the Proofs of the Mondex Challenge with KIV

Gerhard Schellhorn
Holger Grandy, Dominik Haneberg, Wolfgang Reif

University of Augsburg

Overview

- Part 1: The full Mondex case study in KIV
  - Context of Work: Go! Card project
  - The interactive theorem prover KIV
  - Translating Z specifications to algebraic specifications
  - Formal verification of the Mondex refinement in KIV
  - Discussion of results and future work
- Part 2 (optional): Abstract State Machines for Mondex
  - Abstract state machines and ASM refinement
  - A specification of Mondex using ASMs

Context of Work

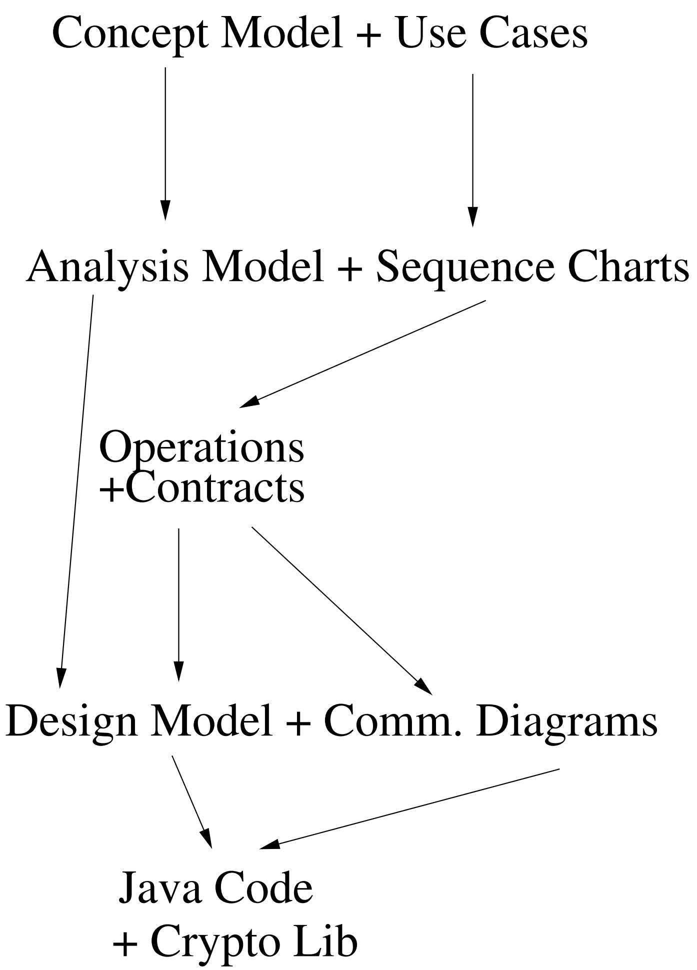
Topic of Go! Card Project (supported by DFG):
Security critical E-Commerce applications such as

- Electronic Purses (Mondex, Copy Cards, Mensa Cards)
- Ordering cinema tickets with a cell phone
- Getting on-board railway tickets using a PDA
- Getting on-line tickets for railway via WWW

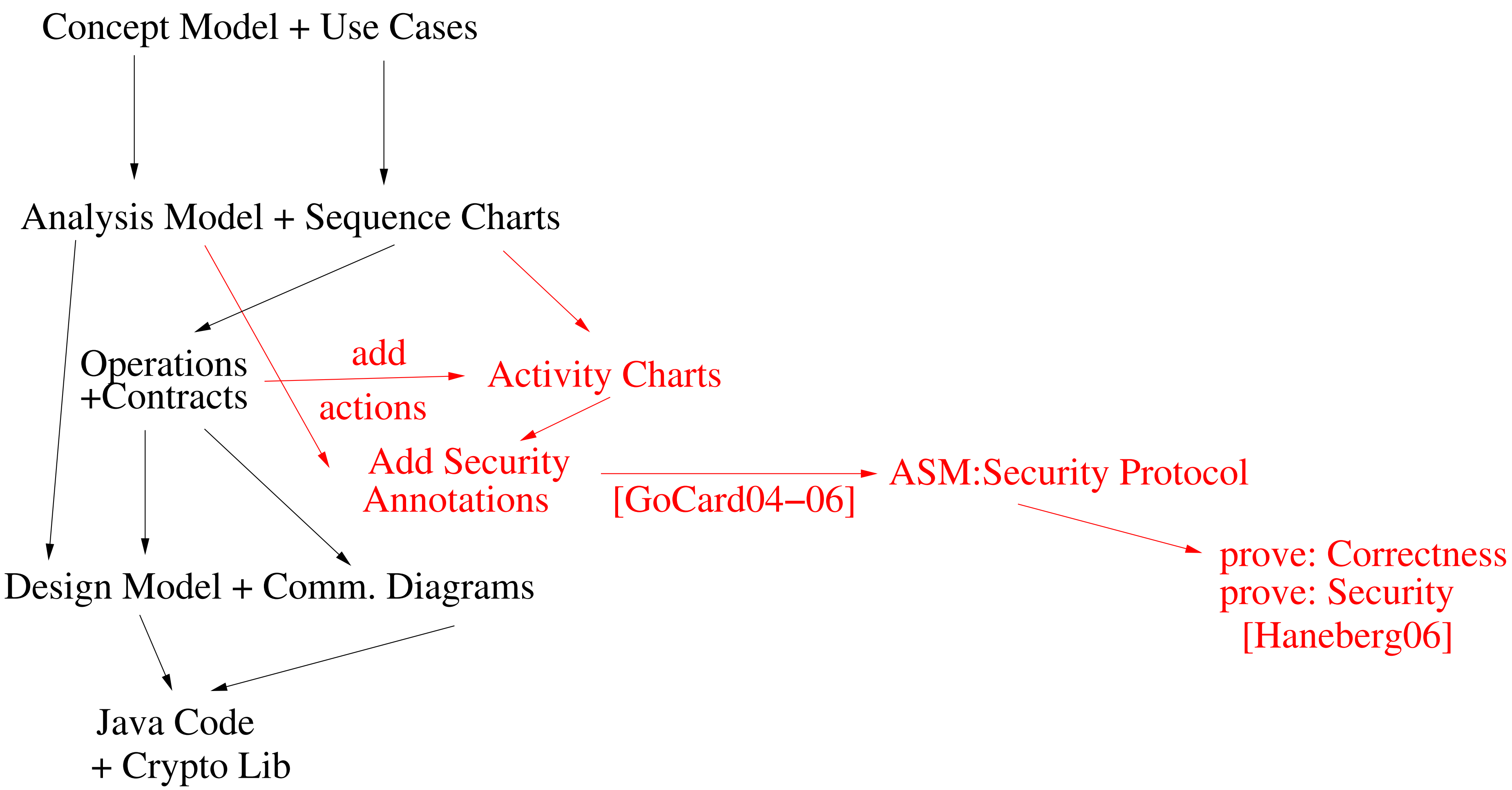
Goal: Develop an approach that integrates

- Classical software engineering using UML (and Java)
- Formal Security Analysis using theorem proving

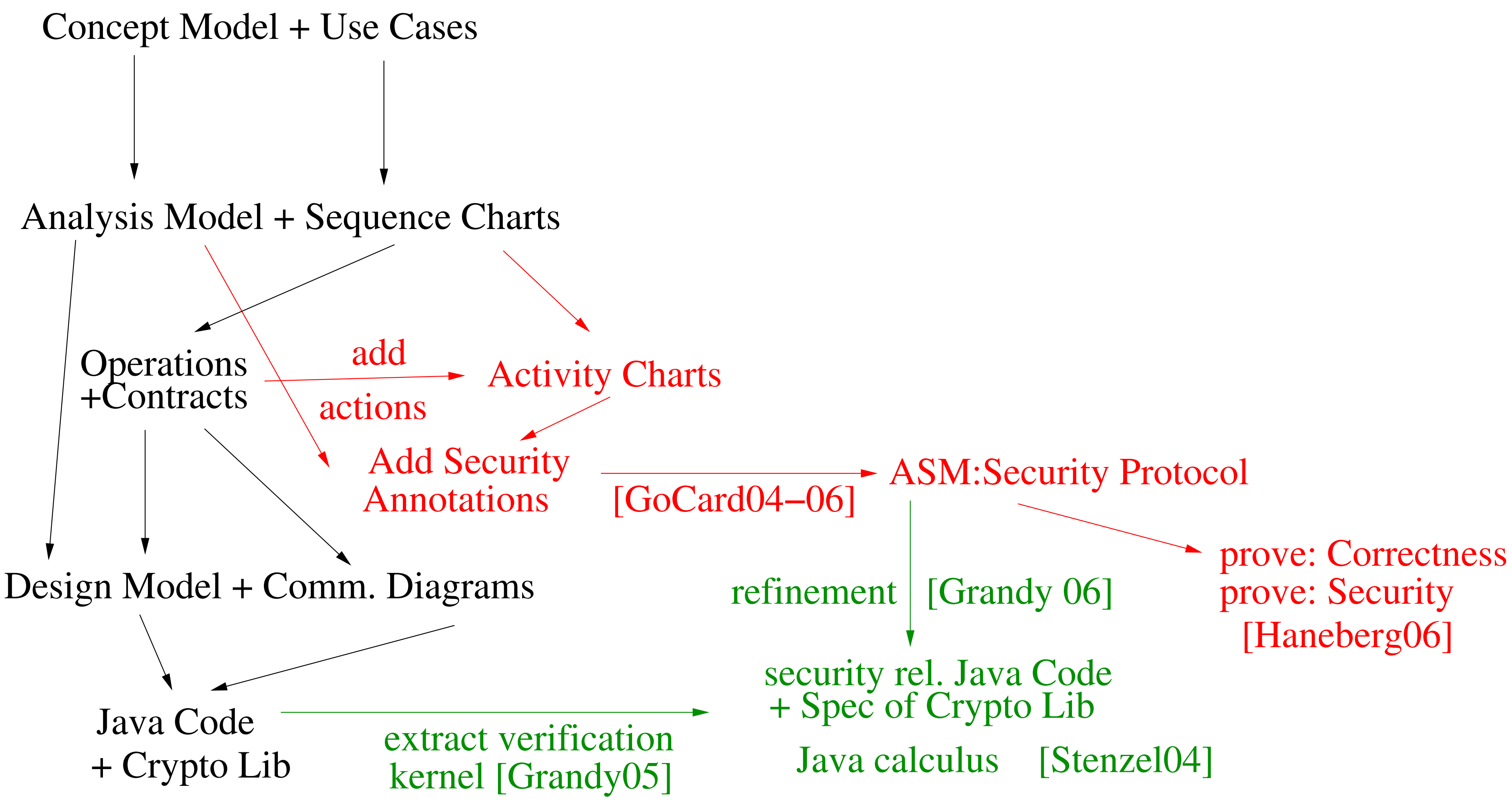
SW Engineering using UML (e.g. [Larman])



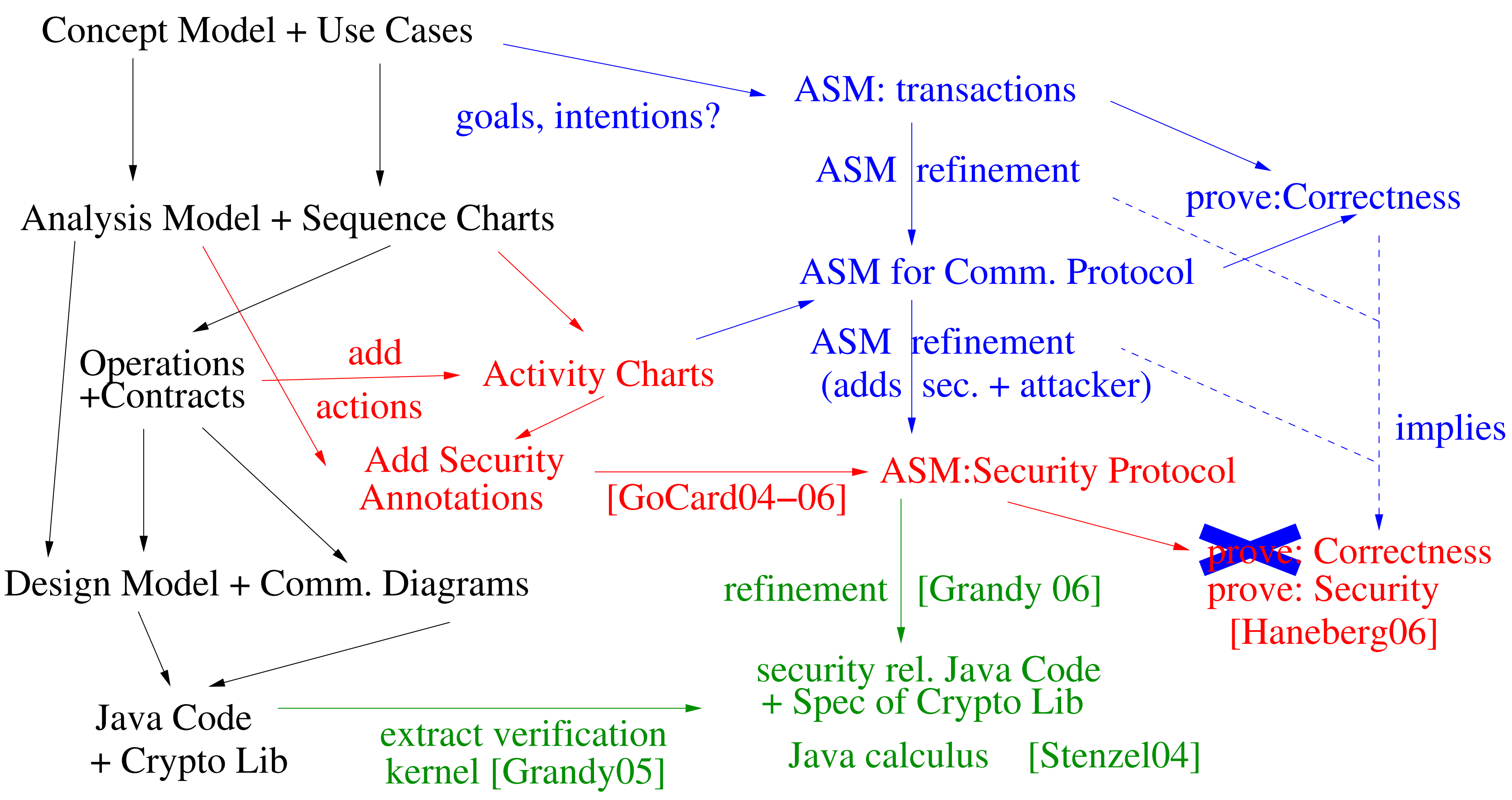
The ProSecCo approach - Security Protocols



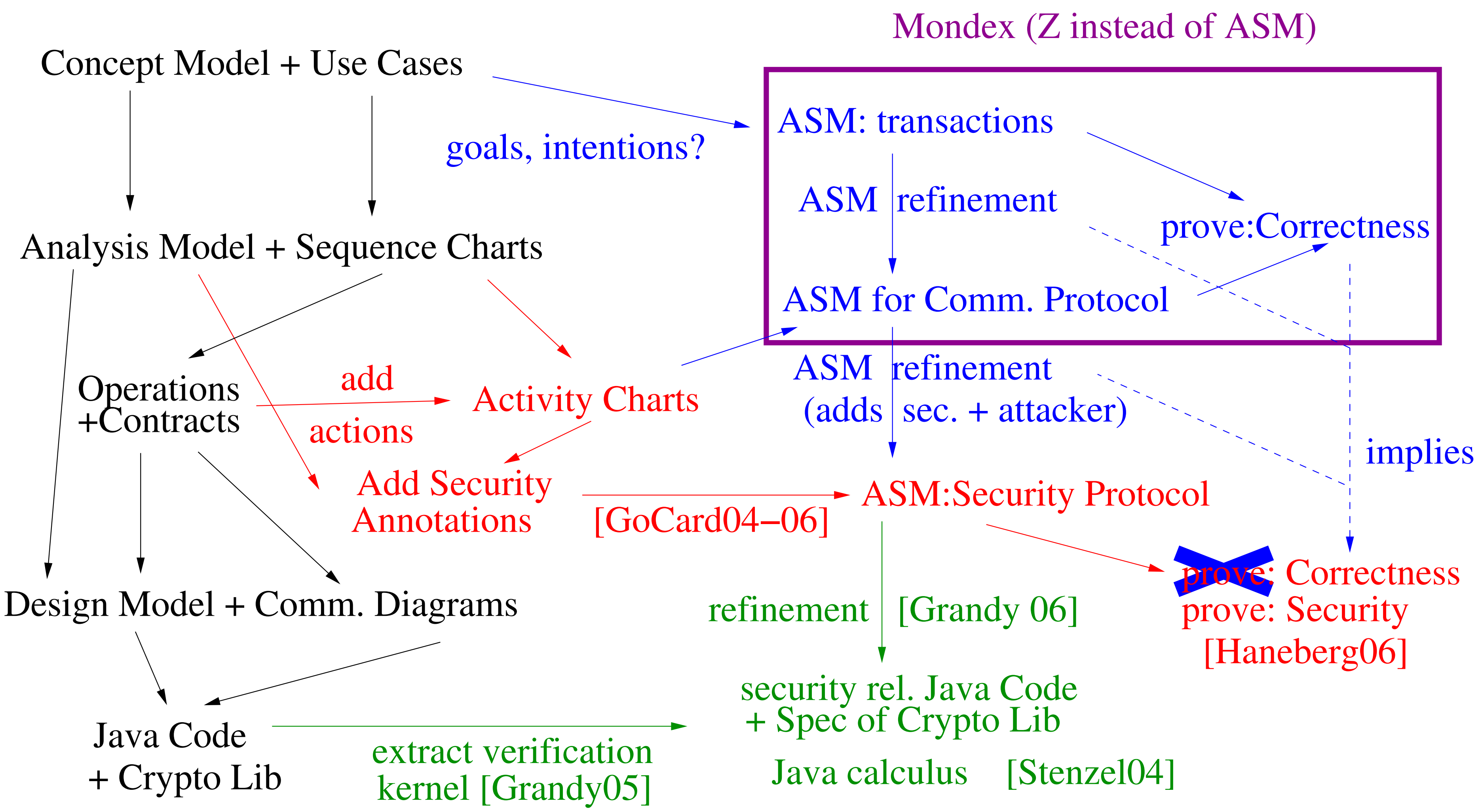
The ProSecCo approach - Refine to Java



The ProSecCo approach - Abstraction



The ProSecCo approach - Mondex



Mondex Purses

- Smartcards (www.mondex.com) that implement an electronic wallet
- Money transfer with (all of ? all with the specified protocol?)
  - smart card reader with two slots
  - over telephone
  - with two smart card readers over internet
- how is transfer authenticated ?:
  - just inserting card
  - giving a PIN
- famous for being the first product that got an ITSEC security level E6
- E6 is the highest level, requires formal methods

Mondex Purses: The Original Work

- 2 specifications in Z:
  - abstract level: transactions for money transfer
  - concrete level: communication protocol
- verification of correctness on abstract level:
  no money lost/generated
- development of a data refinement theory suitable for the case study [Cooper,Stepney,Woodcock02]
- correspondence proof between abstract and concrete level using two data refinements [Stepney,Cooper,Woodcock00]
- very detailed proofs on paper
- Formal Verification was recently proposed as “Grand Challenge 6”

Solving the Challenge: My Work

- Formal specification of the data refinement theory in KIV
  Improvement: use invariants together with backward simulation
- Mechanized exactly the backward simulation proofs of Mondex in KIV
  (improvement: use one instead of two data refinements)

  Claims:

  * Doing proofs of this size by hand is never 100% correct
  * Machine proofs can be done in reasonable time

- Definition of an alternative formalisations using ASMs

  Claims:

  * ASMs are simple and easy to understand
    (esp. for people not familiar with formal methods)
  * ASMs are easier to verify (mostly future work)

KIV - History

- Project started in 1985 (Reif, Heisel, Stephan) in Karlsruhe as a theorem prover construction project
- KIV = Karlsruhe Interactive Verifier
- Design Decisions:
  - Built-in data structure of Proof Trees for sequent calculus
  - Use Harel's Dynamic Logic (DL)
    $\Rightarrow$ Express total correctness, program inclusion (refinement)
  - Use a Graphical Interface to communicate with the user (now in Java, then programmed in motif . . .)

KIV - Higher-Order Dynamic Logic (DL)

- sorts S, types $\mathsf{T} = \mathsf{S} \mid \mathsf{T}^+ \to \mathsf{T}$
- Operations OP, variables X (for any type T)
- sorts bool, nat and their operations are predefined
- expressions $\mathsf{E} = \mathsf{OP} \mid \mathsf{X} \mid \mathsf{E}\,(\mathsf{E}^+) \mid \lambda\ \mathsf{X}^+.\ \mathsf{E} \mid \forall\ \mathsf{X}^+.\ \mathsf{E}_{\mathsf{bool}} \mid \exists\ \mathsf{X}^+.\ \mathsf{E}_{\mathsf{bool}} \mid$
  $[\alpha]\ \mathsf{E}_{\mathsf{bool}} \mid \langle\alpha\rangle\ \mathsf{E}_{\mathsf{bool}} \mid \langle\!|\alpha|\!\rangle\ \mathsf{E}_{\mathsf{bool}}$
- In wp-calculus: $[\alpha]\ \varphi \equiv$ wlp(α, φ), $\langle\!|\alpha|\!\rangle\ \varphi \equiv$ wp(α, φ)
- $\langle\alpha\rangle\ \varphi \equiv \neg\ [\alpha]\ \neg\ \varphi$
  "there is a terminating run of α such that φ holds at the end"
- Hoare-Calculus is a sublogic: $\{\varphi\}\ \alpha\ \{\psi\} \equiv \varphi \to [\alpha]\ \psi$
- program inclusion with $\underline{x}$ = vars(α):

$$\langle\alpha\rangle\ \underline{x} = \underline{x}_0 \to \langle\beta\rangle\ \underline{x} = \underline{x}_0$$

KIV - Abstract Programs

- abstract sequential programs α with
  - parallel assignment $\underline{x} := \underline{t}$
  - α; β, **if**, **while**, **let**
  - **choose** x **with** φ **in** α
  - α **or** β (Dijkstra's choice)
  - Pascal-like procedures (globally defined)
- Semantics of programs (as in [deBakker80]):
  strict relations over states (= valuations) + $\perp$ for nontermination

Extensions of the logic (not used in the Mondex case study):

- Java programs instead of abstract programs [Stenzel04]
- Temporal logic over statecharts/interleaved programs [Balser05]

Structured Specifications in KIV

- basic specifications = higher-order DL theories (signature + axioms + procedure declarations)
- loose semantics (same as in Z)
- free data types to abbreviate standard axioms
- structured algebraic specifications (similar to CASL) with union, enrichment, renaming, parameters, actualization and instantiation
- instantiation (similar to theory interpretation [Farmer94]): allows
  * to map sorts to tuples (e.g. abstract_state $\Rightarrow$ balance, lost)
  * to generate proof obligations (e.g. those of data refinement)
  * Restrict-Identify possible (refinement of algebraic data types)

Specification structure displayed as a development graph

Theorem Proving in KIV

- Based on sequent calculus
- Using proof trees to keep track of proof structure
- (very efficiently implemented) simplification with rewrite rules (Mondex: 1200 rules, most of them from standard data types from library)
- configurable heuristics for automation
- Symbolic Execution of programs:
  (iteratively compute the strongest postcondition
  for the first statement of a program)
- Patterns define situations, where to apply certain proof rules

Mondex in KIV - Overview

- Formalize the Data Refinement Theory of Mondex as in [Cooper,Stepney,Woodcock02]:
  * Step 1: Definition of Refinement [Hoare,He86]
  * Step 2: Forward and Backward Simulation [Hoare,He86]
  * Step 3: Contract Approach [Woodcock,Davies96]
  * Step 4: Mondex refinement theory:
    backward simulation with elementwise related input/output lists
  * Step 5: Mondex refinements

Each step is formalized as a specification that instantiates the previous step. Steps 1-3 already done for [Schellhorn05]. Steps 3 and 4 were done with additional invariants.

Improving Data Refinement Theory (1)

Mondex consists of 2 refinements:

- First Refinement: Backward simulation assuming concrete state restricted by an invariant (“between level”)
- Second Refinement: Forward Simulation to prove the invariant with the same operations, except losing messages from ether

Question: Can this be solved with one refinement?

If yes, only one set of proof obligations has to be proved

Improving Data Refinement Theory (2)

- Losing messages already on “between” level is easy: modify invariant to hold for a “full ether”, which has not lost messages, current ether is a subset
- Using an invariant when operations are total as in Mondex is ok: restrict the concrete state space to be only the states where the invariant holds
- The answer is trivially yes for forward simulations (and the contract approach)
- Difficult question: Is an invariant always admissible for backward simulation and the contract approach?
- Relevant in general for iterated refinements

Improving Data Refinement Theory (3)

Theorem (Backward Refinement, Contract Approach with Invariants) Backward Simulation is compatible with using invariants AINV and CINV for both the abstract and concrete level. (formal definitions are in the technical report).

Proof (in KIV) As in the proof that reduces the backward simulation conditions of the contract approach to those of Hoare, by using a state space augmented with $\perp$. The backward simulation T must be augmented to $\mathring{\mathrm{T}} = (\mathrm{T} \rhd \mathrm{AINV}) \cup (\{\mathrm{CS}_\perp \setminus \mathrm{CINV}\} \times \mathrm{AS}_\perp)$ instead of $\mathring{\mathrm{T}} = \mathrm{T} \cup (\{\perp\} \times \mathrm{AS}_\perp)$.

Improving Data Refinement Theory (4)

Theorem (Mondex Refinement) Invariants can be added to the conditions of Mondex (backward simulation) refinement. An additional proof obligation is needed: Totality of the refinement relation that maps concrete to abstract inputs is required.

Proof (in KIV) Very similar to [Cooper,Stepney,Woodcock02], except the relation empty[A,B] $\subseteq$ seq[A] $\times$ seq[B] must be defined as

$$\text{empty}[A,B] := \text{seq}[A] \times \{[]\}$$

instead of

$$\text{empty}[A,B] := \{[]\} \times \{[]\}$$

The Mondex Case Study: From Z to KIV (1)

- Z has built-in set theory: Use isomororphism between sets and their characteristic function (e.g. sets of messages)
- Predefined Finiteness in Z: $\{x : \mathbb{N} \bullet p(x)\} \in \mathbb{F}\mathbb{N}$:
  Use data type finset of finite sets from library:
  $\exists$ sn : finset. $\forall$ x. x $\in$ sn $\leftrightarrow$ p(x)
- Z has partial functions (represented as sets of pairs):
  * use total function + domain predicate (authentic)
  * alternatives: use $\perp$-element in range, use representation
- Z has a promotion: Use abbreviations, Parameters for procedures, or expand (operations in Mondex)

Translating Z schemas

Z schemas are used for various purposes:

- to define tuples and other basic types:
  use algebraic types (e.g. PayDetails).
- to define (new) functions and predicates:
  use specifications and enrichment (e.g. loggedFrom)
- to define invariants: must be extracted to a global invariant
- main use: to define operations with pre- and postcondition

Operations Schemas in Z vs. Programs in KIV

- Operations in KIV:
  - option: purely algrebraic definition of $\mathsf{OP} : \mathsf{state} \times \mathsf{state} \rightarrow \mathsf{bool}$
  - more natural and efficient: use programs + axiom:
    $\mathsf{OP}(\mathsf{s}, \mathsf{s}') \leftrightarrow \langle \mathsf{OP\#}(\mathsf{s}) \rangle\ \mathsf{s} = \mathsf{s}'$
- **choose** can encode pre-postcondition specification:
  if $\mathsf{pre}(\mathsf{x})$ **then choose** x' **with** $\mathsf{post}(\mathsf{x}, \mathsf{x}')$ **in** $\mathsf{x} := \mathsf{x}'$ **else chaos**
- programs are more expressive (no need for embedding with $\perp$):
  contract approach = **chaos**, behavioral approach = **abort**
- Schema composition becomes program structure:
  $(\mathsf{ABORT} \fatsemi \mathsf{STARTFROM} \vee \mathsf{IGNORE})(\mathsf{s}, \mathsf{s}')$
  $\Rightarrow \langle \mathsf{ABORT\#};\ \mathsf{STARTFROM\#}$ **or** $\mathsf{IGNORE\#} \rangle\ \mathsf{s} = \mathsf{s}'$
- Programs have explicit (and more) structure
  $\Rightarrow$ better automation with symbolic execution

Mondex in KIV: Summary

- Specify the relevant part of the built-in set theory of Z
- Expand promotion
- Use programs instead of schema composition
- Simplify a few technical details:
  (e.g. states **eaFrom** and **eaTo** can be merged to **idle**)
- Apart from technical differences, the formalisation of Mondex is exactly as in [Stepney,Cooper,Woodcock00]
- Verification problem is exactly the same

Understanding the Verification Problem

- Understand how the main protocol works
  $\Rightarrow$ Write an Abstract State Machine (ASM)
- Understand the core problem: Atomic transactions are split into small protocol steps which may be interleaved
- Extract the invariant (careful: distributed in 3 places)
- Type in the simulation relation (just copy)
- Understand the role of sets maybelost, chosenlost, definitelylost and how they change
  $\Rightarrow$ Write down how they change in the protocol steps
  $\Rightarrow$ Prove invariance and simulation for the ASMs
- I didn't work through the proofs and lemmas in detail

The Mondex Case Study: Results of Verification

- Two big proofs for invariance and simulation for the ASM (no archiving, elementary steps) 1839 proof steps / 372 interactions
- Data Refinement Proofs: 3108 proof steps / 623 interactions
- 30% due to adding archiving, 20 % other proof obl.
  50 % due to nonelem. steps, data refinement
- 2 small corrections for the invariant were necessary for the main protocol
  - P-3 and P-4 must get the same constraints as P-2
  - authentic(pdAuth(receiver).to/from) is needed in P-3/P-4
- 2 small corrections for archiving:
  - empty sets of PayDetails must be avoided in exLogResult and exLogClr messages, since hash is injective on nonempty sets only
  - In exLogResult messages, the purse names must be authentic (not carefully checked)

The Mondex Case Study: Summary of Effort

- 1 week to get familiar with the case study and to set up the ASMs
- 1 week to do the 2 main proofs for the ASMs
- 1 week to specify, verify and generalize Mondex refinement theory
- 1 week to prove the Mondex case study the theories
  Result: nearly identical ASM refinement proofs
  $\Rightarrow$ ASM spec looses nothing essential compared to Z
- 3 days to verify the protocol for archiving
  (1 small additional problem in the invariant)

Alltogether: 1 person month to verify the full Mondex protocol

Jim Woodcock: 1.5 pages of spec/proof per day, ca. 10-11 PM.

(referee of FM 06: 500 person days over 18 months?)

Verification Effort: Be careful!

Beware: Setting up the original specification and getting invariants/simulations right is surely much more effort!

Beware: Effort depends much on expertise on theorem proving in general and in KIV in particular

Nevertheless: Formal verification of this case study can be done with KIV (and I think with other provers too) in reasonable time

Estimation: For a student who attended our courses in predicate logic and in theorem proving with KIV the verification of this case study should be a nice master (or diploma) thesis.

Mondex Proofs: Can they be automated?

Proof with data refinement:

- Half of the KIV interactions could be eliminated defining additional simplifier rules
- A significant rest of the interactions is creative
(e.g. how do the relevant sets maybelost, chosenlost, definitelylost change in protocol steps) cannot be automated by any tool
- Also: Adding automation is only helpful if one already knows invariant and simulation
- A much more significant improvement would be to generate (even parts) of the correct invariant and simulation
- Personal opinion: Current techniques are far from generating such complex, irregular invariants automatically

Mondex Refinement:
Can it be proved automatically? (1)

- Current proof uses data refinement and backward simulation
  $\Rightarrow$ needs complex (irregular) invariant
- Mondex is an instance of nonatomic refinement:
  1 transaction refined by n protocol steps
  $\Rightarrow$ Use a more “intelligent” refinement notion
- General m:n diagrams $\Rightarrow$ ASM refinement
- Situation is complicated by interleaving diagrams
  $\Rightarrow$ Specialize ASM refinement so that it can deal with it:
  * Coupled Refinement [Derrick,Wehrheim03]:
    Proved to be a specialisation of ASM refinement [Schellhorn05]
  * DLX Case Study [Börger, Mazzanti 98]:
    Pipelining is similar to interleaving
  * Use program invariants to avoid talking about intermediate states:
    some promising experience in a similar case study on copy cards ([Haneberg06])

Mondex Refinement: Can it be proved automatically? (2)

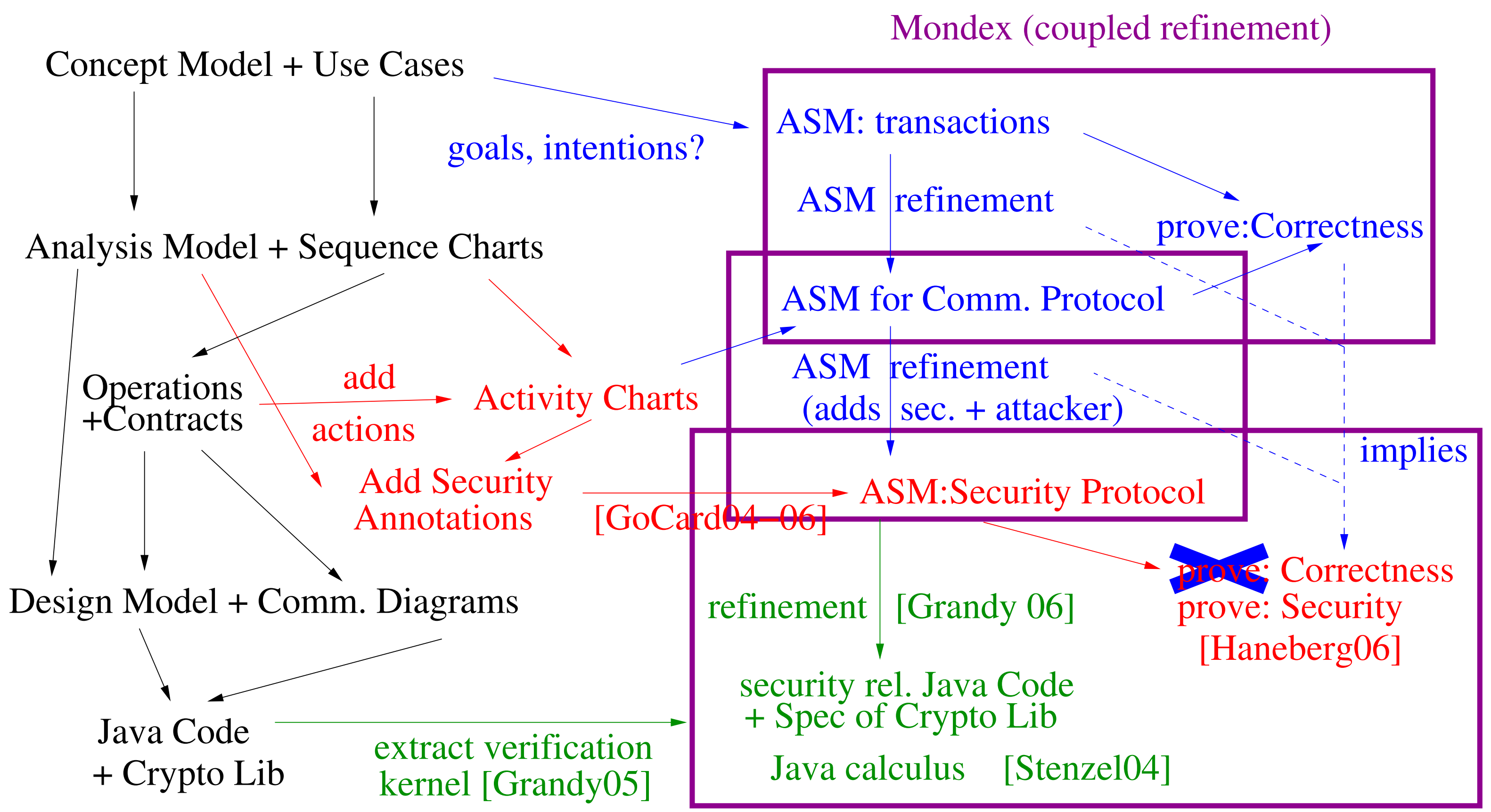
Future work and claim: Proving the refinement to be a specific kind of ASM refinement should be much easier and require a much simpler (systematically defined) invariant than proving data refinement.

In KIV: Proofs and definition of invariants/simulations will not be fully automatic.

Open question: maybe with specialized tools/techniques the “semi-finiteness” of the protocol can be exploited to get fully automatic proofs for the resulting proof obligations.

Mondex Purses: Other Future Work

- Mondex case study assumes Security rather than proving it
- Additional problem: Java Code uses data structures (byte arrays) with limited size (current master thesis)



Mondex Purses: Summary

- Intuitive Formalisation as an ASM
- The additional effort needed for machine-checked proofs is not that high
- There is still room for improvement
- Mondex is a very good challenge.
  - It can be used to compare specification styles
  - Benchmark for interactive theorem provers
    (how does it scale up for non-toy examples?)
  - to evaluate/improve definitions of refinement
  - to evaluate proof techniques and automation
  - many additional unsolved problems:
    - adding new purses on the fly
    - security using cryptography
    - infinite vs. finite data types
    - no dynamic objects on Javacards
- Full details (techn. report + all specs + all proofs) on
  http://www.informatik.uni-augsburg.de/swt/projects/mondex.html

Overview

- Part 1: The full Mondex case study in KIV
  * Context of Work: Go! Card project
  * The interactive theorem prover KIV
  * Translating Z specifications to algebraic specifications
  * Formal verification of the Mondex refinement in KIV
  * Discussion of results and future work
- Part 2 : ASMs for Mondex
  * Abstract State Machines and ASM refinement
  * A specification of Mondex using ASMs

Abstract State Machines (ASMs)

Informally: Abstract State Machines are Automata with a rule to modify some abstract state (formal definitions in [Börger03]).

- Algorithms are transition systems with
  * a set of states S
  * initial states I
  * a transition relation $\rho \subseteq$ S $\times$ S
- No state can be more general than an algebra $\mathcal{A}$ (over some first-order signature Σ)
- Therefore: Describe states by a signature, initial states any way you like (KIV: by an algebraic specifications)
- Theoretical result (ASM thesis): Under simple assumptions (invariance under isomorphism, bounded exploration etc.) any transition relation can be expressed as an ASM rule
  $\Rightarrow$ any algorithm can be easily expressed as an ASM

ASM Rules - Syntax

- function update: $f(\underline{t}) := t'$
  - * dynamic functions get updated
  - * “dynamic constants” are program variables.
  - * static functions model operations on data types (+,*,length)
- parallel execution: R_1
  R_2
- sequential execution: R_1 **seq** R_2
- conditional: **if** ε **then** R_1 **else** R_2
- indeterministic choice: **choose** x **with** $\varphi(x)$ **in** $R(x)$
- parallel execution: **forall** x **with** $\varphi(x)$ **do** $R(x)$
- local variables **let** $x = t$ **in** $R(x)$
- extensions: macros, recursive definitions + calls etc.

ASM Rules - Semantics

Semantics: Compute a (finite or infinite) set of updates $(\mathsf{f}, \underline{\mathsf{a}}) \leftarrow \mathsf{b}$:

$[\![\mathsf{f}(\underline{\mathsf{t}}) := \mathsf{t}']\!] \Rightarrow \{(\mathsf{f}, \underline{\mathsf{t}}_{\mathcal{A}}) \leftarrow \mathsf{t}'_{\mathcal{A}}\}$

$[\![\begin{smallmatrix}\mathsf{R}\\\mathsf{S}\end{smallmatrix}]\!] \Rightarrow [\![\mathsf{R}]\!] \cup [\![\mathsf{S}]\!]$

$[\![\textbf{forall}\ \mathsf{x}\ \textbf{with}\ \varphi(\mathsf{x})\ \textbf{do}\ \mathsf{R}(\mathsf{x})]\!] \Rightarrow \bigcup_{\mathsf{a}:\varphi(\mathsf{a})} [\![\mathsf{R}(\mathsf{a})]\!]$

$[\![\textbf{choose}\ \mathsf{x}\ \textbf{with}\ \varphi(\mathsf{x})\ \textbf{do}\ \mathsf{R}(\mathsf{x})]\!] \Rightarrow [\![\mathsf{R}(\mathsf{a})]\!]$ for any a with $\varphi(\mathsf{a})$

If the set of updates is consistent (no two updates for the same f($\underline{\mathsf{a}}$))
then apply it to $\mathcal{A}$, to get a successor state.

Runs (also called "computations"): Finite or infinite sequences of algebras.

ASM Rules - Termination

A run ends (termination) either when the set of updates becomes empty in the last state (stuttering at the end) or when an explicitly given predicate becomes true (in KIV). A run may also end in a state where the set of updates cannot be computed or is inconsistent (interpreted as blocking, divergence).

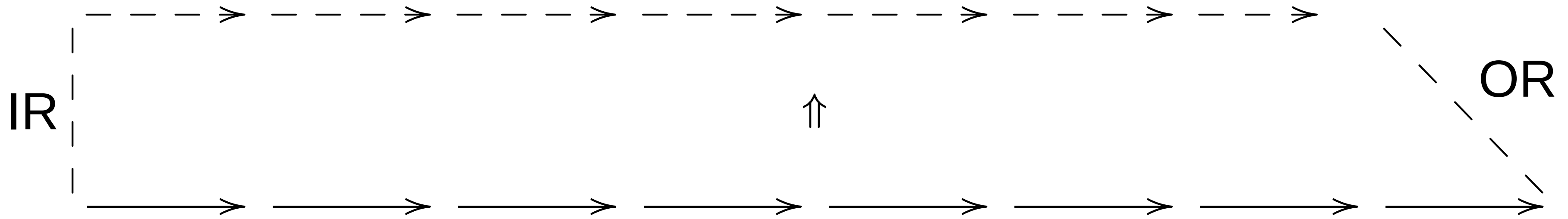
Usually either

- Rule has the form **if** φ **then** R, and $\neg\,\varphi$ indicates final states
- A set of rules of the form **if** φ_i **then** R_i is given,
  one is chosen indeterministically each time, so in final states all φ_i are false

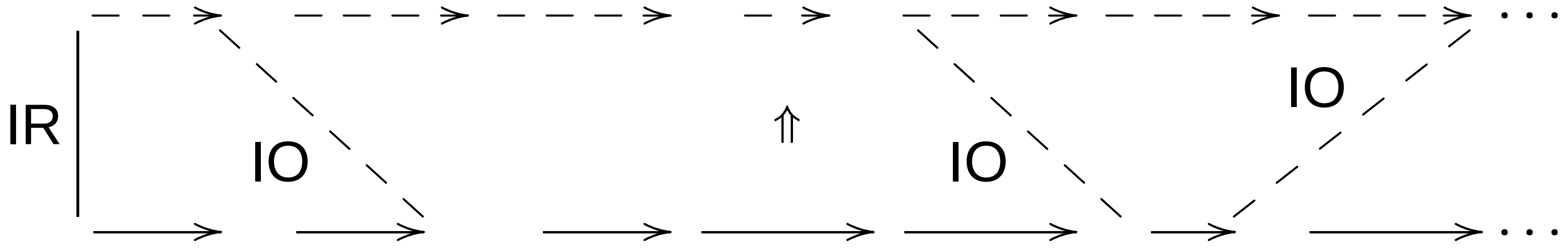
ASM Refinement

CSM refines ASM via IR, OR and IO iff

For finite runs:

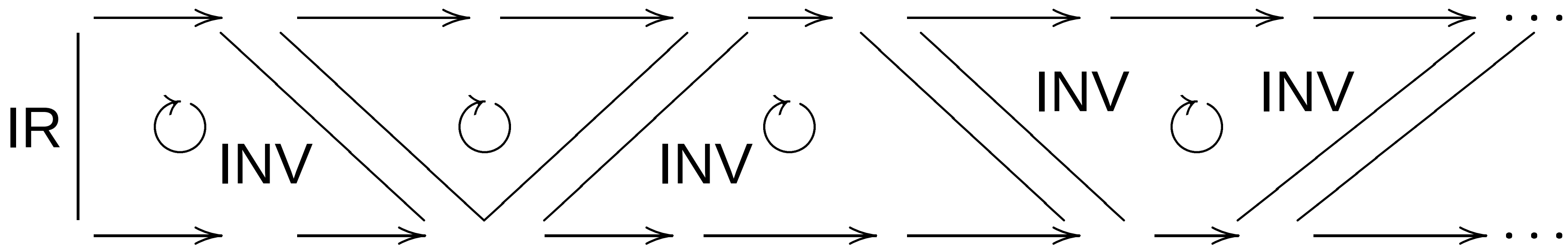


For infinite runs:



Generalized Forward Simulation

Find INV, such that



Proof obligations:

- When states are not final, a commuting m:n diagram can be added
- IR $\rightarrow$ INV
- INV $\rightarrow$ IO
- INV $\wedge$ final $\rightarrow$ OR

ASM Rules in KIV

- Idea: Map dynamic functions of a first-order signature to function variables
- $f(\underline{t}) := t'$ abbreviates $f := \lambda\, \underline{x}.\ \textbf{if}\ \underline{x} = \underline{t}\ \textbf{then}\ t'\ \textbf{else}\ f(\underline{x})$
- Can capture all of ASMs except: Parallel execution with $\textbf{forall}\ x\ \textbf{with}\ \varphi\ \textbf{do}\ R(x)$
- Z supports set theory encoding functions as sets of tuples
  $\Rightarrow$ essentially the same expressive power as dynamic functions

Abstract Specification of Transactions

$\text{ABTRANSFER}\#$

choose from, to, value, fail$?$

with authentic$(\text{from}) \wedge$ authentic$(\text{to}) \wedge$ from $\neq$ to $\wedge$ value $\leq$ balance(from)

in if $\neg$ fail$?$ **then** balance$(\text{from}) :=$ balance$(\text{from}) -$ value

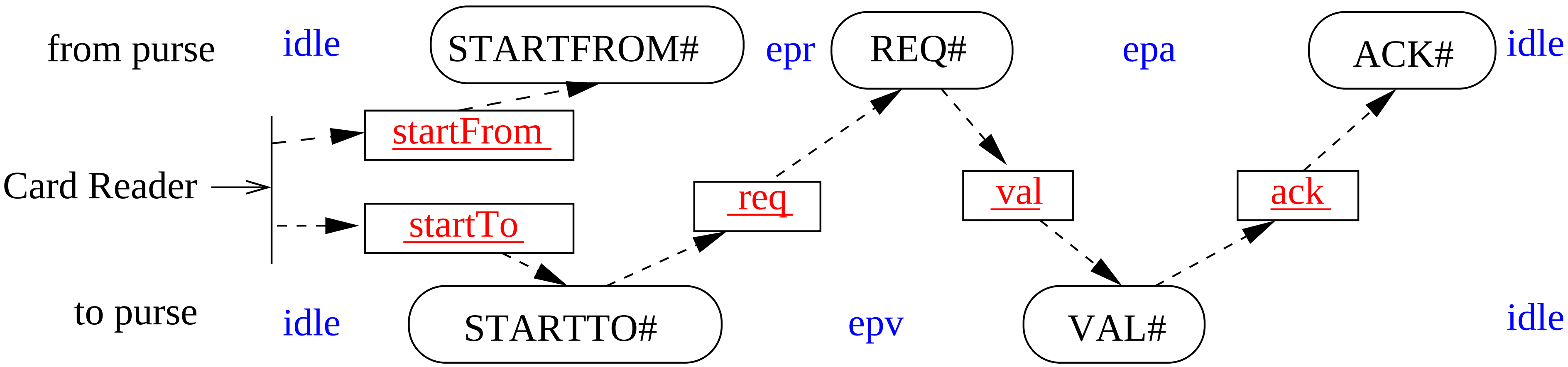
balance$(\text{to}) :=$ balance$(\text{to}) +$ value

else balance$(\text{from}) :=$ balance$(\text{from}) -$ value

lost$(\text{from}) :=$ lost$(\text{from}) +$ value

- fail? decides whether transfer worked
  (card may be pulled out of reader, insufficient memory, etc.)
- lost(from) stores money if transfer didn't work
- Functional Correctness:
  Σ_{purse} balance$(\text{purse}) +$ lost(purse) invariant $\Rightarrow$ no money generated/lost

The Concrete Protocol as an Activity Chart



- Activity chart for the protocol (drawn horizontally)
- content of actions abstracted to names of ASM rules (with #)
- states (upated in actions) added for clarity
- Not modelled: Before start, card Reader collects relevant data about both the to and from purse

Messages and Security

`message = startFrom | startTo | req | val | ack |` $\perp$

Idea: (implicit) attacker may at any time

- read old messages, modify the next message to a purse
- generate (publically available) `startFrom`, `startTo` messages
- send illegal messages $\perp$ (also used for “empty message”)

Security Assumption:

The attacker cannot forge `req, val, ack` messages by use of suitable cryptography

Representation of Security Assumption

- global variable ether : set(message) stores all previously sent messages and all publically available ones
- purse receiver receives some message msg from ether, produce output message outmsg
- ether may loose messages randomly
- to avoid replay attacks, each purse increments variable nextSeqNo in every transaction
- Both from and to purse store detailled information (PayDetails) pdAuth(purse) = (from, from_SeqNo, to, to_SeqNo, value) about the currently ongoing transaction

The Mondex Case Study: startFrom

- `receiver` = from purse gets
  `msg = startFrom(to_name, value, to_SeqNo)`

$$
\begin{array}{l}
\text{STARTFROM}\# \\
\textbf{let } \text{to\_name} = \text{msg.name}, \text{value} = \text{msg.value}, \\
\quad\ \ \text{to\_SeqNo} = \text{msg.nextSeqNo} \\
\textbf{in if} \quad \text{authentic}(\text{to\_name}) \wedge \text{receiver} \neq \text{to\_name} \\
\qquad \wedge\ \text{value} \leq \text{balance}(\text{receiver}) \wedge \neg\ \text{fail?} \\
\quad \textbf{then } \text{pdAuth}(\text{receiver}) := \text{mkpd}(\text{receiver}, \text{nextSeqNo}(\text{receiver}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{to\_name}, \text{to\_SeqNo}, \text{value}) \\
\qquad\qquad\quad \text{state}(\text{receiver}) := \text{epr} \\
\qquad\qquad\quad \text{INCREMENT}\#(\text{nextSeqNo}(\text{receiver})) \\
\qquad\qquad\quad \text{outmsg} := \perp \\
\quad \textbf{else } \text{outmsg} := \perp
\end{array}
$$

The Mondex Case Study: startTo

- `receiver` = to purse receives
  `msg = startTo(from_name, value, from_SeqNo)`
- dual to STARTFROM#, except for sending outmsg

STARTTO#
let from_name $=$ msg.name, value $=$ msg.value,
  from_SeqNo $=$ msg.nextSeqNo
in if authentic$($from_name$)$ $\wedge$ receiver $\neq$ from_name $\wedge$ $\neg$ fail$?$
  then pdAuth$($receiver$)$ $:=$ mkpd$($from_name, from_SeqNo, receiver,
        nextSeqNo$($receiver$)$, value$)$
      state$($receiver$)$ $:=$ epv
      INCREMENT$\#($nextSeqNo$($receiver$))$
      seq outmsg $:=$ req$($pdAuth$($receiver$))$
  else outmsg $:=$ $\perp$

The Mondex Case Study: req

- `receiver` = from purse receives `msg = req(PayDetails)`

REQ$\#$

if msg $=$ req$($pdAuth$($receiver$))$ $\wedge$ $\neg$ fail$?$

then balance$($receiver$)$ $:=$ balance$($receiver$)$ $-$ pdAuth$($receiver$)$.value

state$($receiver$)$ $:=$ epa

outmsg $:=$ val$($pdAuth$($receiver$))$

else outmsg $:=$ $\perp$

The Mondex Case Study: val and ack

- `receiver` = to purse receives `msg = val(PayDetails)`
- `receiver` = from purse receives `msg = ack(PayDetails)`

VAL#
if msg $=$ val(pdAuth(receiver)) $\wedge\ \neg$ fail?
then balance(receiver) $:=$ balance(receiver) $+$ pdAuth(receiver).value
state(receiver) $:=$ idle
outmsg $:=$ ack(pdAuth(receiver))
else outmsg $:= \perp$

ACK#
if msg $=$ ack(pdAuth(receiver)) $\wedge\ \neg$ fail?
then state(receiver) $:=$ idle
outmsg $:= \perp$
else outmsg $:= \perp$

The Mondex Case Study: Full ASM Rule

PROTOCOLSTEP$\#$

choose msg, receiver, fail$?$ **with** msg $\in$ ether $\wedge$ authentic(receiver) **in**

  if isStartTo(msg) $\wedge$ state(receiver) $=$ idle **then** STARTTO$\#$

  else if isStartFrom(msg) $\wedge$ state(receiver) $=$ idle **then** STARTFROM$\#$

  else if isreq(msg) $\wedge$ state(receiver) $=$ epr **then** REQ$\#$

  else if isval(msg) $\wedge$ state(receiver) $=$ epv **then** VAL$\#$

  else if isack(msg) $\wedge$ state(receiver) $=$ epa **then** ACK$\#$

  else ABORT$\#$

/* add outmsg to ether and */

/* nondeterministically loose some messages from ether */

choose ether$'$ **with** ether$'$ $\subseteq$ ether $\cup$ $\{$outmsg$\}$ **in** ether $:=$ ether$'$

The Mondex Case Study: Aborting and Logging

- All purses have exception logs exLog(purse) for failed transactions

$$
\begin{array}{l}
\text{ABORT}\# \\
\quad \text{INCREMENT}\#(\text{nextSeqNo}(\text{receiver})) \\
\quad \text{LOGIFNEEDED}\# \\
\quad \text{state}(\text{receiver}) := \text{idle} \\
\quad \text{outmsg} := \perp
\end{array}
$$

$$
\begin{array}{l}
\text{LOGIFNEEDED}\# \\
\quad \textbf{if } \text{state}(\text{receiver}) = \text{epa} \vee \text{state}(\text{receiver}) = \text{epv} \\
\quad \textbf{then } \text{exLog}(\text{receiver}) := \text{exLog}(\text{receiver}) \ \cup \ \{\text{pdAuth}(\text{receiver})\}
\end{array}
$$

- Core property that holds at the end of a protocol run:
  money in abstract lost $\Leftrightarrow$ money in both exLog(from) and exLog(to)

The Mondex Case Study: ASM vs. Z Specification

- Additional independent protocol for moving exception logs to a global archive of the bank (simple)
- Z Specification has additional complexity due to
  * the wish to model the implementation of protocol steps on purses (APDU's) faithfully (e.g. prefixing STARTFROM/TO# with ABORT# etc.)
  * the use of pre-postcondition style (Z) (e.g. lots of Ξ schemes to cope with frame problem)
  * the use of data refinement (e.g. disjuncting all operations with skip)

The Mondex Case Study: ASM vs. Z Specification: More differences

- Two idle states (one is unnecessary)
- Operations defined for a fixed purse, then lifted (promoted) to a set
- From and to purse are not chosen randomly, but from an input stream
- Frame problem: Each Z scheme must explicitly define the variables it does not modify (Ξ schemes)
- Each operations must be able to ignore input messages
- Not *one* ASM rule, but *many* operations
- Operations must be total (disjunction with skip)
- One protocol step must refine the transaction, all others skip