Reconstruction of attacks in Proverif

Jules Timmerman Vincent Cheval

2025

Outline

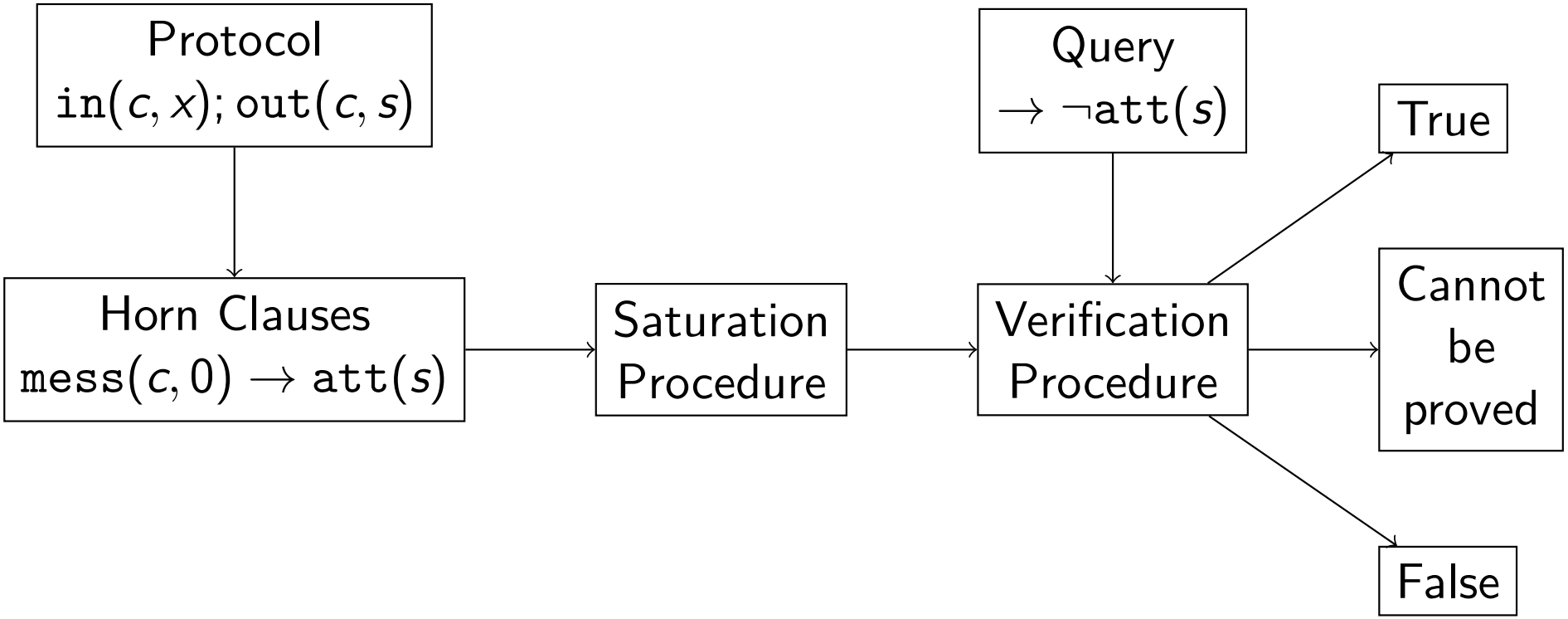
What is Proverif?

- Protocol:

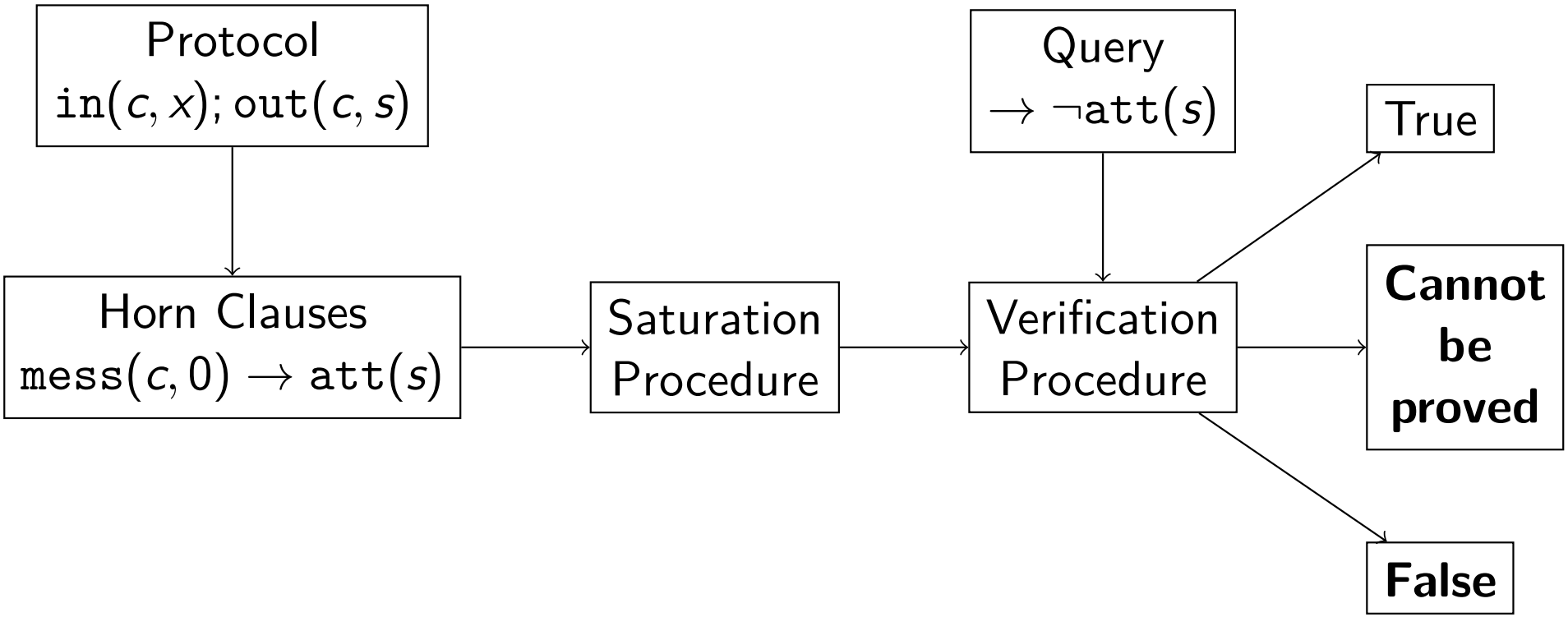
$$P_a = \texttt{new}\ k; \texttt{out}(c, \mathit{sign}(\mathit{enc}(k, pk_b), sk_a))$$
$$P_b = \texttt{in}(c, x); \texttt{if}\ \mathit{verify}(x, pk_a)\ \texttt{then}\ \texttt{out}(c, \mathit{enc}(s, \mathit{dec}(x, sk_b)))$$
$$P = P_a \mid P_b$$

- Security property: can the attacker learn s?
- Query: $\rightarrow \texttt{att}(s)$
- Omnipotent attacker (Dolev-Yao)

Structure of Proverif



Structure of Proverif



- Overapproximations
- Derivation $\rightarrow$ Real trace?

Example process

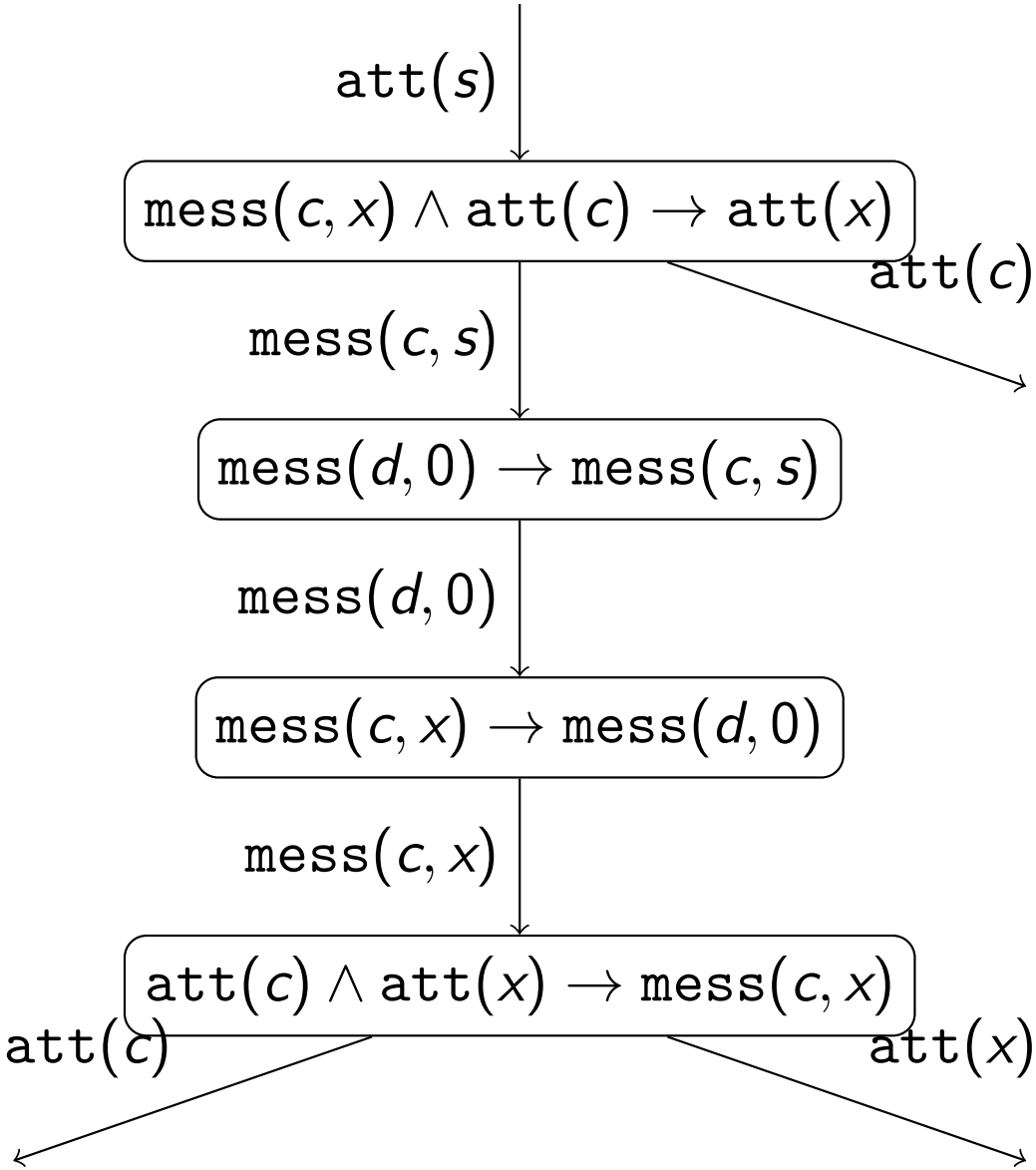
Example

$$\begin{aligned} P = \;& in(c, x); out(d, 0) \\ \mid\; & in(d, y); out(c, s) \end{aligned}$$

with query $\neg att(s)$, where:

- c is a public channel,
- d is a private channel
- s is a secret value.

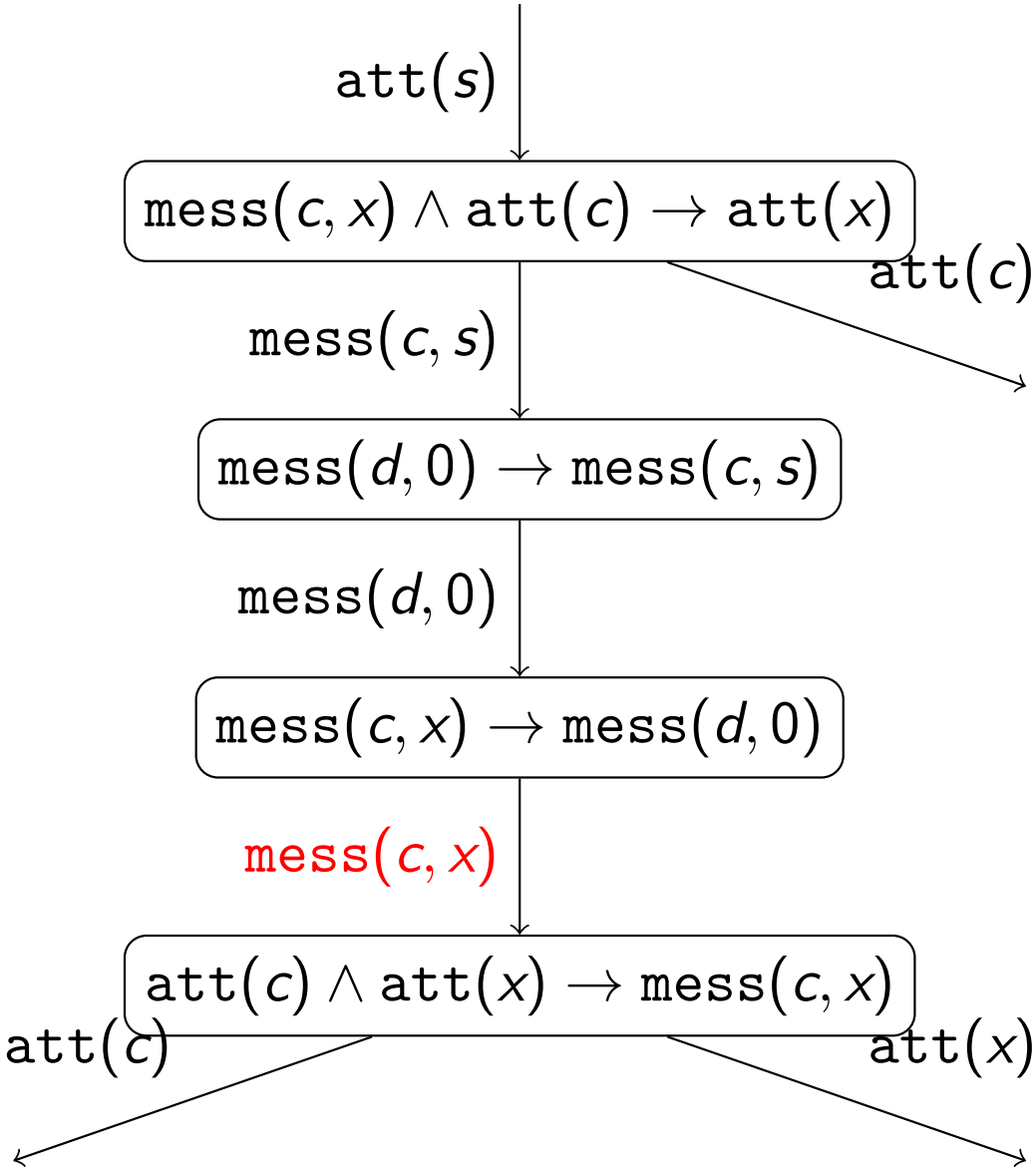
Reconstruction idea: guiding steps (from 2005)



$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

How do I step?

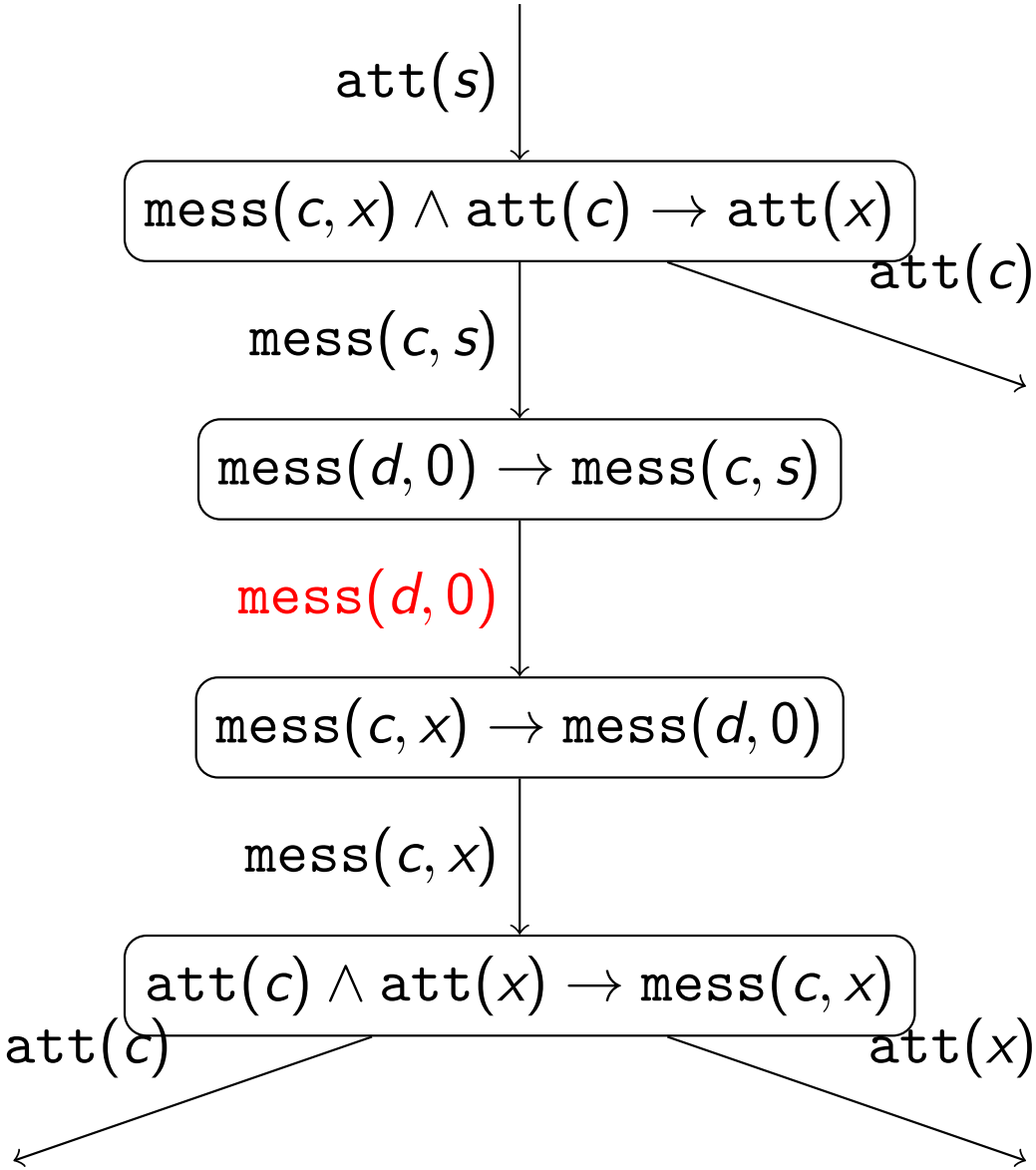
Reconstruction idea: guiding steps (from 2005)



$$P = \texttt{in}(c, x); \texttt{out}(d, 0) \mid \texttt{in}(d, y); \texttt{out}(c, s)$$

How do I step?

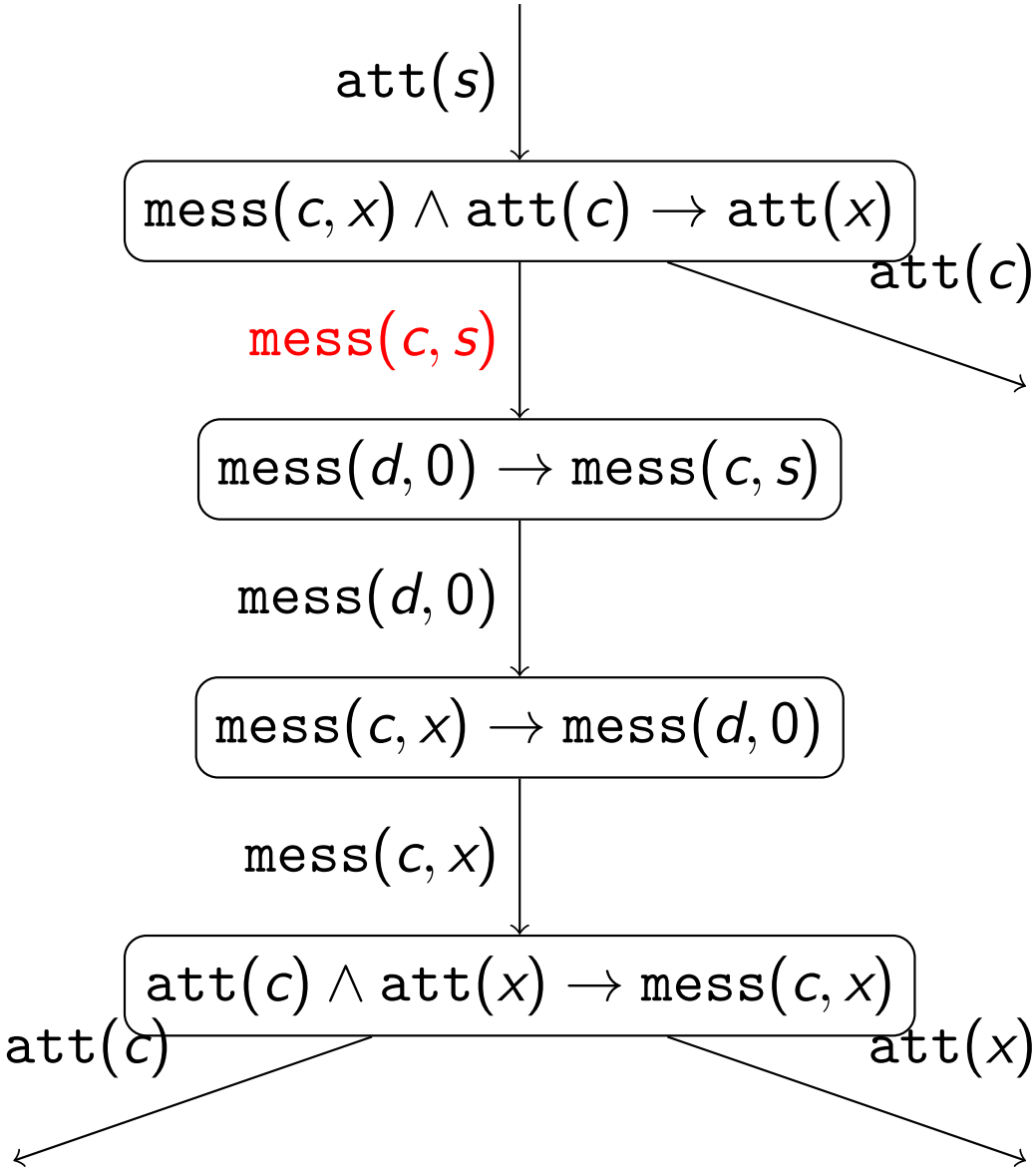
Reconstruction idea: guiding steps (from 2005)



$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

How do I step?

Reconstruction idea: guiding steps (from 2005)



$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

How do I step?

Translation of process

- Process to Horn clauses
- Hypothesis: inputs and others (ex: conditional)
- Conclusions: outputs

Example

$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

Translation of process

- Process to Horn clauses
- Hypothesis: inputs and others (ex: conditional)
- Conclusions: outputs

Example

$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

$$\mathtt{mess}(c, x)$$

Translation of process

- Process to Horn clauses
- Hypothesis: inputs and others (ex: conditional)
- Conclusions: outputs

Example

$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

$$\mathtt{mess}(c, x) \rightarrow \mathtt{mess}(d, 0)$$

Translation of process

- Process to Horn clauses
- Hypothesis: inputs and others (ex: conditional)
- Conclusions: outputs

Example

$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

$$\begin{array}{l} \mathtt{mess}(c, x) \rightarrow \mathtt{mess}(d, 0) \\ \mathtt{mess}(d, y) \end{array}$$

Translation of process

- Process to Horn clauses
- Hypothesis: inputs and others (ex: conditional)
- Conclusions: outputs

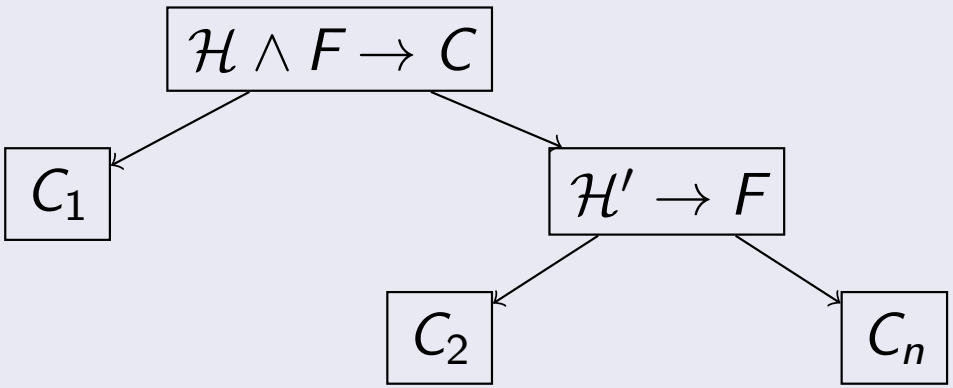
Example

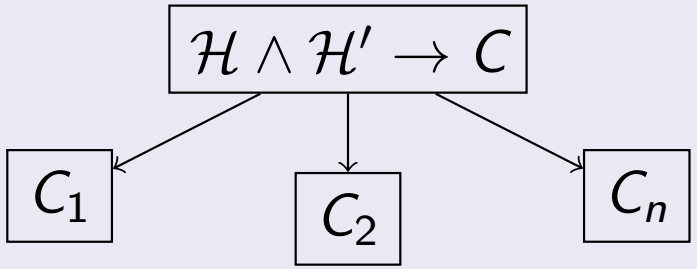
$$P = \begin{array}{l} \mathtt{in}(c, x); \mathtt{out}(d, 0) \\ \mid \mathtt{in}(d, y); \mathtt{out}(c, s) \end{array}$$

$$\begin{array}{l} \mathtt{mess}(c, x) \rightarrow \mathtt{mess}(d, 0) \\ \mathtt{mess}(d, y) \rightarrow \mathtt{mess}(c, s) \end{array}$$

Saturation and derivation example

Idea: Squish!



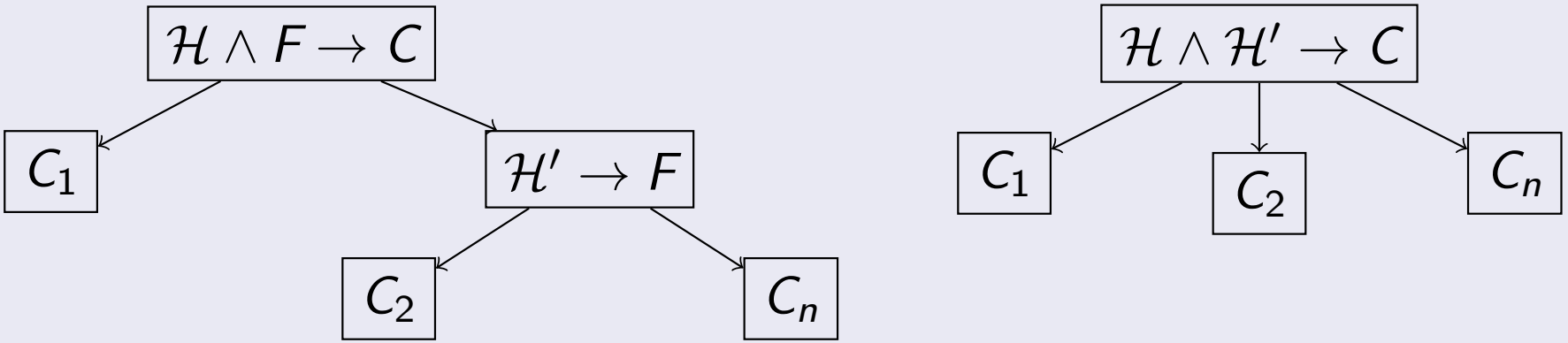


Example

$$mess(c, x) \to mess(d, 0)$$
$$mess(d, y) \to mess(c, s)$$
$$att(c) \wedge att(x) \to mess(c, x)$$

Saturation and derivation example

Idea: Squish!



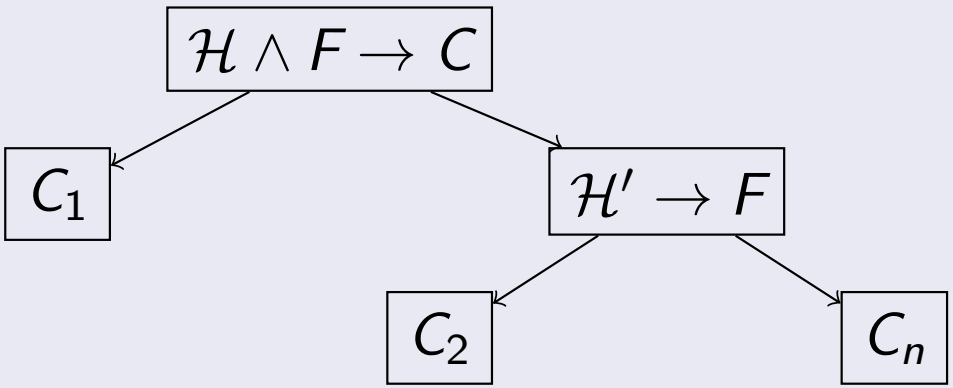
Example

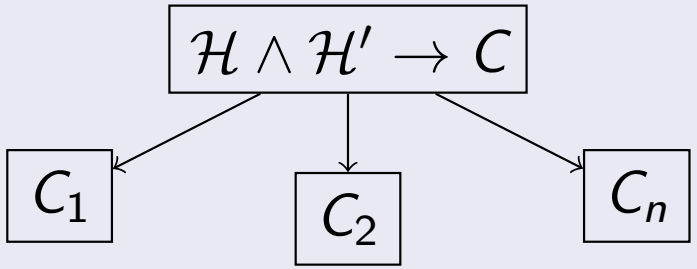
$$\begin{aligned} \mathit{mess}(c, x) &\to \mathit{mess}(d, 0) \\ \mathit{mess}(d, y) &\to \mathit{mess}(c, s) \\ \mathit{att}(c) \wedge \mathit{att}(x) &\to \mathit{mess}(c, x) \end{aligned}$$

$$\mathit{att}(c) \wedge \mathit{att}(x) \to \mathtt{mess}(d, 0)$$

Saturation and derivation example

Idea: Squish!





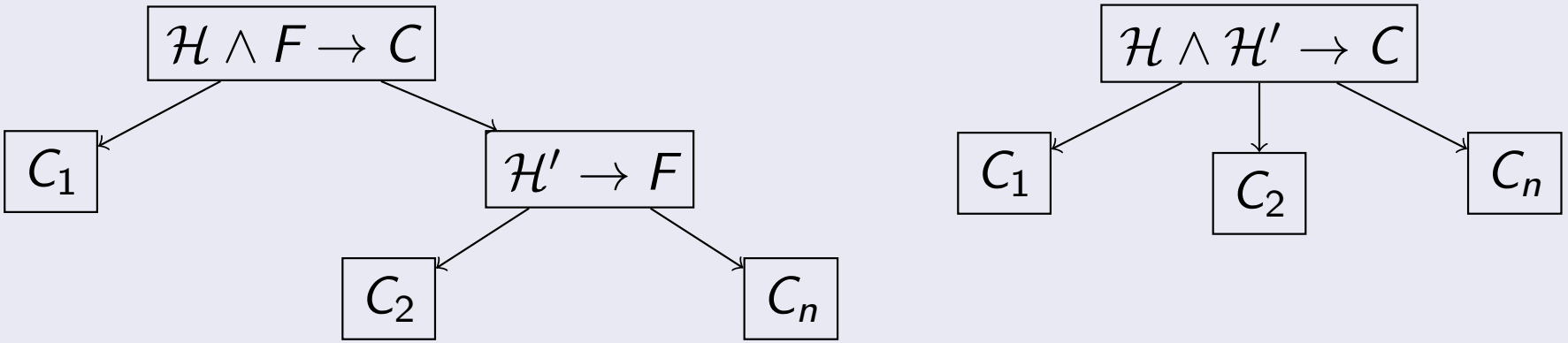
Example

$$\begin{aligned} mess(c, x) &\to mess(d, 0) \\ mess(d, y) &\to mess(c, s) \\ att(c) \wedge att(x) &\to mess(c, x) \end{aligned}$$

$$\to \mathtt{mess}(d, 0)$$

Saturation and derivation example

Idea: Squish!



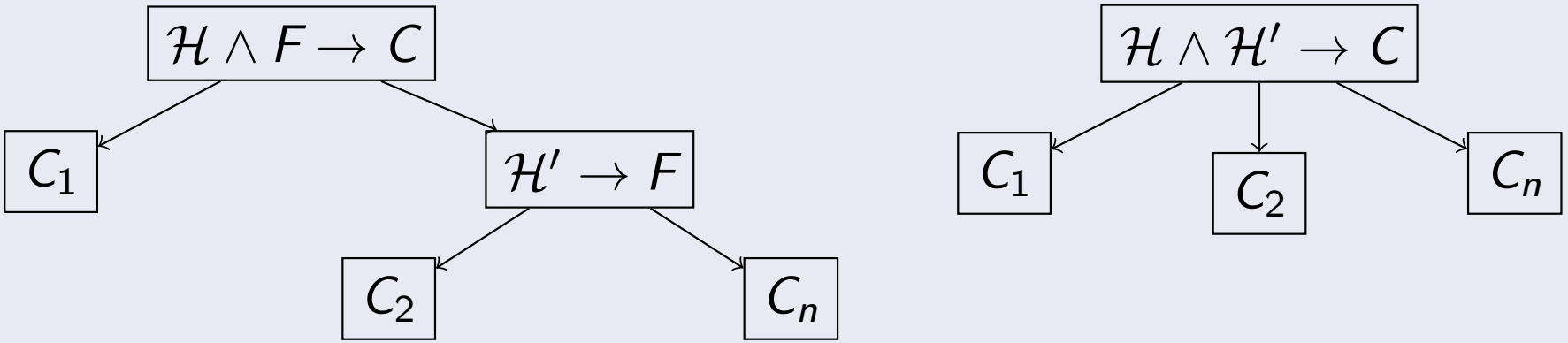
Example

$$\begin{aligned} mess(c, x) &\to mess(d, 0) \\ mess(d, y) &\to mess(c, s) \\ att(c) \wedge att(x) &\to mess(c, x) \end{aligned}$$

$$\begin{aligned} &\to \texttt{mess}(d, 0) \\ &\to mess(c, s) \end{aligned}$$

Saturation and derivation example

Idea: Squish!



Example

$$
\begin{array}{r|l}
mess(c, x) \to mess(d, 0) & \to \texttt{mess}(d, 0) \\
mess(d, y) \to mess(c, s) & \to mess(c, s) \\
att(c) \wedge att(x) \to mess(c, x) &
\end{array}
$$

1. Background
   - Proverif
   - Saturation Procedure
   - Reconstruction of attacks

2. Contribution
   - Improving reconstruction of attacks
   - Other optimizations
     - Matching replication occurrences
     - Smarter Let constructs

When reconstruction fails

Consider: $P = \begin{array}{l} \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\ \mid \mathtt{out}(d, s); \mathtt{out}(c, s) \end{array}$

When reconstruction fails

Consider: $P = \begin{array}{l} \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\ \mid \mathtt{out}(d, s); \mathtt{out}(c, s) \end{array}$

Demo Time!

When reconstruction fails

Consider: $P = \ \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$
$\quad\quad\quad\quad \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$

$\rightarrow \mathtt{mess}(d, s)$

When reconstruction fails

Consider: $P = \ \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$
$\quad\quad\quad\quad \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

When reconstruction fails

Consider: $P = \ \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$
$\qquad\quad \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

$\mathtt{mess}(c, x)$

When reconstruction fails

Consider: $P = \ \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$
$\mid \mathtt{out}(d, s); \mathtt{out}(c, s)$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

$\mathtt{mess}(c, x) \wedge \mathtt{mess}(d, y)$

When reconstruction fails

Consider: $P = \begin{array}{l} \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\ \mid \mathtt{out}(d, s); \mathtt{out}(c, s) \end{array}$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

$\mathtt{mess}(c, x) \wedge \mathtt{mess}(d, y) \rightarrow ?$

When reconstruction fails

Consider: $P = \begin{array}{l} \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\ \mid \mathtt{out}(d, s); \mathtt{out}(c, s) \end{array}$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

$\mathtt{mess}(c, x) \wedge \mathtt{mess}(d, y) \rightarrow ?$

Loss of info!

When reconstruction fails

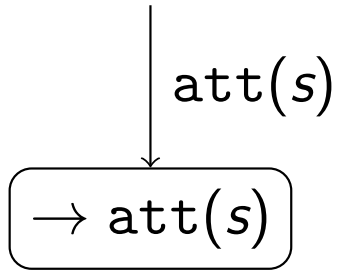
Consider: $P = \begin{array}{l} \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\ \mid \mathtt{out}(d, s); \mathtt{out}(c, s) \end{array}$

$\rightarrow \mathtt{mess}(d, s)$

$\rightarrow \mathtt{att}(s)$

$\mathtt{mess}(c, x) \wedge \mathtt{mess}(d, y) \rightarrow ?$

Loss of info!



What's happening

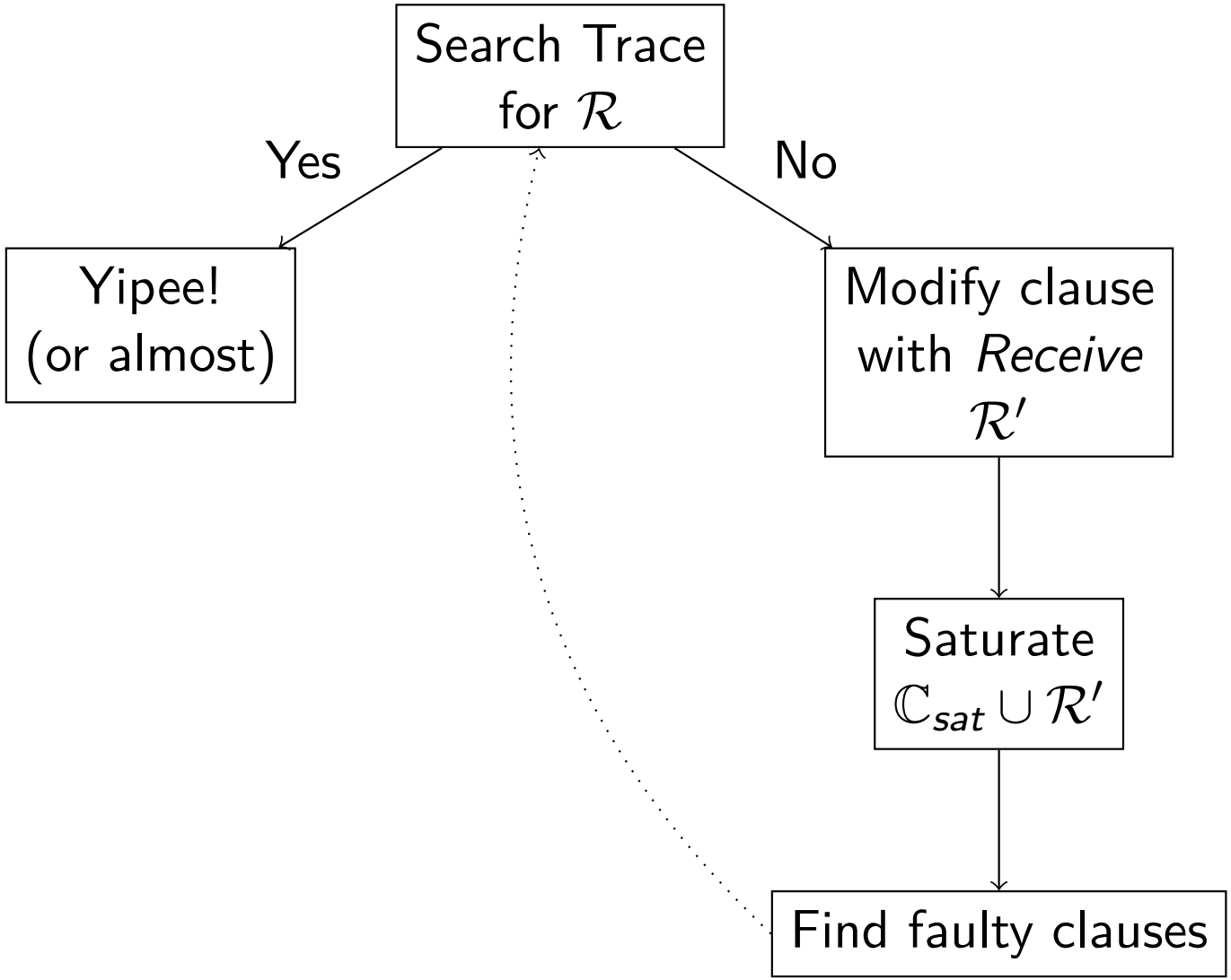
Why does it sometimes work?

- Current implementation not the same as the theory
- Heuristics
- Backtracking

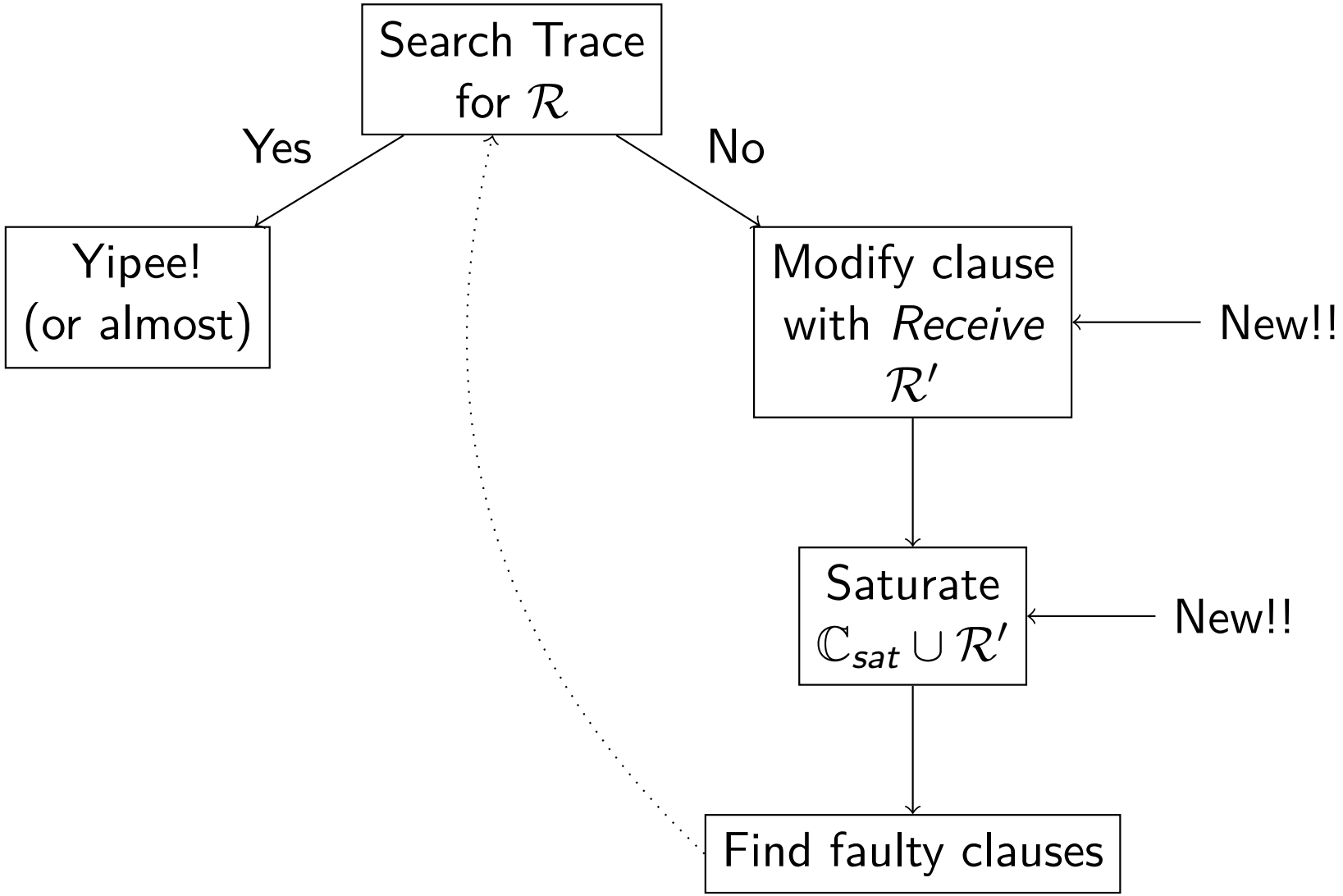
Can we improve it?

- Private sync in derivation
- Consistent
- Less to no backtracking
- *We care*: private channels are everywhere (ex: memory cells)

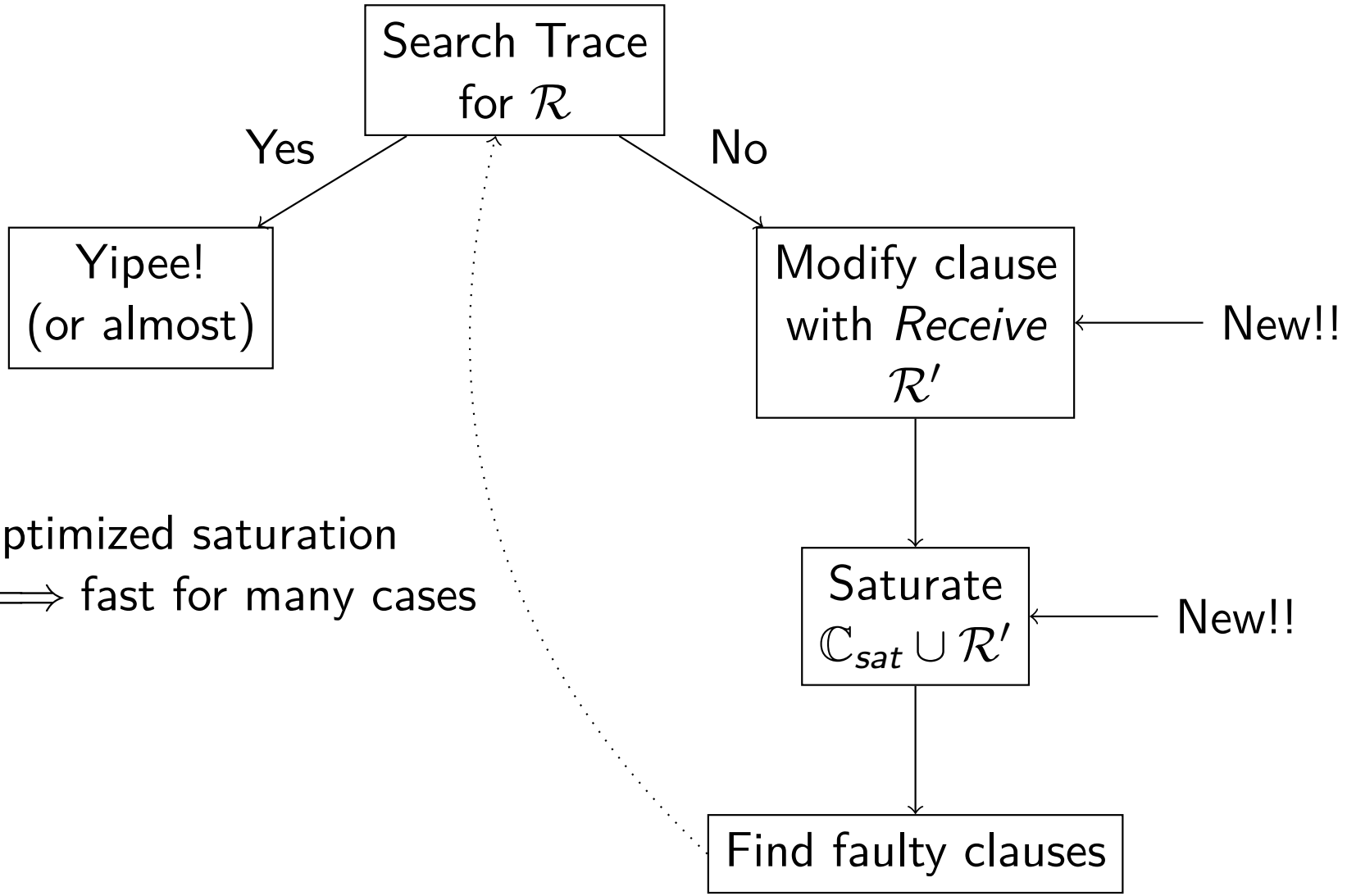
Progressively searching



Progressively searching



Progressively searching



Rely on optimized saturation

Iterative $\Longrightarrow$ fast for many cases

Modifying generation of clauses: *Receive* predicate

Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$P = \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$$

Modifying generation of clauses: *Receive* predicate

Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$P = \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$$
$$\mid \mathtt{out}(d, s); \mathtt{out}(c, s)$$

$$\rightarrow \mathtt{mess}(d, s)$$

$\mathit{Receive}(d, x_o)$

Modifying generation of clauses: *Receive* predicate

Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$P = \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y)$$
$$\mid \mathtt{out}(d, s); \mathtt{out}(c, s)$$

$$\rightarrow \mathtt{mess}(d, s)$$
$$\mathit{Receive}(d, x_o) \rightarrow \mathtt{att}(s)$$

Modifying generation of clauses: *Receive* predicate

Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$P = \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$$

$$\begin{aligned} &\to \mathtt{mess}(d, s) \\ \mathit{Receive}(d, x_o) &\to \mathtt{att}(s) \\ &\to \mathit{Receive}(c, o_1) \\ \mathtt{mess}(c, x) & \end{aligned}$$

Modifying generation of clauses: *Receive* predicate

Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$P = \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \mid \mathtt{out}(d, s); \mathtt{out}(c, s)$$

$$\begin{aligned} &\to \mathtt{mess}(d, s) \\ \mathit{Receive}(d, x_o) &\to \mathtt{att}(s) \\ &\to \mathit{Receive}(c, o_1) \\ \mathtt{mess}(c, x) &\to \mathit{Receive}(d, o_2) \end{aligned}$$

Modifying generation of clauses: *Receive* predicate

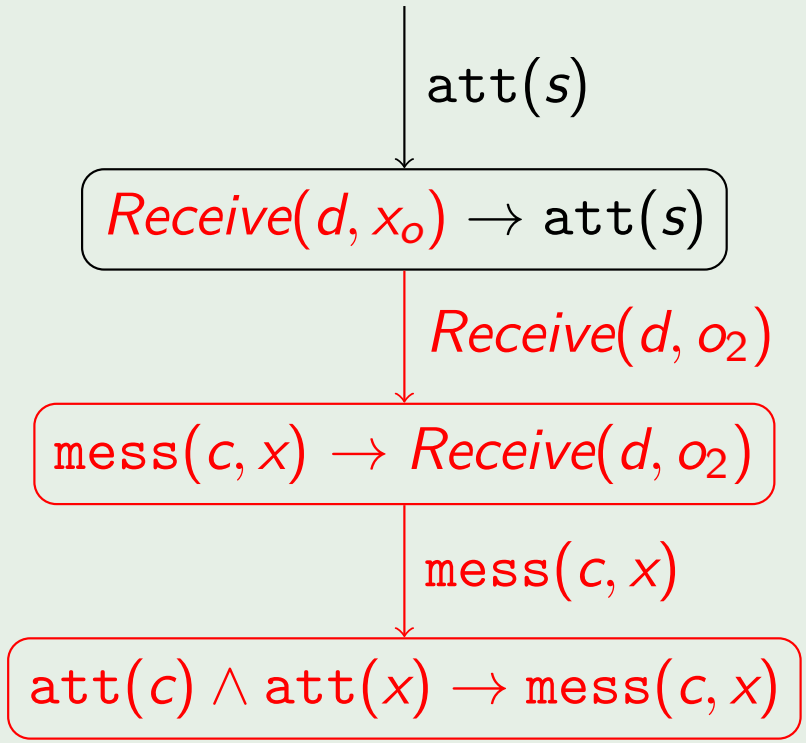
Idea: new predicate

- $\mathit{Receive}(c, O)$: "Message is received on channel c"
- Keep occurrence O that was used.

Example

$$
\begin{aligned}
P = \ & \mathtt{in}^{o_1}(c, x); \mathtt{in}^{o_2}(d, y) \\
& \mid \mathtt{out}(d, s); \mathtt{out}(c, s)
\end{aligned}
$$

$$
\begin{aligned}
& \to \mathtt{mess}(d, s) \\
\mathit{Receive}(d, x_o) & \to \mathtt{att}(s) \\
& \to \mathit{Receive}(c, o_1) \\
\mathtt{mess}(c, x) & \to \mathit{Receive}(d, o_2)
\end{aligned}
$$



Running the algorithm on an example

$$
\begin{aligned}
P = \;& \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\
\mid\; & \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\
\mid\; & \mathtt{in}^{o_5}(d, x)
\end{aligned}
$$

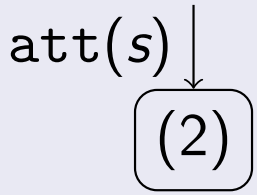
Running the algorithm on an example

$$\begin{aligned} P = \ & \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ | \ & \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ | \ & \mathtt{in}^{o_5}(d, x) \end{aligned}$$

Initial call

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



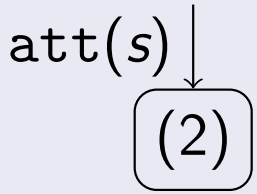
Running the algorithm on an example

$$P = \begin{array}{l} \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ \mid \mathtt{in}^{o_5}(d, x) \end{array}$$

First iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



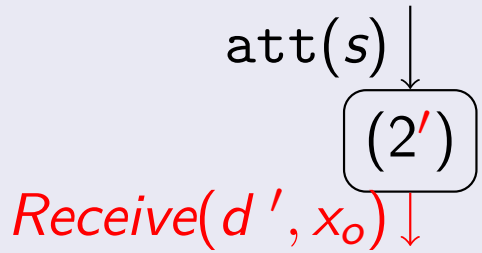
Running the algorithm on an example

$$P = \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \mid \mathtt{in}^{o_5}(d, x)$$

First iteration

$$\mathcal{R}' = \mathit{Receive}(d', x_o) \to \mathtt{att}(s)$$

$$\begin{array}{rll} & \to \mathtt{mess}(d', s) & (1) \\ \mathit{Receive}(d', x_o) & \to \mathtt{att}(s) & (2') \\ & \to \mathtt{mess}(d, s) & (3) \\ & \to \mathit{Receive}(d, o_5) & (4) \\ & \to \mathit{Receive}(d', o_4) & (5) \end{array}$$



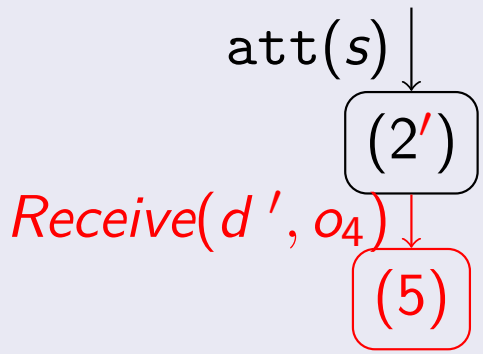
Running the algorithm on an example

$$P = \begin{array}{l} \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ \mid \mathtt{in}^{o_5}(d, x) \end{array}$$

First iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



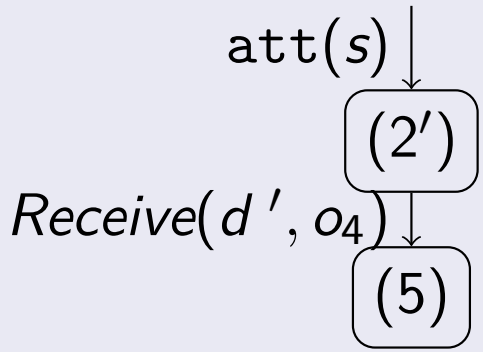
Running the algorithm on an example

$$\begin{aligned} P = \ & \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ | \ & \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ | \ & \mathtt{in}^{o_5}(d, x) \end{aligned}$$

Second iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



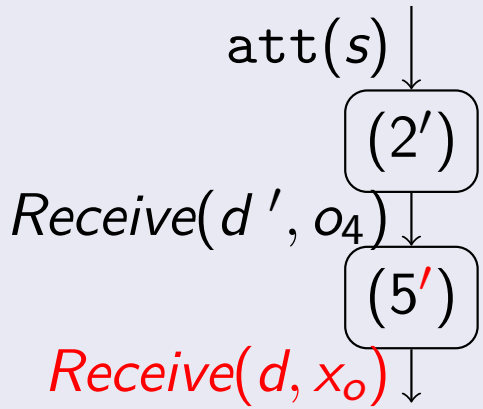
Running the algorithm on an example

$$P = \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \mid \mathtt{in}^{o_5}(d, x)$$

Second iteration

$$\mathcal{R}' = \mathit{Receive}(d, x_o) \rightarrow \mathtt{att}(s)$$

$$\begin{aligned}
&\rightarrow \mathtt{mess}(d', s) && (1)\\
&\rightarrow \mathtt{att}(s) && (2)\\
&\rightarrow \mathtt{mess}(d, s) && (3)\\
&\rightarrow \mathit{Receive}(d, o_5) && (4)\\
\mathit{Receive}(d, x_o) &\rightarrow \mathit{Receive}(d', o_4) && (5')
\end{aligned}$$



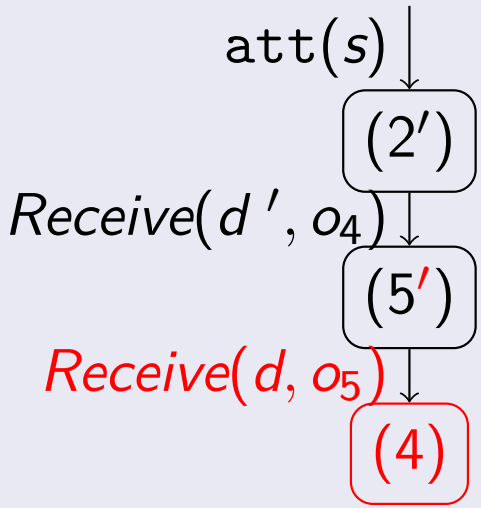
Running the algorithm on an example

$$P = \begin{array}{l} \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ \mid \mathtt{in}^{o_5}(d, x) \end{array}$$

Second iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



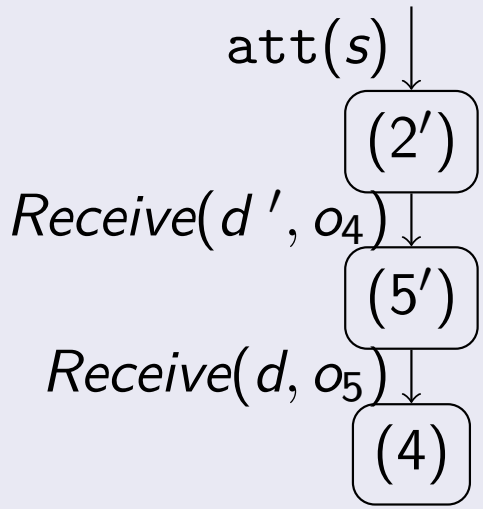
Running the algorithm on an example

$$P = \begin{array}{l} \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ \mid \mathtt{in}^{o_5}(d, x) \end{array}$$

Third iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



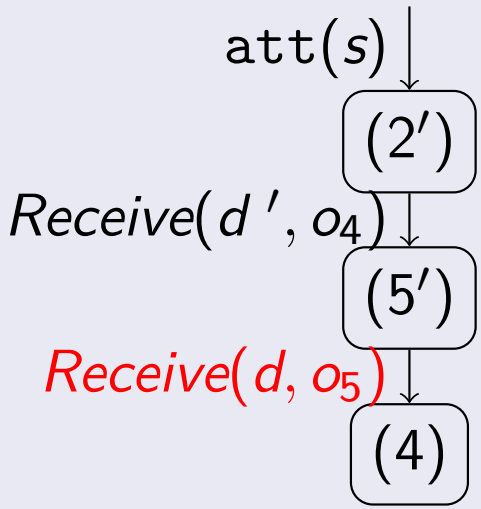
Running the algorithm on an example

$$P = \begin{array}{l} \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \\ \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \\ \mid \mathtt{in}^{o_5}(d, x) \end{array}$$

Third iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



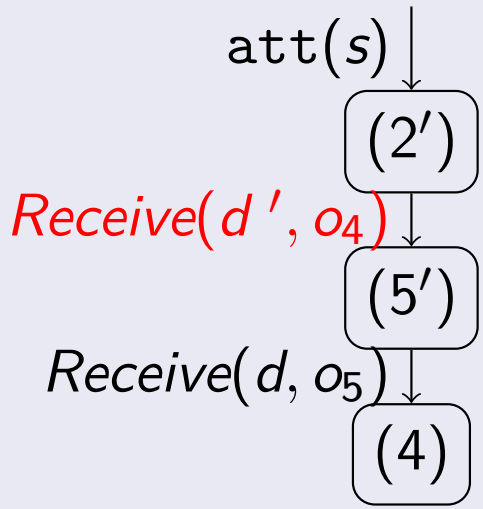
Running the algorithm on an example

$$P = \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \mid \mathtt{in}^{o_5}(d, x)$$

Third iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



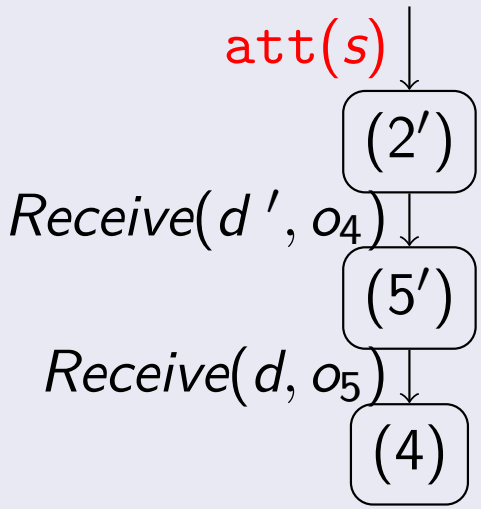
Running the algorithm on an example

$$P = \mathtt{out}^{o_1}(d', s);\ \mathtt{out}^{o_2}(c, s) \mid \mathtt{out}^{o_3}(d, s);\ \mathtt{in}^{o_4}(d', x) \mid \mathtt{in}^{o_5}(d, x)$$

Third iteration

$$\mathcal{R} = \rightarrow \mathtt{att}(s)$$

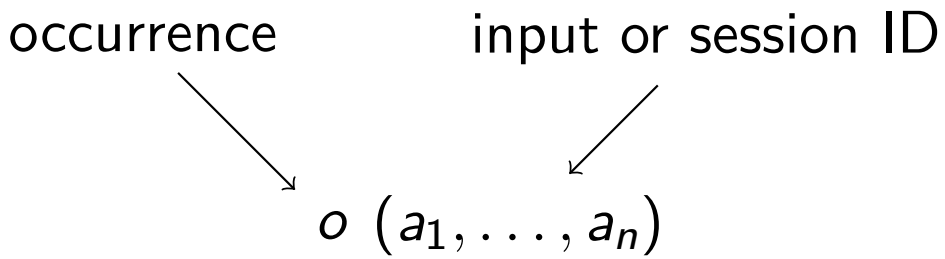
$\rightarrow \mathtt{mess}(d', s)$	(1)
$\rightarrow \mathtt{att}(s)$	(2)
$\rightarrow \mathtt{mess}(d, s)$	(3)
$\rightarrow \mathit{Receive}(d, o_5)$	(4)
$\rightarrow \mathit{Receive}(d', o_4)$	(5)



1. Background
   - Proverif
   - Saturation Procedure
   - Reconstruction of attacks

2. Contribution
   - Improving reconstruction of attacks
   - Other optimizations
     - Matching replication occurrences
     - Smarter Let constructs

Occurrences and instrumentalisation

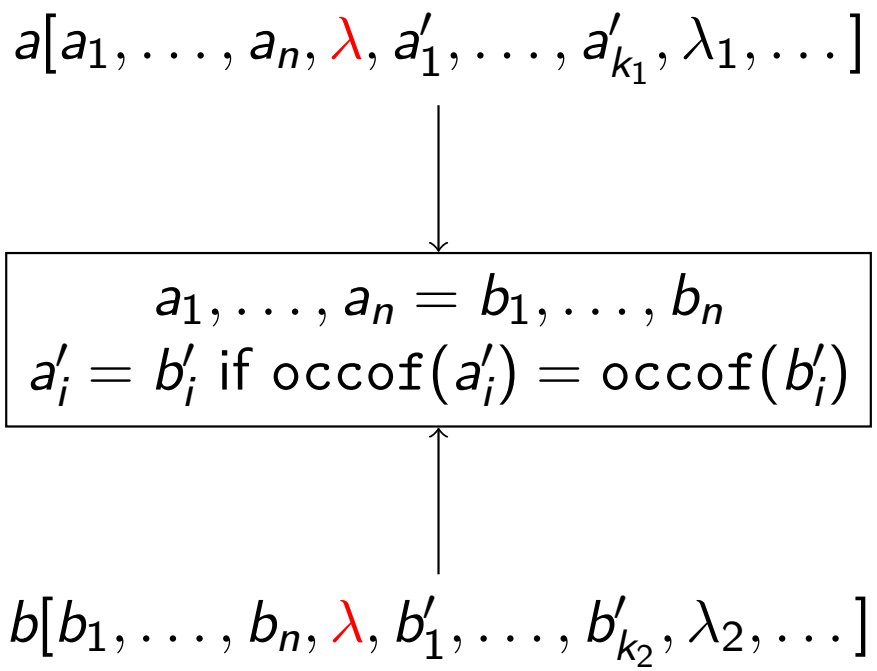


Example

$$P = !\ \mathtt{in}(c, x); \mathtt{new}\ a; \mathtt{out}(c, a)$$

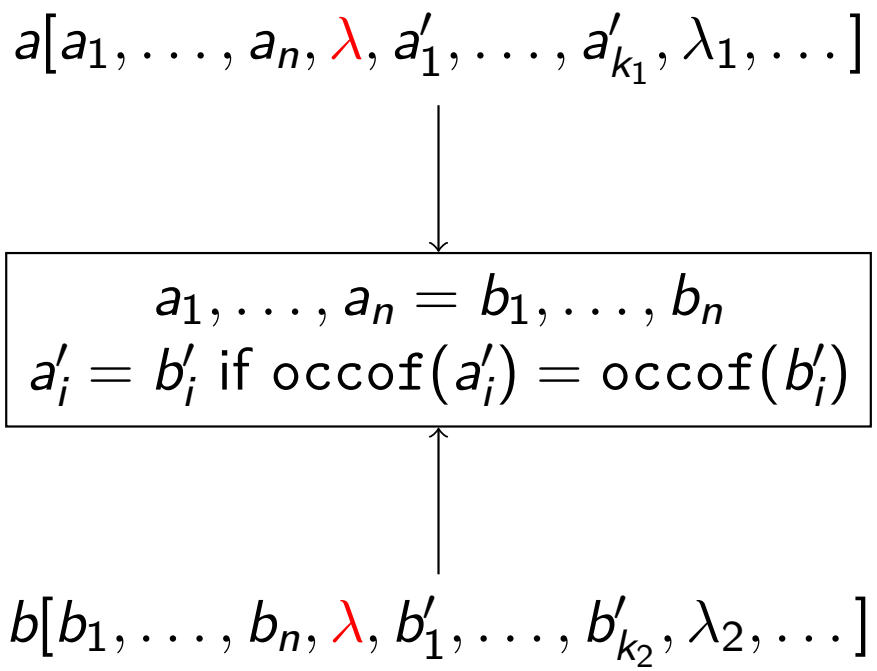
$$P = !_i^{o_1(i)}\ \mathtt{in}^{o_2(i)}(c, x); \mathtt{out}(c, a[i, x])$$

Matching replication occurrences



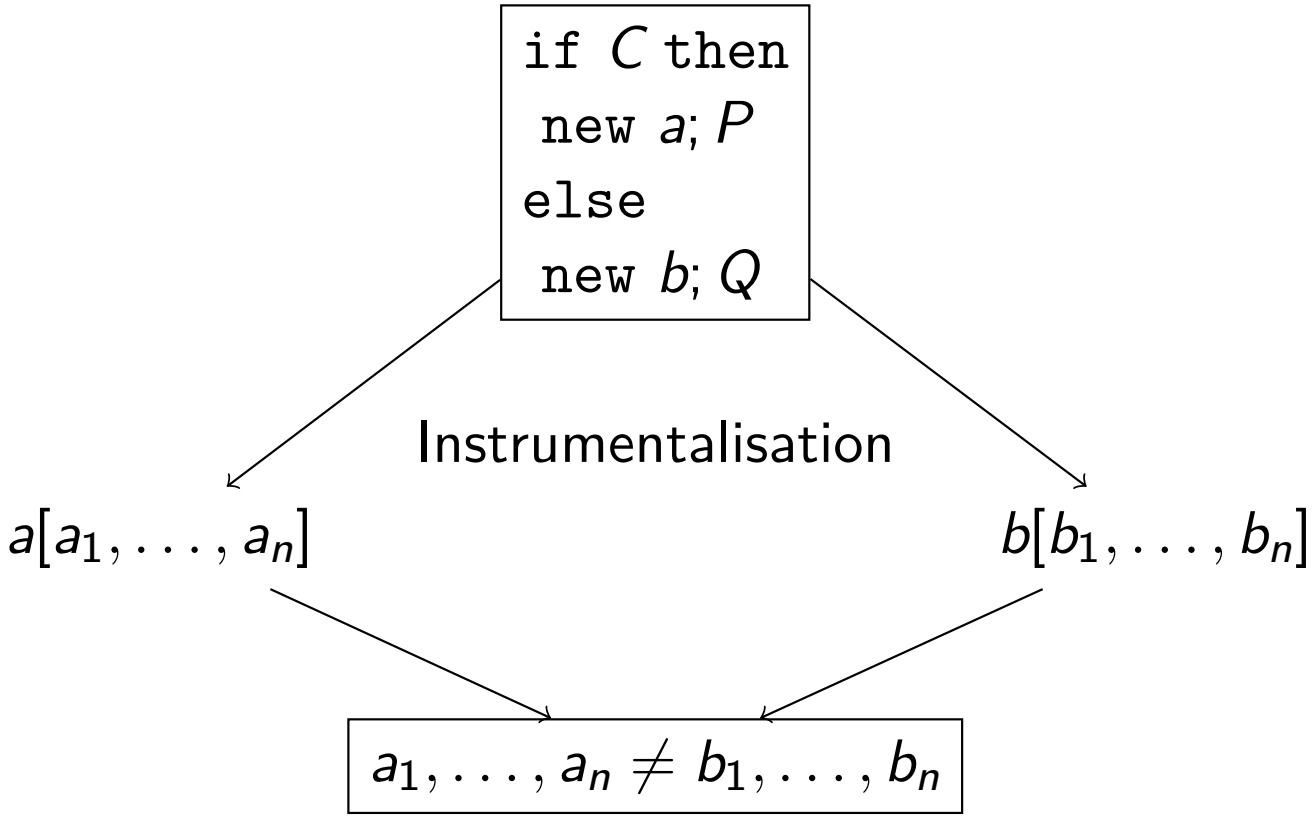
- Used throughout saturation and derivation

Matching replication occurrences



- Used throughout saturation and derivation
- Like `precise`?

Smarter Let constructs



Conclusion

- Private Channel synchronization
- New algorithm structure with incremental search
- Modernized old theory (from 2005)
- Miscellaneous optimizations
- Future Work: finishing prototype