Preface

The International Colloquium on Theoretical Aspects of Computing (ICTAC) held in 2006 in Tunis, Tunisia, was the third of a series of events created by the International Institute for Software Technology of the United Nations University. The aim of the colloquium is to bring together researchers from academia, industry, and government to present their results, and exchange experience, ideas, and solutions for their problems in theoretical aspects of computing.

The previous events were held in Guiyang, China (2004), and Hanoi, Vietnam (2005). Beyond its scholarly goals, another main purpose of ICTAC is to promote cooperation in research and education between participants and their institutions, from developing and industrial countries, as in the mandate of the United Nations University.

These proceedings record the contributions from the invited speakers and from the technical sessions. We present four invited papers, 21 technical papers, selected out of 78 submissions from 24 countries, and two extended abstracts of tutorials.

The Programme Committee includes researchers from 27 countries. Each of the 78 papers was evaluated by at least three reviewers. After the evaluation, reports were returned to the Programme Committee for discussion and resolution of conflicts. Based on their recommendations, we concluded the consensus process, and selected the 21 papers that we present here. For the evaluation of the submitted tutorials, this year we had the help of a separate Programme Committee especially invited for that purpose.

We are grateful to all members of the Programme and Organizing Committees, and to all referees for their hard work. The support and encouragement of the Advisory Committee were invaluable assets. Without the support of our sponsoring institutions, ICTAC 2006 could not have been a reality. Their recognition of the importance of this event is greatly appreciated.

Finally, we would like to thank all the authors of the invited and submitted papers and tutorials, and all the participants of the colloquium. They are the main focus of the whole event.

November 2006
Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone
Programme Chairs
ICTAC 2006
ICTAC 2006 was organized by the International Institute for Software Technology of the United Nations University, the University of Tunis El Manar, the Ecole Nationale d’Ingénieurs de Tunis, the University of York, and the Conservatoire National des Arts et Métiers.

Conference Chairs

Programme Chairs
- Kamel Barkaoui (CNAM Paris, France)
- Ana Cavalcanti (University of York, UK)
- Antonio Cerone (UNU-IIST, Macau, SAR China)

Conference Chairs
- Mohamed Bettaz (Philadelphia University, Jordan)
- Ali Mili (NJIT and SEI/CMU, USA)

Finance Chairs
- Rahma Ben Ayed (Ecole Nationale d’Ingénieurs de Tunis, Tunisia)
- Dang Van Hung (UNU-IIST, Macau, SAR China)

Publicity Chairs
- Chris George (UNU-IIST, Macau, SAR China)
- Karim Djouani (Université Paris 12, France)

Advisory Committee

Dines Bjørner
- Technical University of Denmark and Japan
- Advanced Institute of Science and Technology

Manfred Broy
- Technische Universität München, Germany

José Luiz Fiadeiro
- University of Leicester, UK

Jifeng He
- East China Normal University, China

Mathai Joseph
- TATA Consultancy Services Limited, India

Shaoying Liu
- Hosei University, Japan

Zhiming Liu
- UNU-IIST, Macau, SAR China

Zohar Manna
- Stanford University, USA

Tobias Nipkow
- Technische Universität München, Germany
VIII  Organization

Mike Reed  UNU-IIST, Macau, SAR China
Jim Woodcock  University of York, UK

Programme Committee

Marc Aiguier  University of Evry, France
Eric Badouel  INRIA, France
Kamel Barkaoui (Co-chair)  CNAM Paris, France
José Barros  Universidade do Minho, Portugal
Hubert Baumeister  Technical University of Denmark, Denmark
Rahma Ben Ayed  Ecole Nationale d’Ingénieurs de Tunis, Tunisia
Mohamed Bettaz  Philadelphia University, Jordan
Jonathan Bowen  Museophile Limited, UK
Khaled Bsaies  Faculté des Sciences de Tunis, Tunisia
Christian Calude  University of Auckland, New Zealand
Ana Cavalcanti (Co-chair)  University of York, UK
Antonio Cerone (Co-chair)  UNU-IIST, Macau, SAR China
Jessica Chen  University of Windsor, Canada
Yifeng Chen  Durham University, UK
Jim Davies  University of Oxford, UK
Mourad Debbabi  Concordia University, Canada
David Deharbe  Federal University of Rio Grande do Norte, Brazil
Jin Song Dong  National University of Singapore, Singapore
Wan Fokkink  Vrije Universiteit, Netherlands
Marcelo Frias  University of Buenos Aires, Argentina
Maurizio Gabbrielli  Università di Bologna, Italy
Valentin Goranko  University of the Witwatersrand, South Africa
Susanne Graf  VERIMAG, France
Lindsay Groves  Victoria University of Wellington, New Zealand
Michael Hansen  Technical University of Denmark, Denmark
Thomas Henzinger  EPFL, Switzerland
Dang Van Hung  UNU-IIST, Macau, SAR China
Purush Iyer  North Carolina State University, USA
Petr Jančar  Technical University of Ostrava, Czech Republic
Takuya Katayama  JAIST, Japan
Maciej Koutny  Newcastle University, UK
Xuandong Li  Nanjing University, China
Xinxin Liu  Chinese Academy of Sciences, China
Antonia Lopes  University of Lisbon, Portugal
Andrea Maggiolo-Schettini  University of Pisa, Italy
Antoni Mazurkiewicz  Polish Academy of Sciences, Poland
Mohamed Mezguiche
Ali Mili
Roland Mittermeir
Carroll Morgan
Alberto Pardo
Jean-Eric Pin
R. Ramamujam
Anders Ravn
Wolfgang Reif
Ingrid Rewitzky
Mark Ryan
Augusto Sampaio
Bernhard Schätz
Emil Sekerinski
Carolyn Talcott
Tayssir Touili
Do Long Van
Martin Wirsing
Hongjun Zheng

Université de Boumerdes, Algeria
NJIT and SEI/CMU, USA
Klagenfurt University, Austria
University of New South Wales, Australia
Universidad de la República, Uruguay
LIAFA and CNRS, France
Institute of Mathematical Sciences, India
Aalborg University, Denmark
University of Augsburg, Germany
University of Stellenbosch, South Africa
University of Birmingham, UK
Federal University of Pernambuco, Brazil
Technische Universität München, Germany
McMaster University, Canada
SRI International, USA
LIAFA and CNRS, France
Institute of Mathematics, Vietnam
Ludwig-Maximilians-Universität Munich, Germany
The MathWorks, USA

Tutorial Programme Committee

Kamal Barkaoui (Co-chair)  
Faiza Belala  
Rahma Ben Ayed  
Ana Cavalcanti (Co-chair)  
Antonio Cerone (Co-chair)  
Karim Dhouani  
Serge Haddad  
Ryszard Janicki  
Ali Jaoua  
Tiziana Margaria  
Mourad Maouche  
Salma Mouline  
Ahmed Nacer  
Hung Ngo  
Gianna Reggio  
Yahya Slimani  
Helen Treharne  

CNAM Paris, France  
University Mentouri of Constantine, Algeria  
Ecole Nationale d’Ingénieurs de Tunis, Tunisia  
University of York, UK  
UNU-IIST, Macau, SAR China  
Université Paris 12, France  
Université Paris-Dauphine, France  
McMaster University, Canada  
University of Quatar, Quatar  
University of Potsdam, Germany  
Philadelphia University, Jordan  
Faculté des Sciences de Rabat, Morocco  
USTHB Algiers, Algeria  
State University of New York at Buffalo, USA  
Università di Genova, Italy  
Faculté des Sciences de Tunis, Tunisia  
University of Surrey, UK

Additional Referees

Stanislaw Ambroszkiewicz  
Agnès Arnould  

Eugene Asarin  
Jeremy Avigad
Marek Bednarczyk
Francine Blanchet-Sadri
Iovka Boneva
Roberto Bruni
Carlos Camarão
Choe Changil
Krishnendu Chatterjee
Jean-paul Comet
Stephanie Delaune
Raymond Devillers
Clelia De Felice
Cinzia Di Giusto
Alessandra Di Pierro
Moreno Falaschi
Adalberto Farias
Wit Foryś
Mickael Foursov
Goran Frehse
Rudolf Freund
Blaise Genest
Patrice Godefroid
Massimiliano Goldwurm
Roberta Gori
Kunihiro Hiraishi
Phan Trung Huy
Ying Jiang
Zhi Jin
Victor Khomenko
Leonid Kof
Martin Kot
Hanna Kozankiewicz
S. N. Krishna
Martin Leucker
Francesca Levi
Kamal Lodaya
Etienne Lozes
Linyun Luo
Assia Mahboubi
Florence Maraninchi
Nicolas Markey
Maria Chiara Meo
Alexandre Miquel
Ben Moszkowski
Regina Motz
Madhavan Mukund
Paritosh Pandya
Dave Parker
Pascal Poizat
Agata Półroda
Daniel Ratiu
Zdeněk Sawa
Guido Sciavicco
Laura Semini
Paula Severi
Natarajan Shankar
Luis Sierra
Mihaela Sighireanu
Ana Sokolova
Jason Steggles
S. P. Suresh
Nguyen Truong Thang
Marcel Tonga
Laurent Van Begin
Vincent van Oostrom
Tran Van Dung
Marcos Viera
César Viho
Phan Cong Vinh
Józef Winkowski
Bożena Woźna
Naijun Zhan
Liming Zhu
Rosalba Zizza

Sponsoring Institutions

International Institute for Software Technology of the United Nations University

Tunisian Ministry of Higher Education
Tunisian Ministry of Scientific Research, Technology and Competency Development

Tunisian Ministry of Communication Technologies

Tunis Science City

The University of Tunis El Manar and l’ Ecole Nationale d’Ingénieurs de Tunis

University of York
# Table of Contents

## Invited Papers

- Verifying a Hotel Key Card System ................................. 1
  *Tobias Nipkow*

- Z/Eves and the Mondex Electronic Purse ............................. 15
  *Jim Woodcock, Leo Freitas*

- Verification Constraint Problems with Strengthening ............... 35
  *Aaron Bradley, Zohar Manna*

## Semantics

- Quantitative $\mu$-Calculus Analysis of Power Management in Wireless Networks .......................................................... 50
  *Annabelle McIver*

- Termination and Divergence Are Undecidable Under a Maximum Progress Multi-step Semantics for LinCa ................................. 65
  *Mila Majster-Cederbaum, Christoph Minnameier*

- A Topological Approach of the Web Classification ...................... 80
  *Gabriel Ciobanu, Dănuţ Rusu*

## Concurrency

- Bisimulation Congruences in the Calculus of Looping Sequences ....... 93
  *Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, Angelo Troina*

- Stronger Reduction Criteria for Local First Search ..................... 108
  *Marcos Kurbán, Peter Niebert, Hongyang Qu, Walter Vogler*

- A Lattice-Theoretic Model for an Algebra of Communicating Sequential Processes .......................................................... 123
  *Malcolm Tyrrell, Joseph M. Morris, Andrew Butterfield, Arthur Hughes*

- A Petri Net Translation of $\pi$-Calculus Terms .......................... 138
  *Raymond Devillers, Hanna Klaudel, Maciej Koutny*
# Table of Contents

## Model Checking

Handling Algebraic Properties in Automatic Analysis of Security Protocols .............................................. 153  
*Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko*

A Compositional Algorithm for Parallel Model Checking of Polygonal Hybrid Systems ........................................... 168  
*Gordon Pace, Gerardo Schneider*

Thread-Modular Verification Is Cartesian Abstract Interpretation .... 183  
*Alexander Malkis, Andreas Podelski, Andrey Rybalchenko*

## Formal Languages

Capture-Avoiding Substitution as a Nominal Algebra ..................... 198  
*Murdoch Gabbay, Aad Mathijssen*

Prime Decomposition Problem for Several Kinds of Regular Codes ...... 213  
*Kieu Van Hung, Do Long Van*

A New Approach to Determinisation Using Bit-Parallelism .......... 228  
*Jan Šupol, Bořivoj Melichar*

## Logic and Type Theory

Proving ATL* Properties of Infinite-State Systems ..................... 242  
*Matteo Slanina, Henny Sipma, Zohar Manna*

Type Safety for FJ and FGJ ............................................. 257  
*Shuling Wang, Quan Long, Zongyan Qiu*

Partizan Games in Isabelle/HOLZF ........................................... 272  
*Steven Obua*

Proof-Producing Program Analysis ........................................... 287  
*Amine Chaieb*

## Real-Time and Mobility

Reachability Analysis of Mobile Ambients in Fragments of AC Term Rewriting ......................................................... 302  
*Giorgio Delzanno, Roberto Montagna*
Interesting Properties of the Real-Time Conformance Relation

Moez Krichen, Stavros Tripakis

Model Checking Duration Calculus: A Practical Approach

Roland Meyer, Johannes Faber, Andrey Rybalchenko

Spatio-temporal Model Checking for Mobile Real-Time Systems

Jan-David Quesel, Andreas Schäfer

Tutorials: Extended Abstracts

Tutorial on Formal Methods for Distributed and Cooperative Systems

Christine Choppy, Serge Haddad, Hanna Klaudel, Fabrice Kordon,
Laure Petrucci, Yann Thierry-Mieg

Decision Procedures for the Formal Analysis of Software

David Déharbe, Pascal Fontaine, Silvio Ranise,
Christophe Ringeissen

Author Index
Verifying a Hotel Key Card System

Tobias Nipkow
Institut für Informatik, TU München

Abstract. Two models of an electronic hotel key card system are contrasted: a state based and a trace based one. Both are defined, verified, and proved equivalent in the theorem prover Isabelle/HOL. It is shown that if a guest follows a certain safety policy regarding her key cards, she can be sure that nobody but her can enter her room.

1 Introduction

This paper presents two models for a hotel key card system and the verification of their safety (in Isabelle/HOL [3]). The models are based on Section 6.2, Hotel Room Locking, and Appendix E in the book by Daniel Jackson [2]. Jackson employs his Alloy system to check that there are no small counterexamples to safety. We confirm his conjecture of safety by a formal proof.

Most hotels operate a digital key card system. Upon check-in, you receive a card with your own key on it (typically a pseudorandom number). The lock for each room reads your card and opens the door if the key is correct. The system is decentralized, i.e. each lock is a stand-alone, battery-powered device without connection to the computer at reception or to any other device. So how does the lock know that your key is correct? There are a number of similar systems and we discuss the one described in Appendix E of [2]. Here each card carries two keys: the old key of the previous occupant of the room (key$_1$), and your own new key (key$_2$). The lock always holds one key, its “current” key. When you enter your room for the first time, the lock notices that its current key is key$_1$ on your card and recodes itself, i.e. it replaces its own current key with key$_2$ on your card. When you enter the next time, the lock finds that its current key is equal to your key$_2$ and opens the door without recoding itself. Your card is never modified by the lock. Eventually, a new guest with a new key enters the room, recodes the lock, and you cannot enter anymore.

After a short introduction of the notation we discuss two very different specifications, a state based and a trace based one, and prove their safety and their equivalence. The complete formalization is available online in the Archive of Formal Proofs at afp.sf.net.

1.1 Notation

HOL conforms largely to everyday mathematical notation. This section introduces further non-standard notation and in particular a few basic data types with their primitive operations.
Types The type of truth values is called \textit{bool}. The space of total functions is denoted by \( \Rightarrow \). Type variables start with a quote, as in \( \textquotelefthalf{a} \), \( \textquotelefthalf{b} \) etc. The notation \( t :: \tau \) means that term \( t \) has type \( \tau \).

Functions can be updated at \( x \) with new value \( y \), written \( f(x := y) \). The range of a function is \( \text{range } f \), \( \text{inj } f \) means \( f \) is injective.

Pairs come with the two projection functions \( \text{fst} :: \textquotelefthalf{a} \times \textquotelefthalf{b} \Rightarrow \textquotelefthalf{a} \) and \( \text{snd} :: \textquotelefthalf{a} \times \textquotelefthalf{b} \Rightarrow \textquotelefthalf{b} \).

Sets have type \( \textquotelefthalf{a} \text{set} \).

Lists (type \( \textquotelefthalf{a} \text{list} \)) come with the empty list \([\] \), the infix constructor \( \cdot \), the infix \( @ \) that appends two lists, and the conversion function \( \text{set} \) from lists to sets. Variable names ending in “\( s \)” usually stand for lists.

Records are constructed like this \( (f_1 = v_1, \ldots) \) and updated like this \( r(f_i := v_i, \ldots) \), where the \( f_i \) are the field names, the \( v_i \) the values and \( r \) is a record.

Datatype \textit{option} is defined like this

\[
\texttt{datatype } \textquotelefthalf{a} \text{ option } = \text{ None } \mid \text{ Some } \textquotelefthalf{a}
\]

and adjoins a new element \textit{None} to a type \( \textquotelefthalf{a} \). For succinctness we write \([a]\) instead of \( \text{ Some } a \).

Note that \([A_1; \ldots; A_n] \Rightarrow A \) abbreviates \( A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow A \), which is the same as “If \( A_1 \) and \( \ldots \) and \( A_n \) then \( A \)”.

2 A State Based Model

The model is based on three opaque types \textit{guest}, \textit{key} and \textit{room}. Type \textit{card} is just an abbreviation for \( \textquotelefthalf{key} \times \textquotelefthalf{key} \).

The state of the system is modelled as a record which combines the information about the front desk, the rooms and the guests.

\[
\texttt{record } \text{ state } =
\begin{align*}
\text{ owns } :: \text{ room } \Rightarrow \text{ guest option } \\
\text{ currk } :: \text{ room } \Rightarrow \text{ key } \\
\text{ issued } :: \text{ key set } \\
\text{ cards } :: \text{ guest } \Rightarrow \text{ card set } \\
\text{ roomk } :: \text{ room } \Rightarrow \text{ key } \\
\text{ isin } :: \text{ room } \Rightarrow \text{ guest set } \\
\text{ safe } :: \text{ room } \Rightarrow \text{ bool }
\end{align*}
\]

Reception records who \textit{owns} a room (if anybody, hence \textit{guest option}), the current key \textit{currk} that has been issued for a room, and which keys have been \textit{issued} so far. Each guest has a set of \textit{cards}. Each room has a key \textit{roomk} recorded in the lock and a set \textit{isin} of occupants. The auxiliary variable \textit{safe} is explained further below; we ignore it for now.

In specification languages like Z, VDM and B we would now define a number of operations on this state space. Since they are the only permissible operations on the state, this defines a set of \textit{reachable} states. In a purely logical environment
like Isabelle/HOL this set can be defined directly by an inductive definition. Each clause of the definition corresponds to a transition/operation/event. This is the standard approach to modelling state machines in theorem provers.

The set of reachable states of the system (called \( \text{reach} \)) is defined by four transitions: initialization, checking in, entering a room, and leaving a room:

\textit{init}:
\[
\text{inj \ initk } \implies \forall \owns = (\lambda r. \text{None}), \text{currk} = \text{initk}, \text{issued} = \text{range \ initk}, \text{cards} = (\lambda g. \{\}), \text{roomk} = \text{initk}, \text{isin} = (\lambda r. \{\}), \text{safe} = (\lambda r. \text{True}) \] \in \text{reach}
\]

\textit{check-in}:
\[
\{ s \in \text{reach}; k \notin \text{issued \ s} \} \implies s(\text{currk} := (\text{currk \ s})(r := k), \text{issued} := \text{issued \ s} \cup \{k\}, \text{cards} := (\text{cards \ s})(g := \text{cards \ s} \cup \{(\text{currk \ s \ r}, k)\}), \text{owns} := (\text{owns \ s})(r := \text{Some \ g}), \text{safe} := (\text{safe \ s})(r := \text{False}) \} \in \text{reach}
\]

\textit{enter-room}:
\[
\{ s \in \text{reach}; (k, k') \in \text{cards \ s \ g}; \text{roomk \ s \ r} \in \{k, k'\} \} \implies s(\text{isin} := (\text{isin \ s})(r := \text{isin \ s \ r} \cup \{g\}), \text{roomk} := (\text{roomk \ s})(r := k'), \text{safe} := (\text{safe \ s})(r := \text{owns \ s \ r} = [g] \land \text{isin \ s \ r} = \{\} \land k' = \text{currk \ s \ r} \\
\lor \text{safe \ s \ r}) \} \in \text{reach}
\]

\textit{exit-room}:
\[
\{ s \in \text{reach}; g \in \text{isin \ s \ r} \} \implies s(\text{isin} := (\text{isin \ s})(r := \text{isin \ s \ r} - \{g\}) \} \in \text{reach}
\]

There is no check-out event because it is implicit in the next check-in for that room: this covers the cases where a guest leaves without checking out (in which case the room should not be blocked forever) or where the hotel decides to rent out a room prematurely, probably by accident. Neither do guests have to return their cards at any point because they may lose cards or may pretend to have lost them. We will now explain the events.

\textit{init} Initialization requires that every room has a different key, i.e. that \text{currk} is injective. Nobody owns a room, the keys of all rooms are recorded as issued, nobody has a card, and all rooms are empty.

\textit{enter-room} A guest may enter if either of the two keys on his card equal the room key. Then \( g \) is added to the occupants of \( r \) and the room key is set to the second key on the card. Normally this has no effect because the second key is already the room key. But when entering for the first time, the first key on the card equals the room key and then the lock is actually recoded.

\textit{exit-room} removes an occupant from the occupants of a room.

\textit{check-in} for room \( r \) and guest \( g \) issues the card (\text{currk \ s \ r}, k) to \( g \), where \( k \) is new, makes \( g \) the owner of the room, and sets \text{currk \ s \ r} to the new key \( k \).
The reader can easily check that our specification allows the intended distributed implementation: entering only reads and writes the key in that lock, and check-in only reads and writes the information at reception.

In contrast to Jackson we require that initially distinct rooms have distinct keys. This protects the hotel from its guests: otherwise a guest may be able to enter rooms he does not own, potentially stealing objects from those rooms. Of course he can also steal objects from his own room, but in that case it is easier to hold him responsible. In general, the hotel may just want to minimize the opportunity for theft.

The main difference to Jackson’s model is that his can talk about transitions between states rather than merely about reachable states. This means that he can specify that unauthorized entry into a room should not occur. Because our specification does not formalize the transition relation itself, we need to include the isin component in order to express the same requirement. In the end, we would like to establish that the system is safe: only the owner of a room can be in a room:

\[
[s \in \text{reach}; \ g \in \text{isin } s \ r] \implies \text{owns } s \ r = \lfloor g \rfloor
\]

Unfortunately, this is just not true. It does not take a PhD in computer science to come up with the following scenario: because guests can retain their cards, there is nothing to stop a guest from reentering his old room after he has checked out (in our model: after the next guest has checked in), but before the next guest has entered his room. Hence the best we can do is to prove a conditional safety property: under certain conditions, the above safety property holds. The question is: which conditions? It is clear that the room must be empty when its owner enters it, or all bets are off. But is that sufficient? Unfortunately not. Jackson’s Alloy tool took 2 seconds [2, p. 303] to find the following “guest-in-the-middle” attack:

1. Guest 1 checks in and obtains a card \((k_1, k_2)\) for room 1 (whose key in the lock is \(k_1\)). Guest 1 does not enter room 1.
2. Guest 2 checks in, obtains a card \((k_2, k_3)\) for room 1, but does not enter room 1 either.
3. Guest 1 checks in again, obtains a card \((k_3, k_4)\), goes to room 1, opens it with his old card \((k_1, k_2)\), finds the room empty, and feels safe . . .

After Guest 1 has left his room, Guest 2 enters and makes off with the luggage.

Jackson now assumes that guests return their cards upon check-out, which can be modelled as follows: upon check-in, the new card is not added to the guest’s set of cards but it replaces his previous set of cards, i.e. guests return old cards the next time they check in. Under this assumption, Alloy finds no more counterexamples to safety — at least not up to 6 cards and keys and 3 guests and rooms. This is not a proof but a strong indication that the given assumptions suffice for safety. We prove that this is indeed the case.

It should be noted that the system also suffers from a liveness problem: if a guest never enters the room he checked in to, that room is forever blocked. In practice this is dealt with by a master key. We ignore liveness.
2.1 Formalizing Safety

It should be clear that one cannot force guests to always return their cards (or, equivalently, never to use an old card). We can only prove that if they do, their room is safe. However, we do not follow Jackson’s approach of globally assuming everybody returns their old cards upon check-in. Instead we would like to take a local approach where it is up to each guest whether he follows this safety policy. We allow guests to keep their cards but make safety dependent on how they use them. This generality requires a finer grained model: we need to record if a guest has entered his room in a safe manner, i.e. if it was empty and if he used the latest key for the room, the one stored at reception. The auxiliary variable safe records for each room if this was the case at some point between his last check-in and now. The main theorem will be that if a room is safe in this manner, then only the owner can be in the room. Now we explain how safe is modified with each event:

\textit{init} sets safe to \textit{True} for every room.
\textit{check-in} for room \( r \) resets safe \( s, r \) because it is not safe for the new owner yet.
\textit{enter-room} for room \( r \) sets safe \( s, r \) if the owner entered an empty room using the latest card issued for that room by reception, or if the room was already safe.
\textit{exit-room} does not modify safe.

The reader should convince his or herself that safe corresponds to the informal safety policy set out above. Note that a guest may find his room non-empty the first time he enters, and safe will not be set, but he may come back later, find the room empty, and then safe will be set. Furthermore, it is important that \textit{enter-room} cannot reset safe due to the disjunct \( \lor \text{safe} s, r \). Hence \textit{check-in} is the only event that can reset safe. That is, a room stays safe until the next \textit{check-in}. Additionally safe is initially \textit{True}, which is fine because initially injectivity of \textit{initk} prohibits illegal entries by non-owners.

Note that because none of the other state components depend on safe, it is truly auxiliary: it can be deleted from the system and the same set of reachable states is obtained, modulo the absence of safe.

We have formalized a very general safety policy of always using the latest card. A special case of this policy is the one called \textit{NoIntervening} by Jackson [2, p. 200]: every \textit{check-in} must immediately be followed by the corresponding \textit{enter-room}.

2.2 Verifying Safety

All of our lemmas are invariants of reach. The complete list, culminating in the main theorem, is this:

\begin{enumerate}
  \item s \in reach \implies currk s, r \in issued s
  \item [s \in reach; (k, k') \in cards s, g] \implies k \in issued s
  \item [s \in reach; (k, k') \in cards s, g] \implies k' \in issued s
\end{enumerate}
4. \( s \in \text{reach} \implies \text{room} k s k \in \text{issued} s \)
5. \( s \in \text{reach} \implies \forall r r'. (\text{curr} k s r = \text{curr} k s r') = (r = r') \)
6. \( s \in \text{reach} \implies (\text{curr} k s r, k') \notin \text{cards} s g \)
7. \( [s \in \text{reach}; (k_1, k) \in \text{cards} s g_1; (k_2, k) \in \text{cards} s g_2] \implies g_1 = g_2 \)
8. \( [s \in \text{reach}; \text{safe} s r] \implies \text{room} k s r = \text{curr} k s r \)
9. \( [s \in \text{reach}; \text{safe} s r; (k', \text{room} k s r) \in \text{cards} s g] \implies \text{owns} s r = \lfloor g \rfloor \)

**Theorem 1.** If \( s \in \text{reach} \) and \( \text{safe} s r \) and \( g \in \text{isin} s r \) then \( \text{owns} s r = \lfloor g \rfloor \).

The lemmas and the theorem are proved in this order, each one is marked as a simplification rule, and each proof is a one-liner: induction on \( s \in \text{reach} \) followed by \text{auto}.

Although, or maybe even because these proofs work so smoothly one may like to understand why. Hence we examine the proof of Theorem 1 in more detail.

The only interesting case is \text{enter-room}. We assume that guest \( g_1 \) enters room \( r_1 \) with card \( (k_1, k_2) \) and call the new state \( t \). We assume \( \text{safe} t r \) and \( g \in \text{isin} t r \) and prove \( \text{owns} t r = \lfloor g \rfloor \) by case distinction. If \( r_1 \neq r \), the claim follows directly from the induction hypothesis using \( \text{safe} s r \) and \( g \in \text{isin} t r \) because \( \text{owns} t r = \text{owns} s r \) and \( \text{safe} t r = \text{safe} s r \). If \( r_1 = r \) then \( g \in \text{isin} t r \) is equivalent with \( g \in \text{isin} s r \lor g = g_1 \). If \( g \in \text{isin} s r \) then \( \text{safe} s r \) follows from \( \text{safe} t r \) by definition of \text{enter-room} because \( g \in \text{isin} s r \) implies \( \text{isin} s r \neq \emptyset \). Hence the induction hypothesis implies the claim. If \( g = g_1 \) we make another case distinction. If \( k_2 = \text{room} k s r \), the claim follows immediately from Lemma 1.9 above: only the owner of a room can possess a card where the second key is the room key. If \( k_1 = \text{room} k s r \) then, by definition of \text{enter-room}, \( \text{safe} t r \) implies \( \text{owns} s r = \lfloor g \rfloor \lor \text{safe} s r \). In the first case the claim is immediate. If \( \text{safe} s r \) then \( \text{room} k s r = \text{curr} k s r \) (by Lemma 1.8) and thus \( (\text{curr} k s r, k) \in \text{cards} s g \) by assumption \( (k_1, k_2) \in \text{cards} s g_1 \), thus contradicting Lemma 1.6.

This detailed proof shows that a number of case distinctions are required. Luckily, they all suggest themselves to Isabelle via the definition of function update (\( := \)) or via disjunctions that arise automatically.

### 2.3 An Extension

To test the flexibility of our model we extended it with the possibility for obtaining a new card, e.g. when one has lost one’s card. Now reception needs to remember not just the current but also the previous key for each room, i.e. a new field \( \text{prev} k = \text{room} \Rightarrow \text{key} \) is added to \( \text{state} \). It is initialized with the same value as \( \text{curr} k \): though strictly speaking it could be arbitrary, this permits the convenient invariant \( \text{prev} k s r \in \text{issued} s \). Upon check-in we set \( \text{prev} k \) to \( (\text{prev} s)(r := \text{curr} k s r) \). Event \text{new-card} is simple enough:

\[
\begin{align*}
[s \in \text{reach}; \text{owns} s r = \lfloor g \rfloor] \\
\implies s((\text{cards} := (\text{cards} s)(g := \text{cards} s g \cup \{(\text{prev} s r, \text{curr} k s r)\}))]) \in \text{reach}
\end{align*}
\]

The verification is not seriously affected. Some additional invariants are required.
\[ s \in \text{reach} \implies \text{prevk } s \in \text{issued } s \]
\[ [s \in \text{reach}; \text{owns } s \begin{pmatrix} r \end{pmatrix} = \begin{pmatrix} g \end{pmatrix}] \implies \text{currk } s \begin{pmatrix} r \end{pmatrix} \neq \text{prevk } s \begin{pmatrix} r \end{pmatrix} \]
\[ [s \in \text{reach}; \text{owns } s \begin{pmatrix} r \end{pmatrix} = \begin{pmatrix} g \end{pmatrix}; g \neq g'] \implies (k, \text{currk } s \begin{pmatrix} r \end{pmatrix}) \notin \text{cards } s \begin{pmatrix} g \end{pmatrix} \]

but the proofs are still of the same trivial induct-auto format.

Adding a further event for loosing a card has no impact at all on the proofs.

3 A Trace Based Model

The only clumsy aspect of the state based model is safe: we use a state component to record if the sequence of events that lead to a state satisfies some property. That is, we simulate a condition on traces via the state. Unsurprisingly, it is not trivial to convince oneself that safe really has the informal meaning set out at the beginning of subsection 2.1. Hence we now describe an alternative, purely trace based model, similar to Paulson’s inductive protocol model [6]. The events are:

\textbf{datatype} \text{event} =
\begin{itemize}
  \item \text{Check-in guest room card}
  \item \text{Enter guest room card}
  \item \text{Exit guest room}
\end{itemize}

Instead of a state, we have a trace, i.e. list of events, and extract the state from the trace:

\begin{itemize}
  \item \text{initk} :: \text{room} \Rightarrow \text{key}
  \item \text{owns} :: \text{event list} \Rightarrow \text{room} \Rightarrow \text{guest option}
  \item \text{currk} :: \text{event list} \Rightarrow \text{room} \Rightarrow \text{key}
  \item \text{issued} :: \text{event list} \Rightarrow \text{key set}
  \item \text{cards} :: \text{event list} \Rightarrow \text{guest} \Rightarrow \text{card set}
  \item \text{rooomk} :: \text{event list} \Rightarrow \text{room} \Rightarrow \text{key}
  \item \text{isin} :: \text{event list} \Rightarrow \text{room} \Rightarrow \text{guest set}
  \item \text{hotel} :: \text{event list} \Rightarrow \text{bool}
\end{itemize}

Except for \text{initk}, which is completely unspecified, all these functions are defined by primitive recursion over traces:

\begin{itemize}
  \item \text{owns} \begin{pmatrix} \text{} \end{pmatrix} \begin{pmatrix} r \end{pmatrix} = \text{None}
  \item \text{owns} \begin{pmatrix} e \cdot s \end{pmatrix} \begin{pmatrix} r \end{pmatrix} = \begin{pmatrix}
      \begin{cases}
        \text{Check-in } \begin{pmatrix} g \end{pmatrix} \begin{pmatrix} r' \end{pmatrix} \begin{pmatrix} c \end{pmatrix} \Rightarrow \text{if } r' = r \text{ then } \begin{pmatrix} g \end{pmatrix} \text{ else } \text{owns } s \begin{pmatrix} r \end{pmatrix} \\
        \text{-} \Rightarrow \text{owns } s \begin{pmatrix} r \end{pmatrix}
      \end{cases}
    \end{pmatrix}
  \end{itemize}

\begin{itemize}
  \item \text{currk} \begin{pmatrix} \text{} \end{pmatrix} \begin{pmatrix} r \end{pmatrix} = \text{initk } r
  \item \text{currk} \begin{pmatrix} e \cdot s \end{pmatrix} \begin{pmatrix} r \end{pmatrix} = \begin{pmatrix}
      \begin{cases}
        k = \text{currk } s \begin{pmatrix} r \end{pmatrix} \Rightarrow \text{in case } e \text{ of Check-in } \begin{pmatrix} g \end{pmatrix} \begin{pmatrix} r' \end{pmatrix} \begin{pmatrix} c \end{pmatrix} \Rightarrow \text{if } r' = r \text{ then } \text{snd } c \text{ else } k \Rightarrow k
      \end{cases}
    \end{pmatrix}
  \end{itemize}

\begin{itemize}
  \item \text{issued} \begin{pmatrix} \text{} \end{pmatrix} = \text{range } \text{initk}
  \item \text{issued} \begin{pmatrix} e \cdot s \end{pmatrix} = \text{issued } s \cup \begin{pmatrix}
      \begin{cases}
        \text{case } e \text{ of Check-in } \begin{pmatrix} g \end{pmatrix} \begin{pmatrix} r \end{pmatrix} \begin{pmatrix} c \end{pmatrix} \Rightarrow \begin{pmatrix} \text{snd } c \end{pmatrix} \text{ else } k \Rightarrow \text{-}
      \end{cases}
    \end{pmatrix}
  \end{itemize}

\begin{itemize}
  \item \text{cards} \begin{pmatrix} \text{} \end{pmatrix} \begin{pmatrix} g \end{pmatrix} = \text{\emptyset}
  \item \text{cards} \begin{pmatrix} e \cdot s \end{pmatrix} \begin{pmatrix} g \end{pmatrix} = \begin{pmatrix}
      \begin{cases}
        C = \text{cards } s \begin{pmatrix} g \end{pmatrix} \Rightarrow \text{in case } e \text{ of Check-in } \begin{pmatrix} g' \end{pmatrix} \begin{pmatrix} r \end{pmatrix} \begin{pmatrix} c \end{pmatrix} \Rightarrow \text{if } g' = g \text{ then } \begin{pmatrix} C \cup C \text{ else } C \end{pmatrix} \Rightarrow \text{-}
      \end{cases}
    \end{pmatrix}
  \end{itemize}
roomk [] r = initk r
roomk (e · s) r =
(let k = roomk s r
  in case e of Enter g r’ (x, y) ⇒ if r’ = r then y else k | - ⇒ k)

isin [] r = ∅
isin (e · s) r =
(let G =isin s r
  in case e of Check-in g r c ⇒ G
      | Enter g r’ c ⇒ if r’ = r then {g} ∪ G else G
      | Exit g r’ ⇒ if r’ = r then G - {g} else G)

However, not every trace is possible. Function hotel tells us which traces correspond to real hotels:

hotel [] = True
hotel (e · s) =
(hotel s ∧
  (case e of Check-in g r (k, k’) ⇒ k = currk s r ∧ k’ /∈ issued s
     | Enter g r (k, k’) ⇒ (k, k’) ∈ cards s g ∧ roomk s r ∈ {k, k’}
     | Exit g r ⇒ g ∈isin s r))

Alternatively we could have followed Paulson [6] in defining hotel as an inductive set of traces. The difference is only slight.

3.1 Formalizing Safety

The principal advantage of the trace model is the intuitive specification of safety. Using the auxiliary predicate no-Check-in

no-Check-in s r ≡ ¬(∃ g c. Check-in g r c ∈ set s)

we define a trace to be safeo for a room if the card obtained at the last Check-in was later actually used to Enter the room:

safeo s r ≡ ∃ s1 s2 s3 g c.
  s = s3 @ [Enter g r c] @ s2 @ [Check-in g r c] @ s1 ∧ no-Check-in (s3 @ s2) r

A trace is safe if additionally the room was empty when it was entered:

safe s r ≡ ∃ s1 s2 s3 g c.
  s = s3 @ [Enter g r c] @ s2 @ [Check-in g r c] @ s1 ∧
  no-Check-in (s3 @ s2) r ∧isin (s2 @ [Check-in g r c] @ s1) r = {}

The two notions of safety are distinguished because, except for the main theorem, safeo suffices.

The alert reader may already have wondered why, in contrast to the state based model, we do not require initk to be injective. If initk is not injective, e.g. initk r1 = initk r2 and r1 ≠ r2, then [Enter g r2 (initk r1, k), Check-in g r1 (initk r1, k)] is a legal trace and guest g ends up in a room he is not the owner of. However, this is not a safe trace for room r2 according to our definition. This reflects that hotel rooms are not safe until the first time their owner has entered them. We no longer protect the hotel from its guests.