## Summary

- ground and non-ground maximality of literal wrt. clause
- ordered resolution with selection $Res_S^{\succ}$
  - inferences limited by ordering $\succ$
  - inferences limited by selection function $S$
- soundness and refutational completeness



# COMP60121: Automated Reasoning II

## Lecture 7

## Previously . . .

- Ways of limiting inferences & enhancing efficiency
  - ordering restriction $\succ$
  - selection function $S$
- Idea:
  - Inferences restricted to $\succ$-maximal or $S$-selected literals
  - $S$ overrides $\succ$
- ground and non-ground maximality of literal wrt. clause



## Application: Compactness of Propositional Logic

- A theoretical application of the refutational completeness of Res is compactness of propositional logic.
- Observe: If $N \vdash_{Cal} F$ then
  there exist $F_1, \ldots, F_n \in N$ s.t. $F_1, \ldots, F_n \vdash_{Cal} F$
  If $N \vdash_{Res} \bot$ then
  there exist $C_1, \ldots, C_n \in N$ s.t. $C_1, \ldots, C_n \vdash_{Res} \bot$
  (resembles compactness).

  Note, $N$ can be an infinite set of formulae.
- Recall

$$N \text{ is unsatisfiable iff for any interpretation } I,\ I \not\models N.$$
$$I \not\models N \text{ iff for some } C \in N,\ I \not\models C$$

## Compactness of Propositional Logic

Property 16 (Compactness)

Let $N$ be a set of propositional clauses. Then:

$N$ is unsatisfiable iff there exists a finite subset $M \subseteq N$ which is unsatisfiable.

Proof:

"$\Leftarrow$": trivial (why?).

"$\Rightarrow$": Let $N$ be unsatisfiable, i.e. $N \models \bot$

$\Rightarrow$ $Res^*(N)$ unsatisfiable, since $N \subseteq Res^*(N)$

$\Rightarrow$ $\bot \in Res^*(N)$ by refutational completeness of resolution

$\Rightarrow$ $\exists n \geq 0$ s.t. $\bot \in Res^n(N)$

$\Rightarrow$ $\bot$ has a finite resolution proof $\Pi$;
let M be the (finite) set of assumptions in $\Pi$.



## Application: Craig-Interpolation

- An application of ordered resolution is Craig-Interpolation, which has practically relevance for distributed reasoning.

Property 17 (Craig 1957)

Let $F$ and $G$ be two propositional formulae such that $F \models G$. Then there exists a formula $H$, such that

(i) $F \models H$ and $H \models G$, and

(ii) each propositional symbol occurring in $H$ occurs in both $F$ and $G$.

- $H$ is called an interpolant for $F \models G$
- The theorem also holds for first-order formulae. In the general case, a proof based on resolution technology is more complicated because of Skolemisation.



## Craig-Interpolation (cont'd)

Proof of Property 17:

Transform $F$ and $\neg G$ into CNF, without using structural transf.

Let $N$ and $M$, resp., denote the resulting clause sets.

Choose any atom ordering $\succ$ satisfying: $p \succ q$, if $p$ does not occur in $G$ and $q$ does occur in $G$.

Let $S$ be the empty selection function. Saturate $N$ wrt. $Res_S^{\succ}$ ~~(with empty selection function $S$)~~ to get $N^*$.

Let $N' = N^* \setminus \{C \mid C$ contains symbols that don't occur in $G\}$.

I.e. $C \in N'$ iff $C \in N^*$ and $C$ contains only symbols in $G$.

Let $H = \bigwedge N'$. Then, $F \models H$. (Why?)

To see that $H \models G$, take $N^* \cup M$ and saturate wrt. $Res_S^{\succ}$.

This derives $\bot$, but no inferences are performed on clauses in $N^* \setminus N'$ with symbols not in $G$.

This implies $N' \cup M \models \bot$ and therefore $H \models G$.



## Hyperresolution

- There are many refinements of resolution (Refer to Bachmair and Ganzinger (2001), "Resolution Theorem Proving", for further reading.)
- Well-known example: hyperresolution (Robinson 1965)
- Calculus parameterised by atom ordering $\succ$ and selection function $S$, as before
- Idea:
  - $S$ is 'maximal' selection function, i.e. selects all negative literals in any clause
  - Resolve and eliminate selected literals one by one
  - Hyperresolution rule does this in one step

## Hyperresolution (cont'd)

- As for $Res^{\succ}_{S}$, the calculus is parameterised by an atom ordering $\succ$ and a selection function $S$.
- But $S$ is the 'maximal' selection function, i.e. selects all negative literals in a clause.



## Hyperresolution (cont'd)

- **Hyperresolution calculus** *HRes*

$$\frac{C_1 \vee A_1 \quad \ldots \quad C_n \vee A_n \qquad \neg B_1 \vee \ldots \vee \neg B_n \vee D}{(C_1 \vee \ldots \vee C_n \vee D)\boldsymbol{\sigma}}$$

provided $\boldsymbol{\sigma}$ is the mgu s.t. $A_1\boldsymbol{\sigma} = B_1\boldsymbol{\sigma}$, $\ldots$, $A_n\boldsymbol{\sigma} = B_n\boldsymbol{\sigma}$, and

(i) $A_i\boldsymbol{\sigma}$ strictly maximal in $C_i\boldsymbol{\sigma}$, $1 \leq i \leq n$;

(ii) nothing is selected in $C_i$ (i.e. $C_i$ is positive);

(iii) the indicated $\neg B_i$ are exactly the ones selected by $S$, and $D$ is positive.

- **Ordered factoring rule:** as before

Property 18

*HRes* is sound and (refutationally) complete for f.o. clause logic



## Hyperresolution (cont'd)

- Note
  1. The positive clauses need not all be different.
  2. We regard clauses as equal if they are equal modulo variable renaming.
  3. $\boldsymbol{\sigma}$ is a simultaneous unifier.
- As we have seen, hyperresolution can be simulated by iterated binary resolution.
- However this yields intermediate clauses which *HRes* might not derive, and many of them might not be extendable into a full *HRes* inference.



## Exercise

- Let $P \succ R \succ Q$ and consider: $P \vee Q$, $R \vee Q$, $\neg R \vee \neg P$
- Give a *HRes* derivation.
- What does the derivation look like under a different ordering? $P \succ Q \succ R$?

## Exercise

- Let $P \succ R \succ Q$ and consider: $P \vee Q$, $R \vee Q$, $\neg R \vee \neg P$
- Give a *HRes* derivation.
  1. $\underline{P} \vee Q$ given
  2. $\underline{R} \vee Q$ given
  3. $\boxed{\neg R} \vee \boxed{\underline{\neg P}}$ given
  4. $\underline{Q} \vee \underline{Q}$ (1, 2, 3, *HRes*)
  5. $\underline{Q}$ (4, *Fact*)
- What does the derivation look like under a different ordering? $P \succ Q \succ R$?



## Summary

- Applications:
  - Compactness of propositional logic
  - Craig Interpolation of propositional logic
- Hyperresolution *HRes*

  = ordered resolution with maximal selection



## Exercise

- Let $P \succ R \succ Q$ and consider: $P \vee Q$, $R \vee Q$, $\neg R \vee \neg P$
- Give a *HRes* derivation.
  1. $\underline{P} \vee Q$ given
  2. $\underline{R} \vee Q$ given
  3. $\boxed{\neg R} \vee \boxed{\underline{\neg P}}$ given
  4. $\underline{Q} \vee \underline{Q}$ (1, 2, 3, *HRes*)
  5. $\underline{Q}$ (4, *Fact*)
- What does the derivation look like under a different ordering? $P \succ Q \succ R$?
  1. $\underline{P} \vee Q$ given
  2. $R \vee \underline{Q}$ given
  3. $\boxed{\neg R} \vee \boxed{\underline{\neg P}}$ given

  no inference step possible



# COMP60121: Automated Reasoning II

## Lecture 8

## Previously ...

- Compactness and Craig interpolation of propositional logic
- Ordered resolution with selection
- One variation:
  - Hyperresolution *HRes*

    = ordered resolution with maximal selection



## Redundancy

- Ordering and selection functions provide local restrictions of the resolution inference rules. They limit inferences to certain literals in clauses.
- What about not performing inferences with clauses altogether?
  Is it possible to just delete clauses?
  Under which circumstances are clauses unnecessary?
  (Conjecture: e. g., if they are tautologies or if they are subsumed by other clauses.)
- Intuition: Inferences with redundant clauses are not needed.



## A Formal Notion of Redundancy

- Let $N$ be a set of ground clauses and $C$ a ground clause (not necessarily in $N$). $C$ is called redundant wrt. $N$, if
  there exist $C_1, \ldots, C_n \in N$, $n \geq 0$, such that
  (i) all $C_i \prec C$, and
  (ii) $C_1, \ldots, C_n \models C$.
- Redundancy for general clauses:
  $C$ is called redundant wrt. $N$, if
  all ground instances $C\sigma$ of $C$ are redundant wrt. $G_\Sigma(N)$.
- $C$ is redundant in $N$ iff $C$ is redundant wrt. $N \setminus \{C\}$.
- Intuition: Redundant clauses are neither minimal exceptions nor productive. Note, the converse is not always true.
- Note: The same ordering $\succ$ is used for ordering restrictions and for redundancy (and for the completeness proof).



## Examples of Redundancy

Property 19

(i) $C$ tautology (i.e., $\models C$) $\Rightarrow$ $C$ redundant wrt. any set $N$.

(ii) $C\sigma \subset D \Rightarrow$ $D$ redundant wrt. $N \cup \{C\}$

(iii) $C\sigma \subseteq D \Rightarrow$ $D \vee \overline{L}\sigma$ redundant wrt. $N \cup \{C \vee L,\ D\}$, where $\overline{L}$ denotes the complement of $L$

- When $C\sigma \subset D$ for some $\sigma$ we say that $D$ is strictly subsumed by $C$. (Under certain conditions one may also use non-strict subsumption, but this requires a slightly more complicated definition of redundancy.)
- $D \vee \overline{L}\sigma$ is redundant wrt. any set containing $D$.

  $C\sigma \subseteq D \Rightarrow$ $D$ is a 'repeated factor' of the resolvent of $C \vee L$ and $D \vee \overline{L}\sigma$

## Saturation up to Redundancy

- Let $Red(N)$ denote the set of clauses redundant wrt. $N$.
- Recall, $N$ is saturated wrt. $Res_S^{\succ}$ iff $Res_S^{\succ}(N) \subseteq N$.
- $N$ is called saturated up to redundancy wrt. $Res_S^{\succ}$ iff

$$Res_S^{\succ}(N \setminus Red(N)) \subseteq N \cup Red(N)$$

In words: every conclusion of an $Res_S^{\succ}$-inference with non-redundant clauses in $N$ is in $N$ or is redundant.

Property 20 ($Res_S^{\succ}$ mod. redundancy is sound & ref. complete)
Let $N$ be saturated up to redundancy wrt. $Res_S^{\succ}$. Then

$$N \models \bot \text{ iff } \bot \in N.$$



## Saturation up to Redundancy (cont'd)

Proof (Sketch):
Ground case:

- consider construction of candidate model $I_N^{\succ}$ for $Res_S^{\succ}$
- redundant clauses are not productive
- redundant clauses in $N$ are not minimal exceptions for $I_N^{\succ}$

The premises of "essential" inferences are either minimal exceptions or productive.



## Preservation/Monotonicity Properties of Redundancy

Property 21

(i) $N \subseteq M \Rightarrow Red(N) \subseteq Red(M)$

(ii) $M \subseteq Red(N) \Rightarrow Red(N) \subseteq Red(N \setminus M)$

Proof: Exercise.

- This says that redundancy is preserved when, during a theorem proving process,
  (i) one adds (derives) new clauses or
  (ii) one deletes redundant clauses.



## A Resolution Prover

- So far: static view on completeness of resolution:
  - Saturated sets are unsatisfiable iff they contain $\bot$.
- We will now consider a dynamic view:
  - How can we get saturated sets in practice?
  - The Properties 20 and 21 are the basis for the completeness proof of our prover *RP*.

## Rules for Simplifications and Deletion

- We want to employ the following rules for simplification of prover states $N$:
  - Deletion of tautologies

    $$N \cup \{C \vee A \vee \neg A\} \triangleright N$$

  - Deletion of subsumed clauses

    $$N \cup \{C,\ D\} \triangleright N \cup \{C\}$$

    if $C\sigma \subseteq D$ ($C$ subsumes $D$).
  - Reduction (also called subsumption resolution)

    $$N \cup \{D \vee L,\ C \vee D\sigma \vee \overline{L}\sigma\} \triangleright N \cup \{D \vee L,\ C \vee D\sigma\}$$
    $$N \cup \{C \vee L,\ D \vee C\sigma \vee \overline{L}\sigma\} \triangleright N \cup \{C \vee L,\ D \vee C\sigma\}$$



## Resolution Prover *RP*

- **3 clause sets:**
  - **N**(ew) containing new conclusions
  - **P**(rocessed) containing simplified resolvents
  - **O**(ld) where clause are put once their inferences have been computed
- **Strategy:** Inferences will only be computed when there are no possibilities for simplification



## Transition Rules for *RP* (I)

Tautology elimination

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$
if $C$ is a tautology

Forward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O}$$
if some $D \in \boldsymbol{P} \cup \boldsymbol{O}$ subsumes $C$

Backward subsumption

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \cup \{D\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$
$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{D\} \quad \triangleright \quad \boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$
if $C$ strictly subsumes $D$



## Transition Rules for *RP* (II)

Forward reduction

$$\boldsymbol{N} \cup \{D \vee L\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \cup \{D\} \mid \boldsymbol{P} \mid \boldsymbol{O}$$
if there exists $C \vee L' \in \boldsymbol{P} \cup \boldsymbol{O}$
such that $\overline{L} = L'\sigma$ and $C\sigma \subseteq D$

Backward reduction

$$\boldsymbol{N} \mid \boldsymbol{P} \cup \{C \vee L\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$$
$$\boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C \vee L\} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C\}$$
if there exists $D \vee L' \in \boldsymbol{N}$
such that $\overline{L} = L'\sigma$ and $D\sigma \subseteq C$

## Transition Rules for *RP* (III)

Clause processing

$$\boldsymbol{N} \cup \{C\} \mid \boldsymbol{P} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O}$$

Inference computation

$$\emptyset \mid \boldsymbol{P} \cup \{C\} \mid \boldsymbol{O} \quad \triangleright \quad \boldsymbol{N} \mid \boldsymbol{P} \mid \boldsymbol{O} \cup \{C\},$$
$$\text{with } \boldsymbol{N} = Res_S^{\succ}(\boldsymbol{O} \cup \{C\})$$



## Soundness and Completeness

Property 22

$$N \models \bot \quad \text{iff} \quad N \mid \emptyset \mid \emptyset \overset{*}{\triangleright} N' \cup \{\bot\} \mid \_ \mid \_$$

Proof in
L. Bachmair, H. Ganzinger (2001), "Resolution Theorem Proving"
(on H. Ganzinger's Web page under Publications/Journals;
published in "Handbook on Automated Reasoning").



## Fairness

- Problem:
  - If $N$ is unsatisfiable, then $N \mid \emptyset \mid \emptyset \overset{*}{\triangleright} N' \cup \{\bot\} \mid \_ \mid \_$.
    This states the existence of a finite derivation of $\bot$.
  - Does this imply that every derivation starting from an unsatisfiable set $N$ eventually produces $\bot$?
  - No: a clause could be kept in $\boldsymbol{P}$ without ever being used for an inference.



## Fairness (cont'd)

- We need in addition a fairness condition:
  - If an inference is possible forever (that is, none of its premises is ever deleted), then it must be performed eventually.
- One possible way to guarantee fairness:
  Implement $\boldsymbol{P}$ as a queue
  (there are other techniques to guarantee fairness).
- With this additional requirement, we get a stronger result:
  If $N$ is unsatisfiable, then every fair derivation will eventually produce $\bot$.

## Summary

- General notion redundancy
  - Justifies deletion of clauses
  - Standard instances:
    tautology deletion, strict subsumption deletion
- Saturation up to redundancy
- Soundness & completeness of $Res_S^{\succ}$ modulo redundancy
- Implementing a (sound & complete) resolution prover
  - Specific redundancy elimination & simplification rules
  - Transition rules for deduction, deletion & simplification steps
  - Fairness



# COMP60121: Automated Reasoning II

## Lecture 9

## Previously . . .

- Advanced techniques of resolution theorem proving
  - optimised transformations to clause form
  - ordering restriction $\succ$
  - selection function $S$
  - redundancy elimination
- Powerful and versatile framework
- Implementing a resolution theorem prover



## Example: Neuman-Stubblebine Protocol

- Formalisation of a concrete application:
  Neuman-Stubblebine key exchange protocol.
- Proof by refutation:
  unsatisfiable $\Rightarrow$ intruder can break the protocol.
- Proof by consistency:
  consistency $\Rightarrow$ no unsafe states exist.
- Non-termination $\Rightarrow$ potential attack on the protocol.
- Termination requires elimination of redundancy.

## The Problem

- Automatic Analysis of Security Protocols using SPASS: An Automated Theorem Prover for First-Order Logic with Equality, by Christoph Weidenbach
- The growing importance of the internet causes a growing need for security protocols that protect transactions and communication. It turns out that the design of such protocols is highly error-prone. Therefore, there is a need for tools that automatically detect flaws like, e.g., attacks by an intruder.
- Here we show that our automated theorem prover SPASS can be used successfully to analyze the Neuman-Stubblebine key exchange protocol [1]. To this end the protocol is formalized in logic and then the security properties are automatically analyzed by SPASS. A detailed description of the analysis can be found in [2].

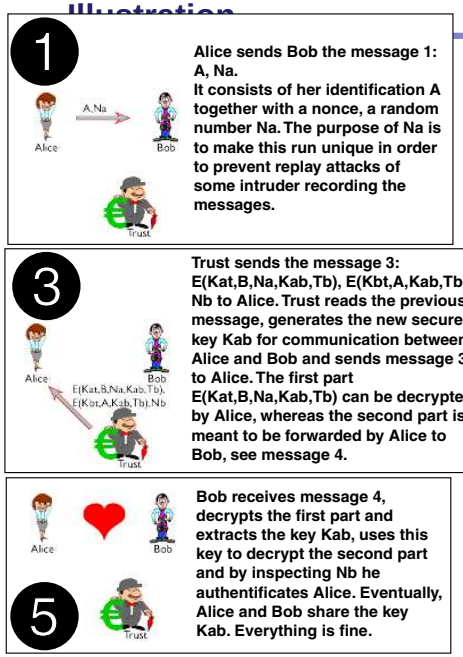


## The Problem (cont'd)

- The animation successively shows two runs of the Neuman-Stubblebine [1] key exchange protocol. The first run works the way the protocol is designed to do, i.e. it establishes a secure key $K_{ab}$ between Alice and Bob.
- The second run shows a potential problem of the protocol. An intruder may intercept the final message sent from Alice to Bob, replace it with a different message and may eventually own a key $N_a$ that Bob believes to be a secure key with Alice.
- The initial situation for the protocol is that the two participants Alice and Bob want to establish a secure key for communication among them. They do so with the help of a trusted key server, Trust, where both already have a secure key for communication with Trust.
- The next picture shows a sequence of four message exchanges that eventually establishes the key.



## Illustration






## Neuman-Stubblebine: A Regular Run

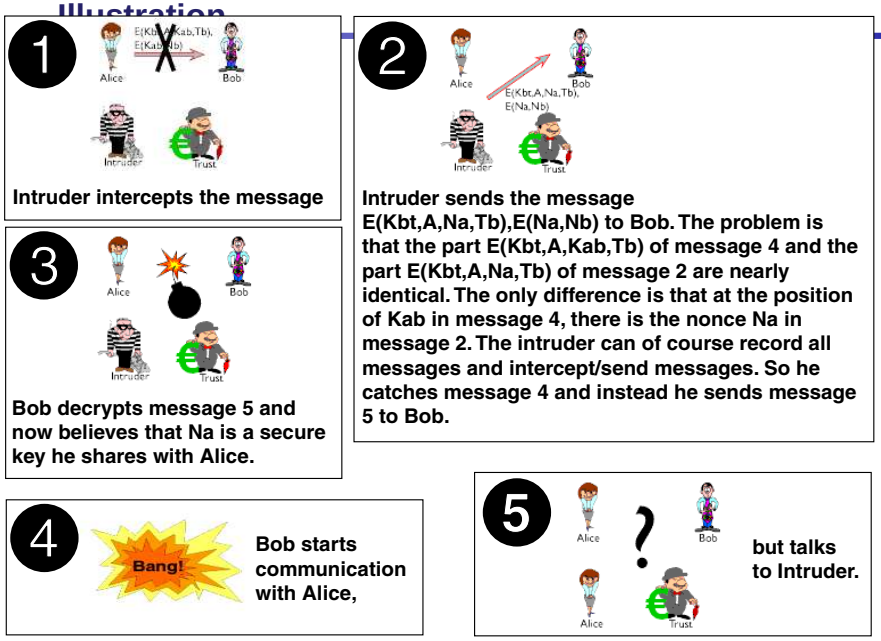
1. $A \xrightarrow{A,\ N_a} B$
2. $B \xrightarrow{B,\ N_b,\ E_{K_{bt}}(A,\ N_a,\ T_b)} T$
3. $T \xrightarrow{E_{K_{at}}(B,\ N_a,\ K_{ab},\ T_b),\ E_{K_{bt}}(A,\ K_{ab},\ T_b),\ N_b} A$
4. $A \xrightarrow{E_{K_{bt}}(A,\ K_{ab},\ T_b),\ E_{K_{ab}}(N_b)} B$

## What Can Happen?

- How can an intruder now break this protocol?
  - The key $K_{ab}$ is only transmitted inside encrypted parts of messages and
  - we assume that an intruder cannot break any keys nor does it know any of the initial keys $K_{at}$ or $K_{bt}$.
- Here is how ...



## Illustration






## Breaking the Neuman-Stubblebine Protocol

1. $A \xrightarrow{A,\ N_a} B$
2. $B \xrightarrow{B,\ N_b,\ E_{K_{bt}}(A,\ N_a,\ T_b)} T$
3. $T \xrightarrow{E_{K_{at}}(B,\ N_a,\ K_{ab},\ T_b),\ E_{K_{bt}}(A,\ K_{ab},\ T_b),\ N_b} A$
4. $A \xrightarrow{E_{K_{bt}}(A,\ K_{ab},\ T_b),\ E_{K_{ab}}(N_b)} B$
3. $I \xrightarrow{E_{K_{bt}}(A,\ N_a,\ T_b),\ E_{N_a}(N_b)} B$



## The Formalisation

- The key idea of the formalisation is to describe the set of sent messages.
- This is done by introducing a monadic predicate $M$ in first-order logic.
- Furthermore, every participant holds its set of known keys, represented by the predicates $Ak$ for Alice's keys, $Bk$ for Bob's keys, $Tk$ for Trust's keys and $Ik$ for the keys the intruder knows. (The remaining symbols are introduced and explained with their first appearance in a formula.)
- Then the four messages can be translated into the following formulae ...

## The Formalisation: Step 1

- **Step 1)** $A \rightarrow B : A, Na$

$$Ak(key(at, t)) \quad (1)$$

$$M(sent(a, b, pair(a, na))) \quad (2)$$

- (1) expresses that initially Alice holds the *key at* for communication with $t$ (for Trust). (2) states that she sends the first message.
- In order to formalize messages we employ a three place function sent where the first argument is the sender, the second the receiver and the third the content of the message.
- So the constant $a$ represents Alice, $b$ Bob, $t$ Trust and $i$ Intruder.
- The functions *pair* (*triple*, *quadr*) simply form sequences of messages of the indicated length.



## The Formalisation: Step 2

- **Step 1)** $A \rightarrow B : A, Na$
  **Step 2)** $B \rightarrow T : B, Nb, E_{Kbt}(A, Na, Tb)$

$$Bk(key(bt, t)) \quad (3)$$

$$\forall xa, xna\ [M(sent(xa, \underline{b}, pair(xa, xna))) \rightarrow M(sent(b, t, triple(b, \underline{nb(xna)}, encr(triple(xa, xna, \underline{tb(xna)}), bt))))] \quad (4)$$

- Bob holds the key *bt* for secure communication with Trust and whenever he receives a message of the form of message 1 (formula (2)), he sends a key request to Trust according to message 2.
- Note, encryption is formalized by the two place function *encr* where the first argument is the message and the second argument the key.
- Every lowercase symbol starting with an $x$ denotes a variable. The functions *nb* and *tb* generate, respectively, a new nonce and time span out of *xa*'s (Alice's) request represented by her nonce *xna*.



## The Formalisation: Step 3

- **Step 2)** $B \rightarrow T : B, Nb, E_{Kbt}(A, Na, Tb)$
  **Step 3)** $T \rightarrow A : E_{Kat}(B, Na, Kab, Tb),\ E_{Kbt}(A, Kab, Tb),\ Nb$

$$Tk(key(at, a))) \wedge Tk(key(bt, b)) \quad (5)$$

$$\forall xb, xnb, xa, xna, xbet, xbt, xat, xk\ [\ (M(sent(xb, \underline{t}, triple(xb, xnb, encr(triple(xa, xna, xbet), xbt)))) \wedge \underline{Tk(key(xbt, xb))} \wedge \underline{Tk(key(xat, xa))}) \rightarrow M(sent(t, xa, triple(encr(quadr(xb, xna, \underline{kt(xna)}, xbet), xat), encr(triple(xa, kt(xna), xbet), xbt), xnb)))\ ] \quad (6)$$

- Trust holds the keys for Alice and Bob and answers appropriately to a message in the format of message 2.
- Note: decryption is formalized by unification with an appropriate term where it is checked that the necessary keys are known to Trust.
- The server generates the key by applying his key generation function *kt* to the nonce *xna*.



## The Formalisation: Step 4

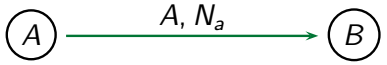
- **Step 3)** $T \rightarrow A : E_{Kat}(B, Na, Kab, Tb),\ E_{Kbt}(A, Kab, Tb),\ Nb$
  **Step 4)** $A \rightarrow B : E_{Kbt}(A, Kab, Tb),\ E_{Kab}(Nb)$

$$\forall xnb, xbet, xk, xm, xb, xna\ [\ M(sent(\underline{t}, \underline{a}, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb)) \rightarrow (M(sent(a, xb, pair(xm, encr(xnb, xk)))) \wedge Ak(key(xk, xb)))\ ] \quad (7)$$

$$\forall xbet, xk, xnb, xa, xna\ [\ M(sent(xa, \underline{b}, pair(encr(triple(xa, xk, \underline{tb(xna)}), bt), encr(\underline{nb(xna)}, xk))) \rightarrow Bk(key(xk, xa))] \quad (8)$$

- Finally, Alice answers according to the protocol to message 3 and stores the generated key for communication, formula (7).
- Formula (8) describes Bob's behaviour when he receives Alice's message. Bob decodes this message and stores the new key as well.

## A's Formalisation Part I




$P(a)$

$Ak(key(at, t))$

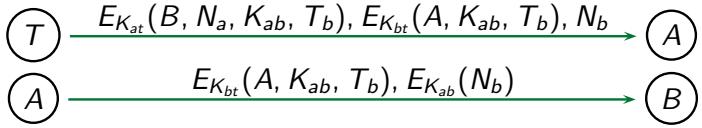
$M(sent(a, b, pair(a, na)))$

$Sa(pair(b, na))$

- $Sa$ is Alice's local store that will eventually be used to verify the nonce when it is sent back to her in Step (3).



## A's Formalisation Part II




$\forall xb, xna, xnb, xk, xbet, xm$

$[\, M(sent(\underline{t}, \underline{a}, triple(encr(quadr(xb, xna, xk, xbet), at), xm, xnb)))$

$\wedge\ \underline{Sa(pair(xb, xna))}$

$\longrightarrow$

$M(sent(a, xb, pair(xm, encr(xnb, xk))))$

$\wedge\ Ak(key(xk, xb))\,]$



## The Intruder

- The Intruder is modeled as an exhaustive, active attacker. It records all messages, decomposes the messages as far as possible and generates all possible new compositions.
- Furthermore, any object it has in hand is considered as a key and tried for encryption as well as for decryption.
- All these messages are posted. The set of messages the intruder has available is represented by the predicate *Im*.
- The participants are Alice, Bob, Trust and Intruder:

$$P(a) \wedge P(b) \wedge P(t) \wedge P(i) \tag{9}$$

- The intruder records all messages:

$$\forall xa, xb, xm\ [M(sent(xa, xb, xm)) \longrightarrow Im(xm)] \tag{10}$$



## The Intruder

- It decomposes and decrypts all messages it owns the key for:

$$\forall u, v\ [Im(pair(u, v)) \longrightarrow Im(u) \wedge Im(v)] \tag{11}$$

$$\forall u, v, w\ [Im(triple(u, v, w)) \longrightarrow Im(u) \wedge Im(v) \wedge Im(w)] \tag{12}$$

$$\forall u, v, w, z\ [Im(quadr(u, v, w, z)) \longrightarrow Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(z)] \tag{13}$$

$$\forall u, v, w\ [Im(encr(u, v)) \wedge Ik(key(v, w)) \longrightarrow Im(u)] \tag{14}$$

- It composes all possible messages:

$$\forall u, v\ [Im(u) \wedge Im(v) \longrightarrow Im(pair(u, v))] \tag{15}$$

$$\forall u, v, w\ [Im(u) \wedge Im(v) \wedge Im(w) \longrightarrow Im(triple(u, v, w))] \tag{16}$$

$$\forall u, v, w, x\ [Im(u) \wedge Im(v) \wedge Im(w) \wedge Im(x) \longrightarrow Im(quadr(u, v, w, x))] \tag{17}$$

## The Intruder

- It considers every item to be a key and uses it for encryption:

$$\forall v, w\ [Im(v) \wedge P(w) \longrightarrow Ik(key(v, w))] \quad (18)$$

$$\forall u, v, w\ [Im(u) \wedge Ik(key(v, w)) \wedge P(w) \longrightarrow Im(encr(u, v))] \quad (19)$$

- It sends everything:

$$\forall x, y, u\ [P(x) \wedge P(y) \wedge Im(u) \longrightarrow M(sent(x, y, u))] \quad (20)$$

- Finally we must formalize the insecurity requirement. Intruder must not have any key for communication with Bob that Bob believes to be a secure key for Alice:

$$\exists x\ [Ik(key(x, b)) \wedge Bk(key(x, a))]$$

This actually states the opposite, i.e. that an attack exists. Thus finding a proof means there is a vulnerability.



## SPASS Solves the Problem

- Now the protocol formulae together with the intruder formulae (9)-(20) and the insecurity formula above can be given to SPASS. Then SPASS automatically proves that this formula holds and that the problematic key is the nonce *Na*.
- The protocol can be repaired by putting type checks on the keys, such that keys can no longer be confused with nonces. This can be added to the SPASS first-order logic formalisation.
  Then SPASS disproves the insecurity formula above.
- The type checking capability is currently unique to SPASS.
- Further details can be found in [2], below. The experiment is available in full detail from the SPASS home page in the download area or the COMP6012 directory.



## References

[1] Neuman, B. C. and Stubblebine, S. G., 1993, A note on the use of timestamps as nonces, ACM SIGOPS, Operating Systems Review, 27(2), 10-14.

[2] Weidenbach, C., 1999, Towards an automatic analysis of security protocols in first-order logic, in 16th International Conference on Automated Deduction, CADE-16, Vol. 1632 of LNAI, Springer, pp. 378-382.



## Summary: Resolution Theorem Proving

- Resolution is a machine calculus.
- Parameters:
  - atom ordering $\succ$ and selection function $S$.
  - On the non-ground level, ordering constraints can (only) be solved approximatively.
- Completeness proof by constructing candidate models
  - from productive clauses $C \vee A$, $A \succ C$;
    inferences with these reduce exceptions.

## Summary: Resolution Theorem Proving (cont'd)

- Local restrictions of inferences via $\succ$ and $S$
  - $\Rightarrow$ fewer proof variants;
  - $\Rightarrow$ justification for numerous standard refinements;
  - $\Rightarrow$ simulation of certain kinds of tableaux and other proof methods.
- Global restrictions of the search space via elimination of redundancy
  - $\Rightarrow$ computing with "smaller" clause sets;
  - $\Rightarrow$ termination on many expressive, decidable fragments.
- Further specialisation of inference systems required for reasoning with orderings, equality and specific algebraic theories (lattices, abelian groups, rings, fields), integers



## Current research

- AR for specific theories: $\approx$, transitive relations, arithmetic, ...
- Resolution decision procedures for solvable fragments of f.-o. logic
- Implementing fast automated theorem provers (optimisations, heuristics, experimentation, ...)
- Making AR tools more intelligent
- Synthesis and analysis of different proof methods
- Platforms for combinations of reasoning tools: other provers (e.g. h.o. provers, theory decision procedures), model checkers, ..., SMT
- AR with distributed, heterogeneous, dynamic information
- AR for description logics, ontologies and the semantic web
- AR for modal logics, multi-agent systems
- Security protocol analysis; soft-/hardware design & verification
- ...



## Research at Manchester (incomplete summary)

- automated theorem proving
  - resolution
  - tableau
  - other methods
- decision procedures
- model generation, s.o. quantifier elimination
- verification, model checking
- first-order logic, modal logics, description logics, program logics, temporal logics
- Vampire, SPASS, SCAN, ... FACT++, ICOM, KAON2, ...
- reasoning for semantic web, ontologies
- AR for natural language processing



# COMP60121: Automated Reasoning II

## Appendix

## Basic Notions

- Let $\star$ be an operator defined over (elements of) a set $X$.
- $\star$ is commutative iff
  for any $x, y \in X$, $x \star y = y \star x$.
- $\star$ is associative iff
  for any $x, y, z \in X$, $((x \star y) \star z) = (x \star (y \star z))$.



## Notation

- $p$, $q$, $P$, $Q$ predicate symbols
- $x$, $y$, $z$ variables
- $a$, $b$, $c$ constants
- $f$, $g$ function symbols
- $s$, $t$ terms
- $A$, $B$ atoms
- $L$ literals
- $C$, $D$ clauses
- $N$ sets of clauses
- $F$, $G$ formulae
- $F[G]$ $G$ is subformula of $F$
- $\sigma$ substitutions
- $\Sigma$ given signature
- $X$ given set of variables
- $T_\Sigma$ terms over signature
- $x \mapsto s$ substitution of term $s$ into variable $x$
- $\mathcal{M}$ f.o. interpretation (mapping)
- $\overline{L}$ complement of literal $L$