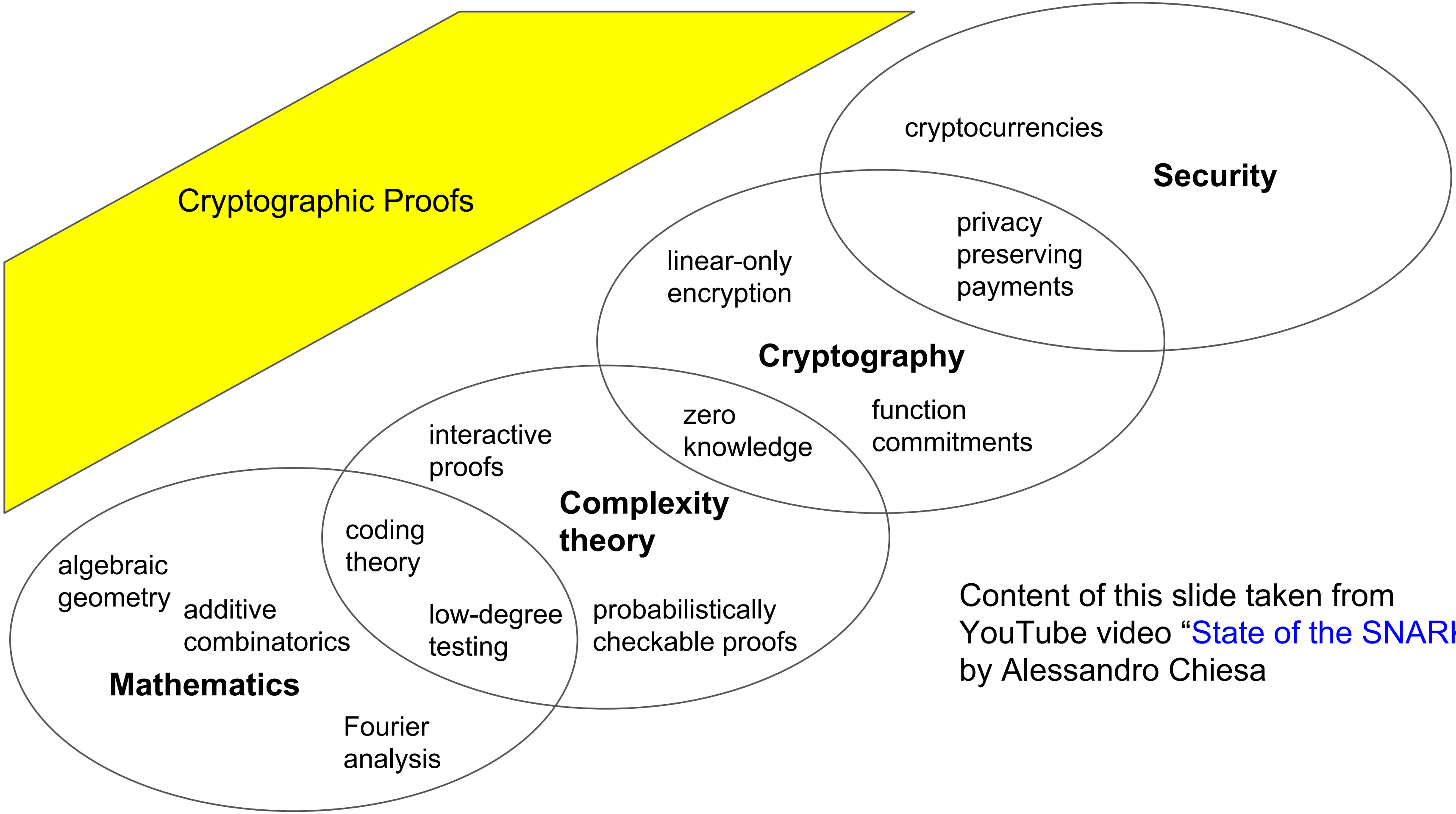
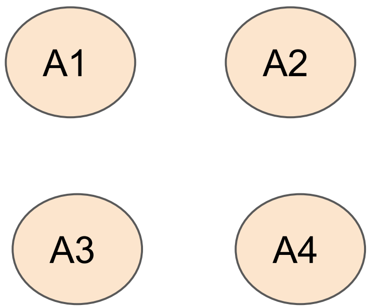
How to prove you solved a puzzle without showing how

Dr Alexey Akhunov, Nov 2016

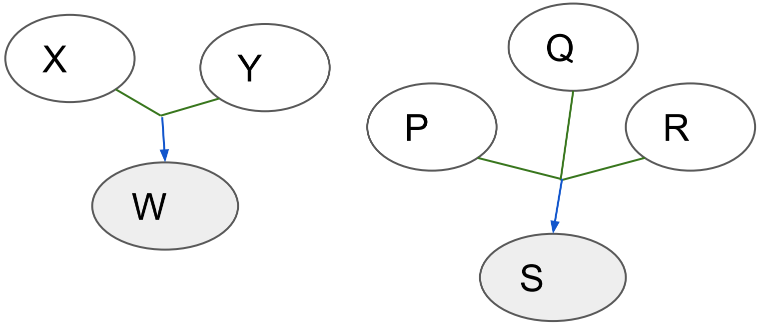


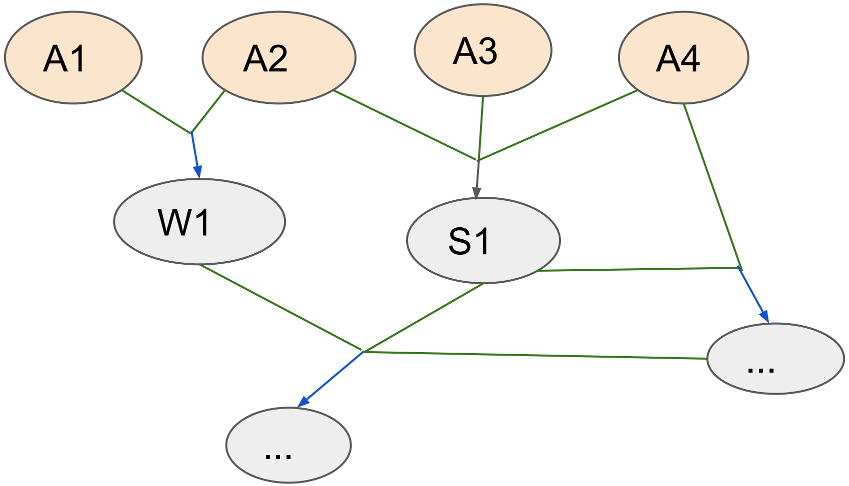
Content of this slide taken from YouTube video "State of the SNARK" by Alessandro Chiesa

Axioms

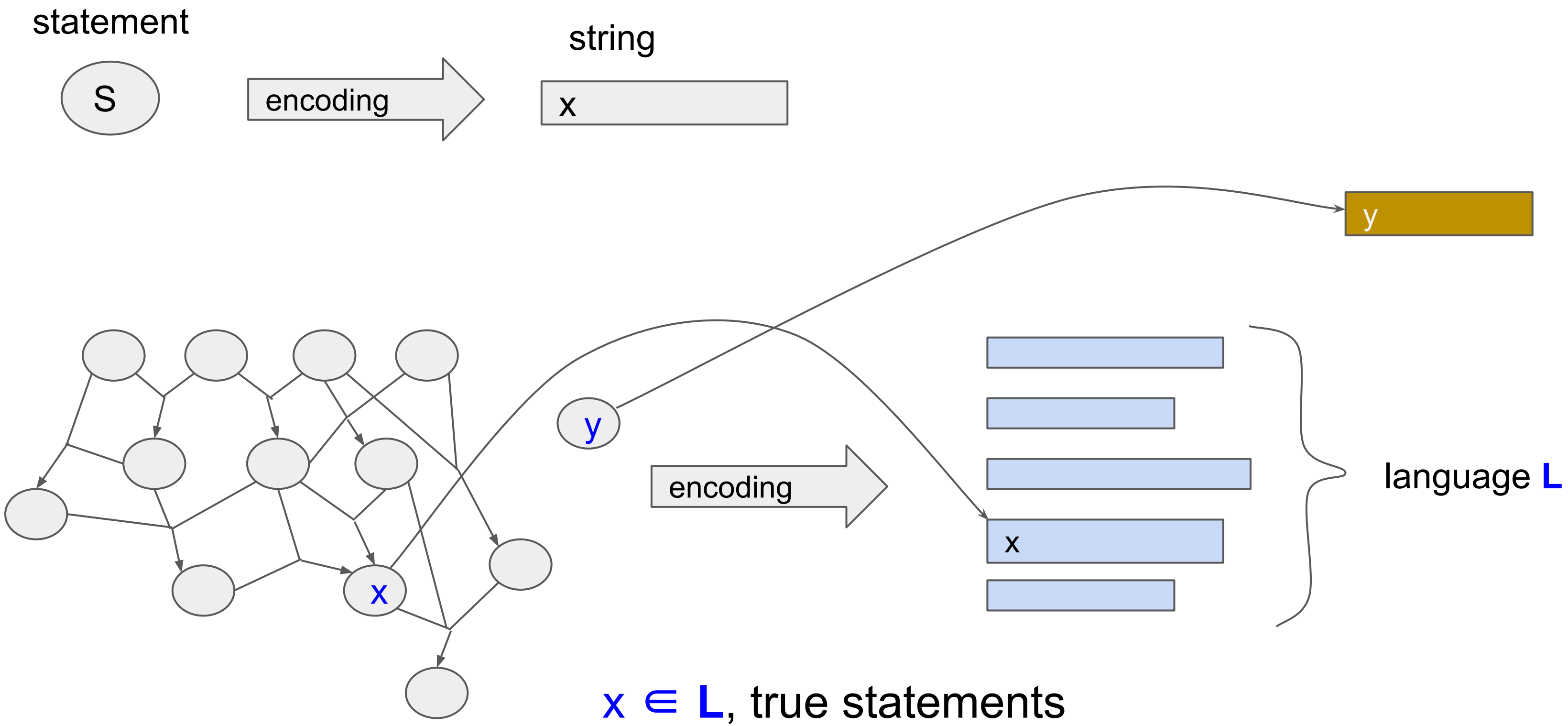


Rules

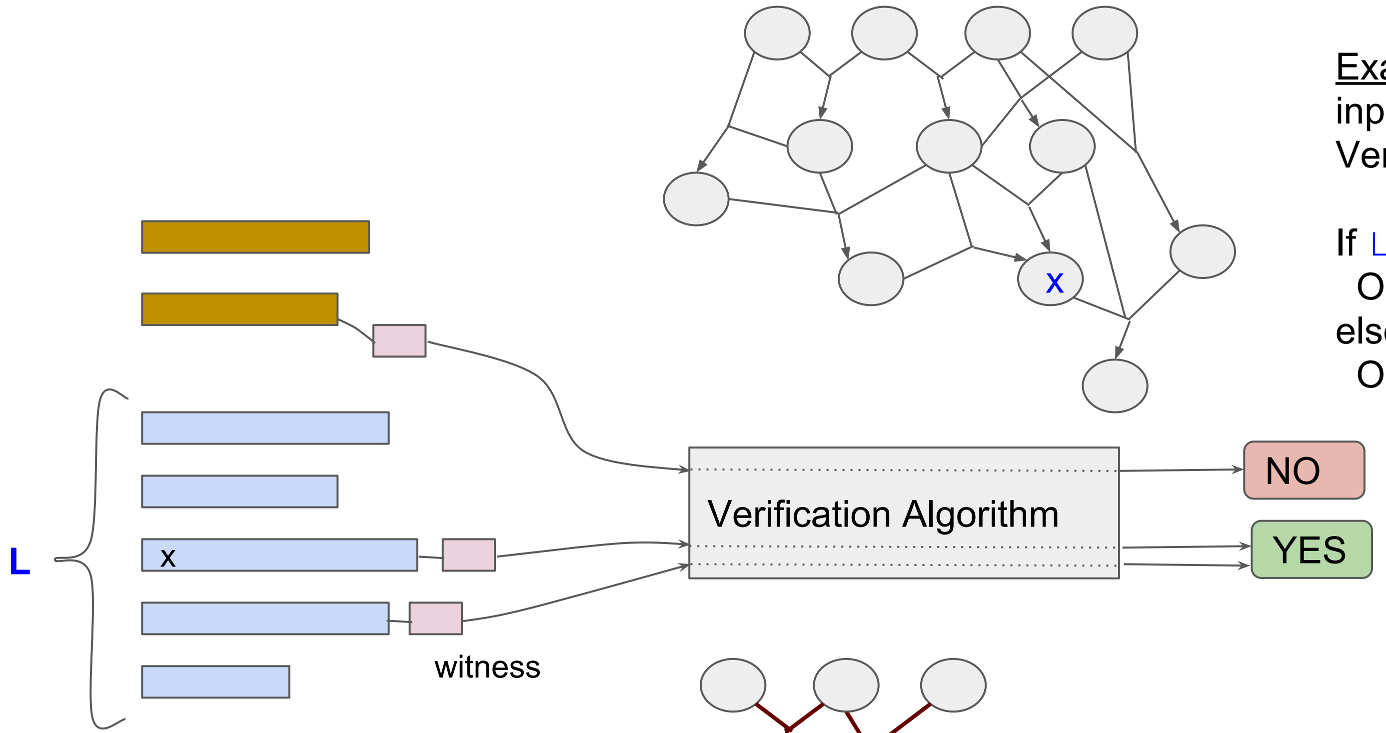


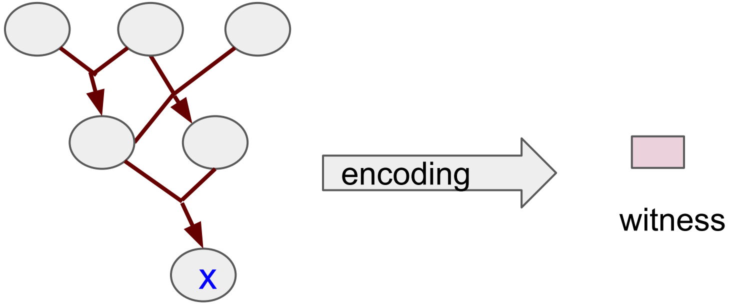


Can derive potentially infinite number of statements



$x \in$ **L**, true statements
$y \notin$ **L**, other statements





<u>Example</u>:
input x, witness a,
Verification Algorithm:

If $\lfloor x/a \rfloor * a == x$:
  Output YES
else:
  Output NO

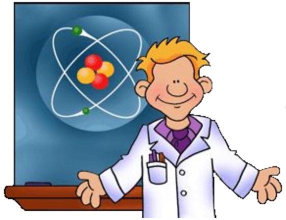
<u>Example</u>: L is language of all composite numbers,
$x \notin L$ means x is prime

Protocol for “classical” proofs

prover

verifier

x



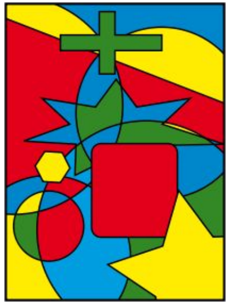
a

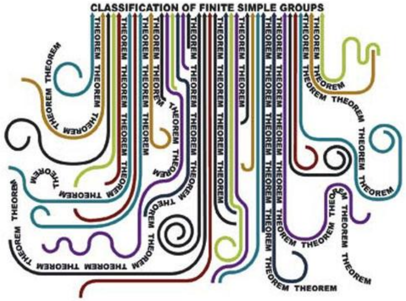
YES

statement

proof=witness

Some proofs are too large





proof ≠ witness?

Another protocol

1535 Mathematical duel Fiorre - Tartaglia

$$x^3 + px + q = 0$$

$$x = \sqrt[3]{-\frac{q}{2} + \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} + \sqrt[3]{-\frac{q}{2} - \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$$

Fiorre

$x^3 + px = q$, $x=?$

Tartaglia

x_s

got more?

yes

no

$x_s^3+px_s=q?$

yes

no

Tartaglia wins

Tartaglia loses

What we want to the proofs to be

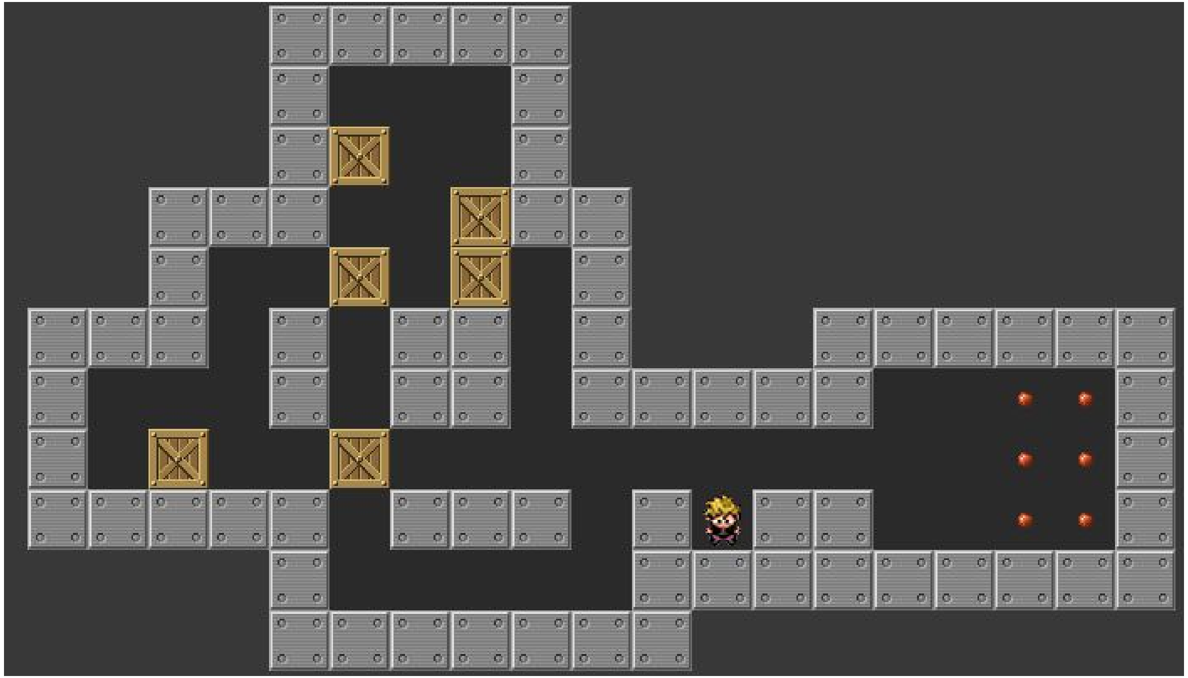
Succinct. Verification time/memory does not depend on the problem size. Allows verifying very large proofs

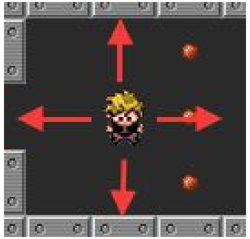
Non-interactive. We do not need a dedicated verifier, anybody can verify any time.

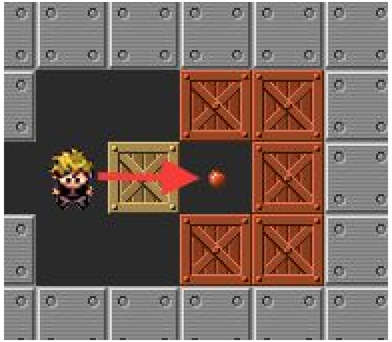
Zero-knowledge. If prover wants to keep the witness secret. Someone who has seen the proof and accepted it, does not learn anything new about the witness.

Proof of knowledge. We want to be sure that the prover must have known the secret if they were able to produce the proof.

Toy example - Sokoban (Japan, early 80s)

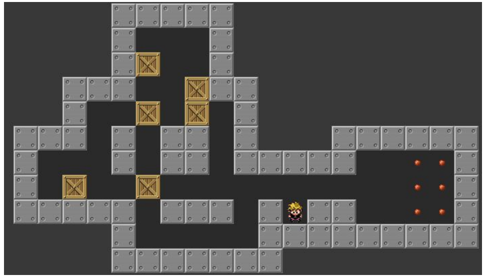


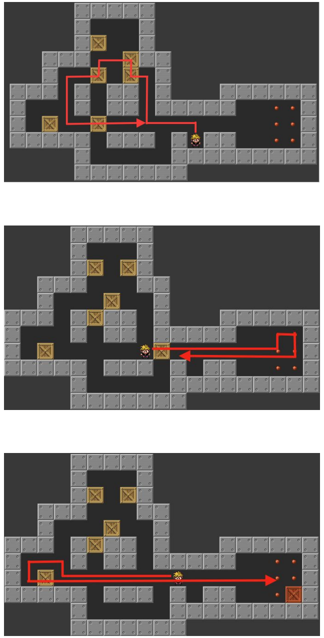




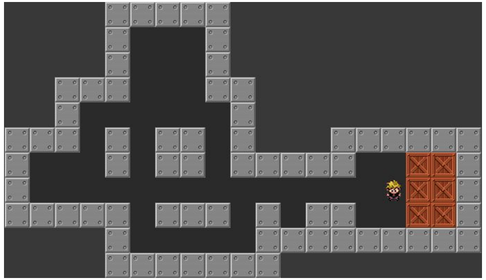
NP-hard problem

initial state





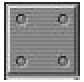
...

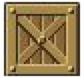


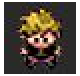
Formalisation

m(ovable), s(olid), v(ertical), n(orthwest) $\in \{0,1\}$

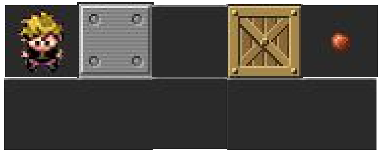
For board of size (2, 5), the state is ($M=[m_{ij}]$, $S=[s_{ij}]$), $0 \le i < 2$, $0 \le j < 5$, m_{ij}, $s_{ij} \in \{0,1\}$.

!m ∧ s 

m ∧ s 

m ∧ !s 

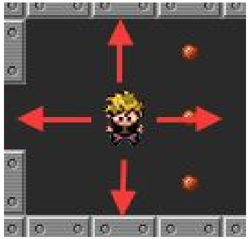
!m ∧ !s 



$$M = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

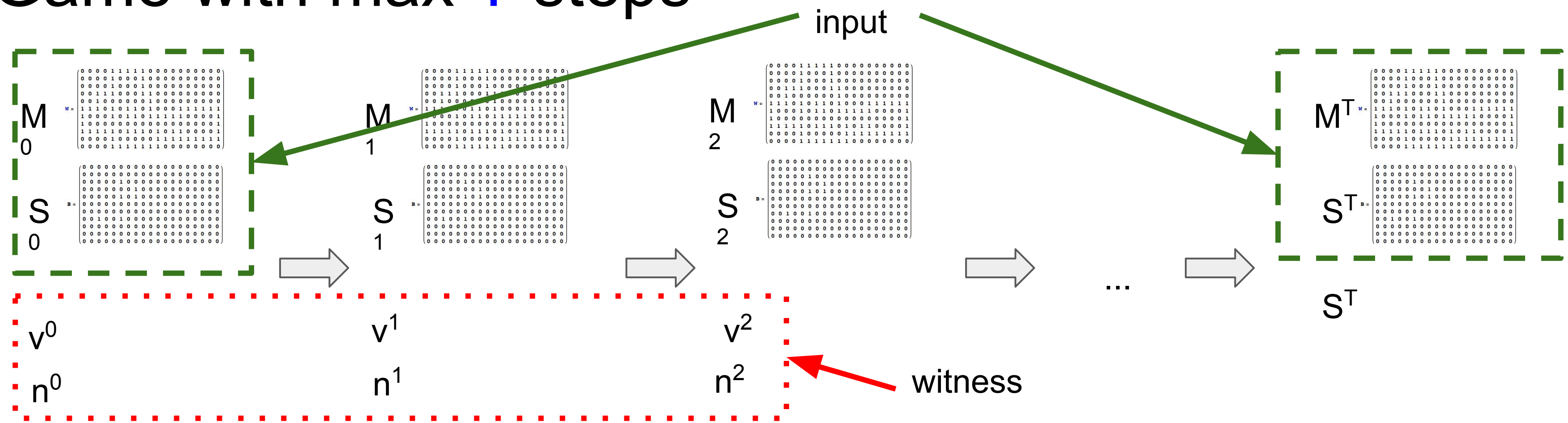
$$S = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

v ∧ n

!v ∧ n  !v ∧ !n

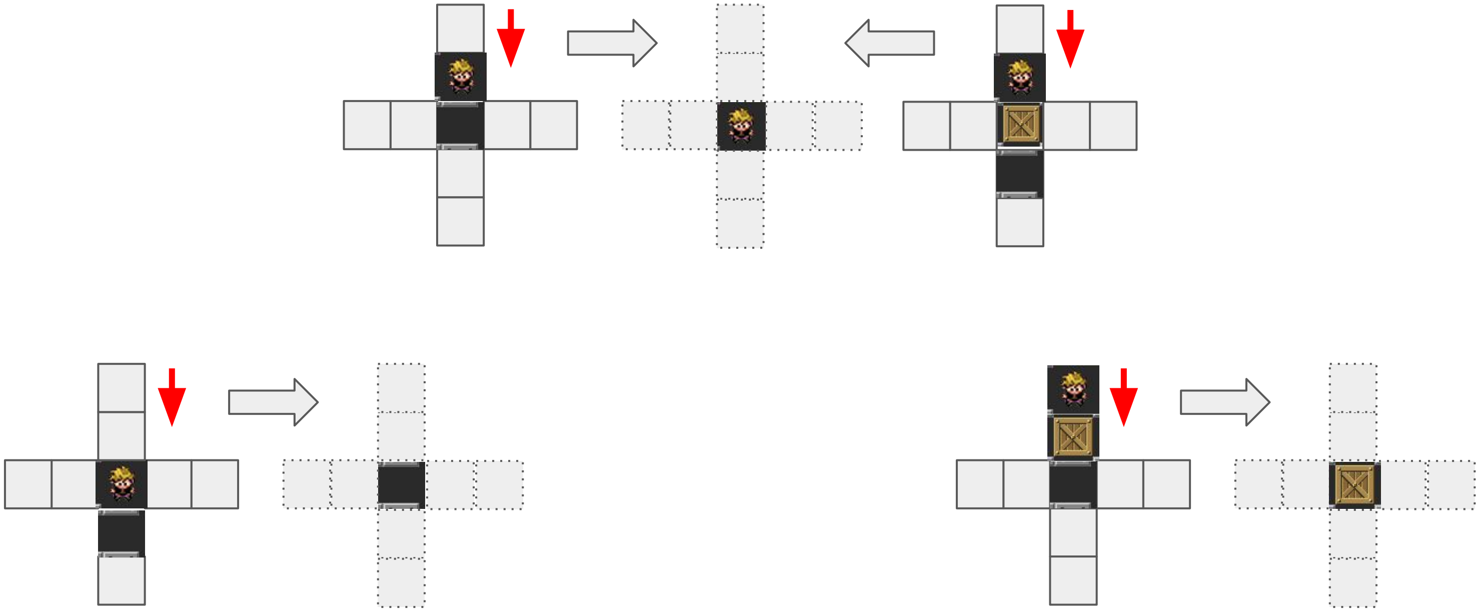
v ∧ !n

Game with max T steps

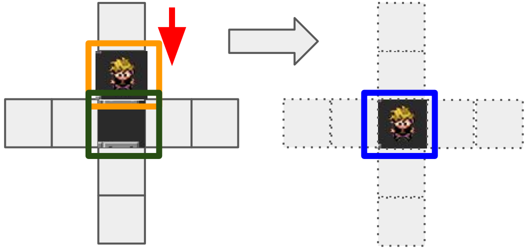


step t to step t+1: $(M^t, S^t, v^t, n^t) \rightarrow (M^{t+1}, S^{t+1})$

Observation: state of the any cell can only depend on the state of 9 cells from the previous step, and on the movement variables. The only 4 ways the state can change (same for all other movement directions):

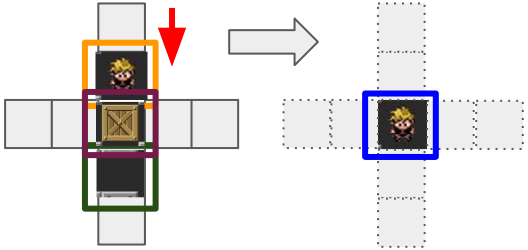


$m_{i-1,j}^{t} \wedge !s_{i-1,j}^{t} \wedge !m_{i,j}^{t} \wedge\ !s_{i,j}^{t} \wedge v^{t} \wedge !n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1}$



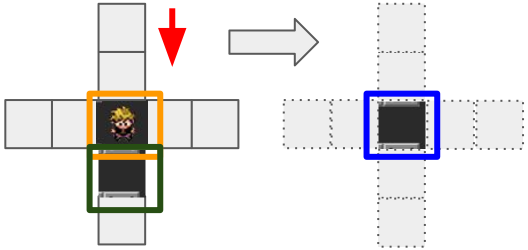
… and 3 more expressions like that for moving up, right, and left ...

$m_{i-1,j}^{t} \wedge !s_{i-1,j}^{t} \wedge m_{i,j}^{t} \wedge s_{i,j}^{t} \wedge !m_{i+1,j}^{t} \wedge\ !s_{i+1,j}^{t} \wedge v^{t} \wedge !n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1}$



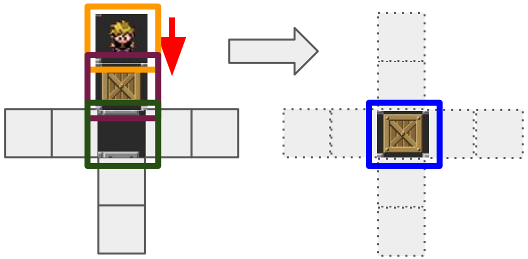
… and 3 more expressions like that for moving up, right, and left ...

$$m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !m_{i+1,j}^{t} \wedge !s_{i+1,j}^{t} \wedge v^{t} \wedge !n^{t} \wedge !m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1}$$



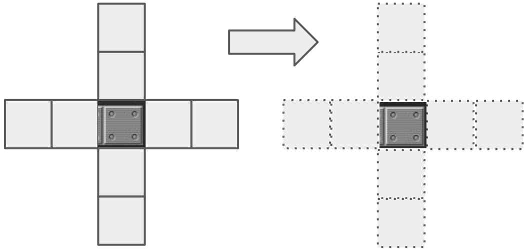
… and 3 more expressions like that for moving up, right, and left ...

$$m_{i-2,j}^{t} \wedge !s_{i-2,j}^{t} \wedge m_{i-1,j}^{t} \wedge s_{i-1,j}^{t} \wedge !m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !v^{t} \wedge n^{t} \wedge m_{i,j}^{t+1} \wedge s_{i,j}^{t+1}$$

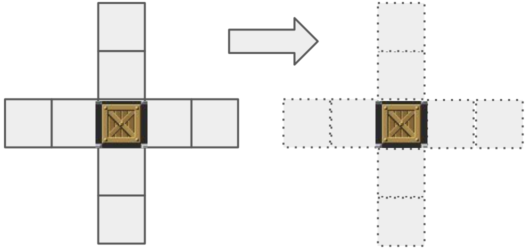


… and 3 more expressions like that for moving up, right, and left ...

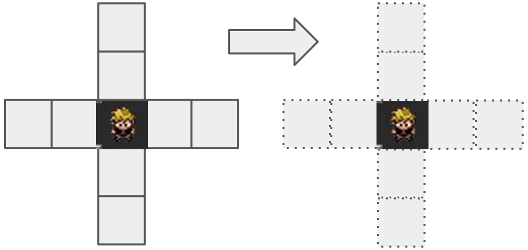
$!m_{i,j}^{t} \wedge s_{i,j}^{t} \wedge !m_{i,j}^{t+1} \wedge s_{i,j}^{t+1}$



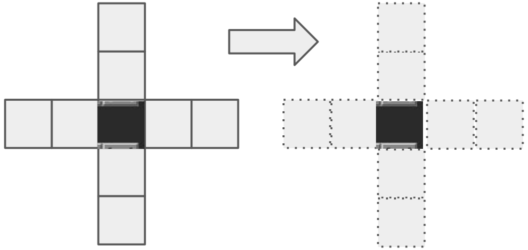
$m_{i,j}^{t} \wedge s_{i,j}^{t} \wedge m_{i,j}^{t+1} \wedge s_{i,j}^{t+1}$



$m_{i,j}^{t} \wedge !s_{i,j}^{t} m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1}$



$!m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !m_{i,j} \wedge !s_{i,j}^{t+1}$



Cell expression (no edge, no corner)

$(m_{i-1,j}^{t} \wedge !s_{i-1,j}^{t} \wedge !m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge v^{t} \wedge !n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$ $(m_{i+1,j}^{t} \wedge !s_{i+1,j}^{t} \wedge !m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge v^{t} \wedge n_{i,j}^{t+1} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$

$(m_{i,j-1}^{t} \wedge !s_{i,j-1}^{t} \wedge !m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !v^{t} \wedge !n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$ $(m_{i,j+1}^{t} \wedge !s_{i,j+1}^{t} \wedge !m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !v^{t} \wedge n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$

$(m_{i-1,j}^{t} \wedge !s_{i-1,j}^{t} \wedge m_{i,j}^{t} \wedge s_{i,j}^{t} \wedge !m_{i+1,j}^{t} \wedge !s_{i+1,j}^{t} \wedge v^{t} \wedge !n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$ $(m_{i+1,j}^{t} \wedge !s_{i+1,j}^{t} \wedge m_{i,j}^{t} \wedge s_{i,j}^{t} \wedge !m_{i-1,j}^{t} \wedge !s_{i-1,j}^{t} \wedge v^{t} \wedge n^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$

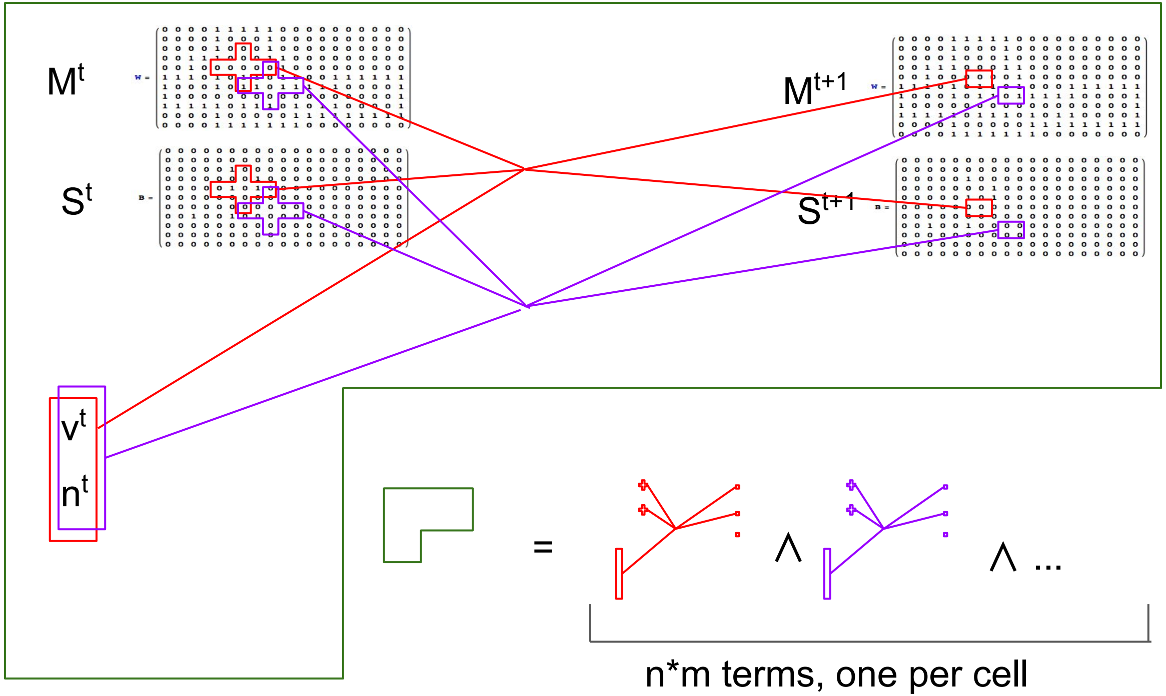
……

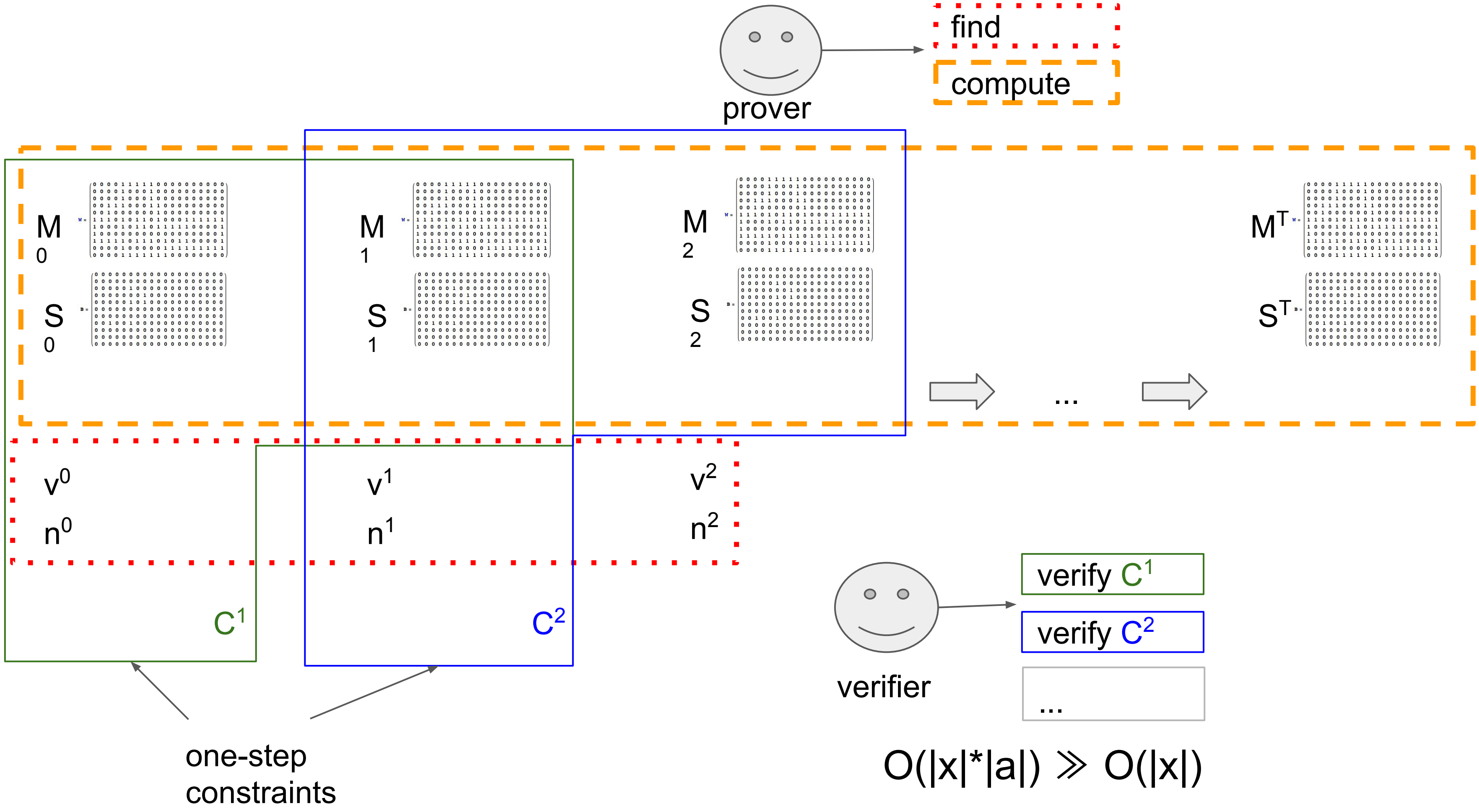
$(m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$ $\vee$ $(!m_{i,j}^{t} \wedge !s_{i,j}^{t} \wedge !m_{i,j}^{t+1} \wedge !s_{i,j}^{t+1})$

each cell expression binds up to 2*(9+1)+2=22 variables together

Step constraint

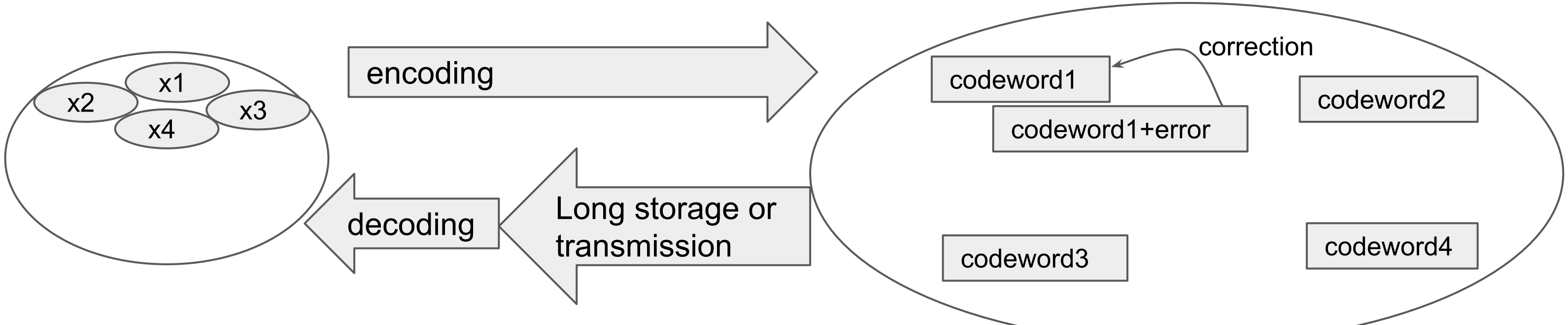
There are n*m cell expressions, and their conjunction (∧) binds 4*n*m+2 variables



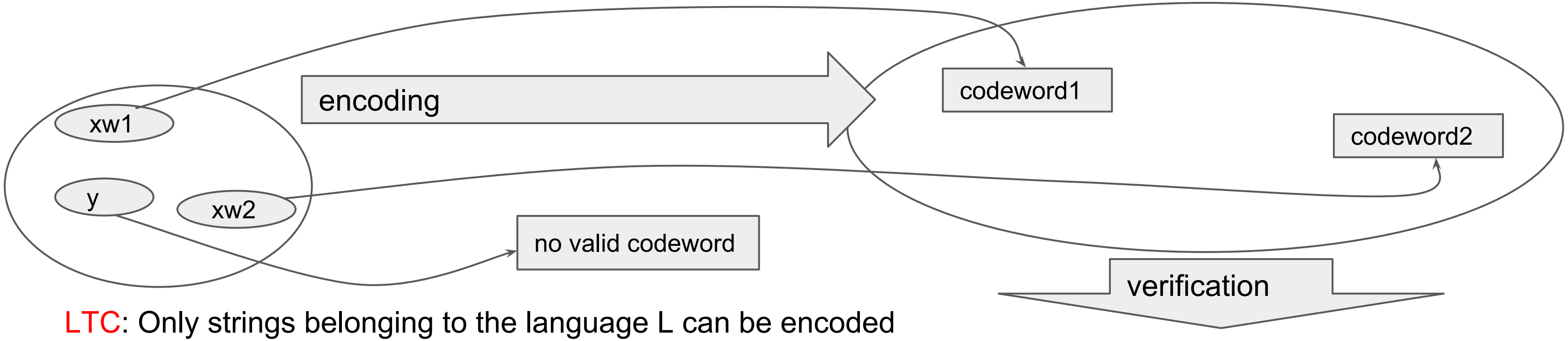


prover
find
compute
M 0
S 0
M 1
S 1
M 2
S 2
...
M T
S T
v0
n0
v1
n1
v2
n2
C1
C2
verifier
verify C1
verify C2
...
one-step constraints
O(|x|*|a|) ≫ O(|x|)

Error correcting and locally testable codes



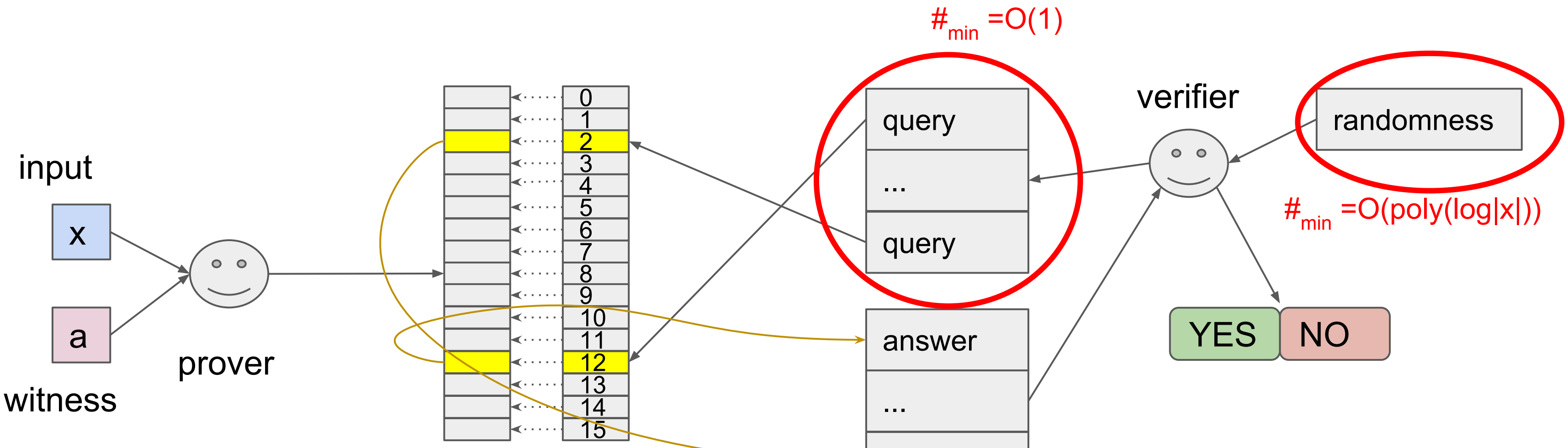
ECC: All messages can be encoded. Noise errors are corrected



LTC: Only strings belonging to the language L can be encoded (with their witness). Deliberate errors are amplified

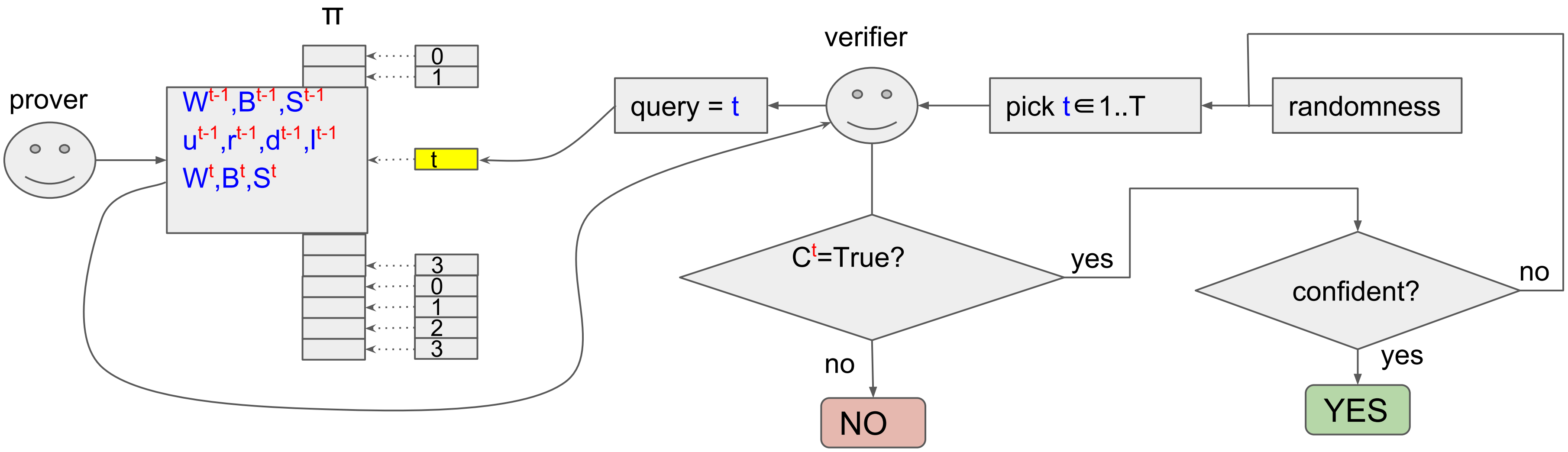
$\color{blue}{P}$robabilistically $\color{blue}{C}$heckable $\color{blue}{P}$roofs

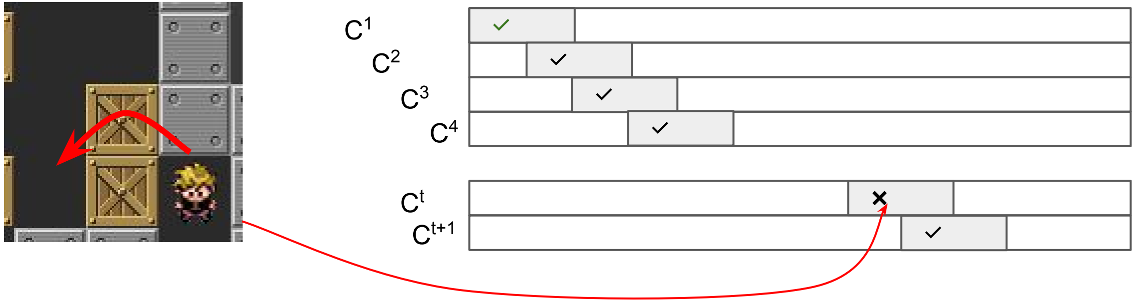
PCP Theorem gives the minimal amount of randomness and number of queries required for this proof system to be complete (If $x \in L$, Prob[YES] = 1) and sound (If $x \notin L$, Prob[NO] $\geq$ ½).



π (codeword). Trying to prove it is a valid codeword

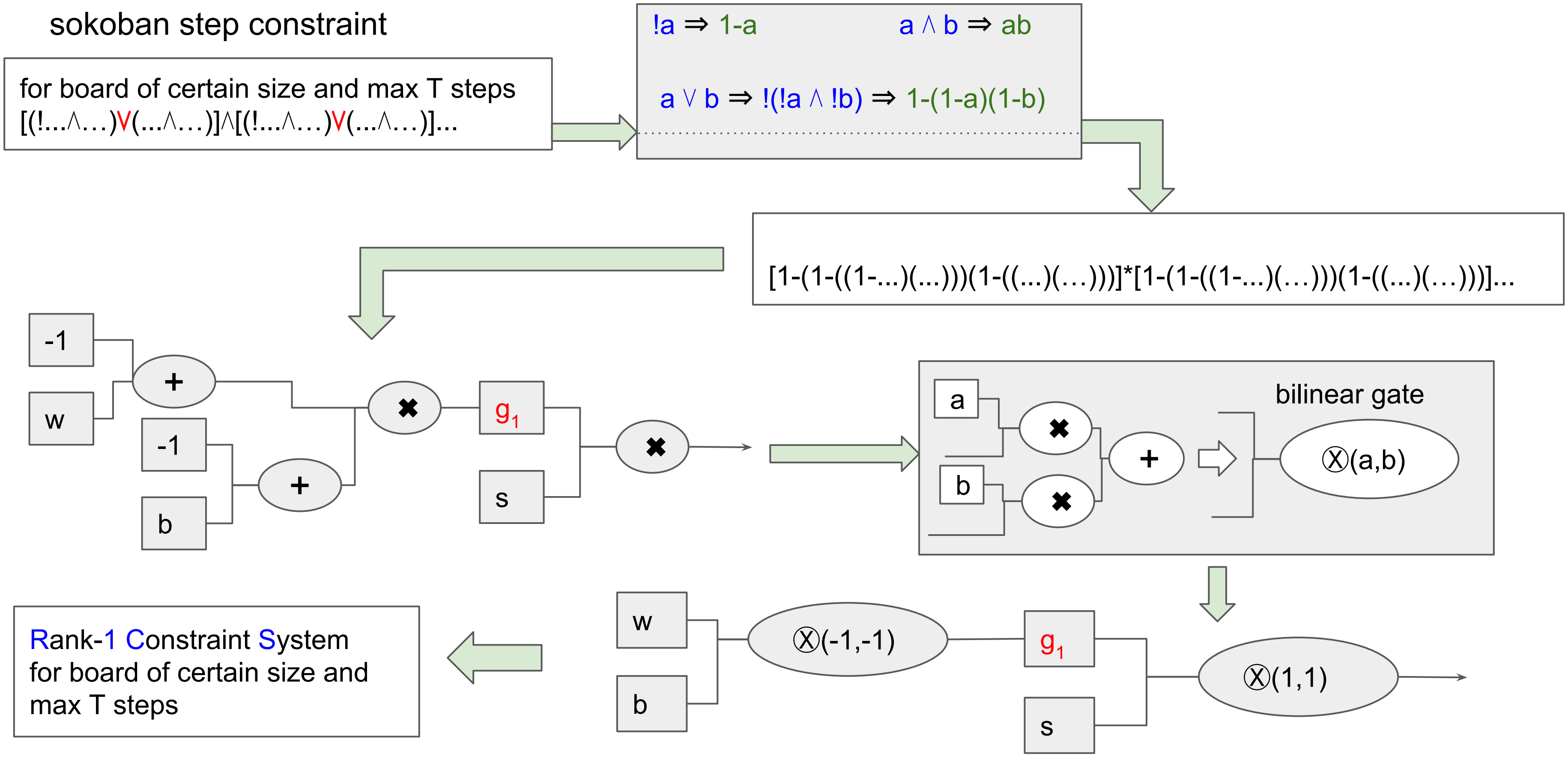
Naive PCP for Sokoban





Prob[NO] ≥ 1/2

#queries = (T+1)/2



sokoban step constraint
for board of certain size and max T steps
[(!...∧…)∨(...∧…)]∧[(!...∧…)∨(...∧…)]...
!a ⇒ 1-a
a ∧ b ⇒ ab
a ∨ b ⇒ !(!a ∧ !b) ⇒ 1-(1-a)(1-b)
[1-(1-((1-...)(...)))(1-((...)(…)))]*[1-(1-((1-...)(…)))(1-((...)(…)))]...
-1
w
+
-1
b
+
✖
g1
s
✖
a
b
✖
✖
+
bilinear gate
Ⓧ(a,b)
w
b
Ⓧ(-1,-1)
g1
s
Ⓧ(1,1)
Rank-1 Constraint System
for board of certain size and
max T steps

Rank-1 Constraint System (R1CS)

We rename all the variables into a vector $Y=(y_1,y_2,.., y_{Nv})$, Nv - number of variables

Then, our system of constraints becomes

constraint 1: $(a_0^1+\sum_i a_i^1 y_i) * (b_0^1+\sum_i b_i^1 y_i) = c_0^1+\sum_i c_i^1 y_i$

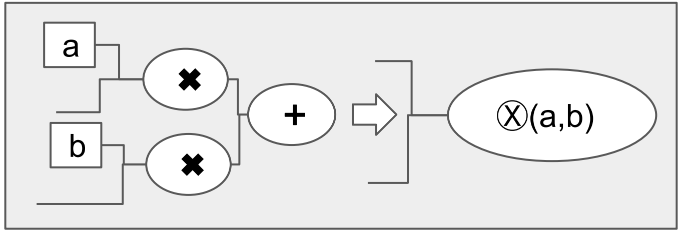
constraint 2: $(a_0^2+\sum_i a_i^2 y_i) * (b_0^2+\sum_i b_i^2 y_i) = c_0^2+\sum_i c_i^2 y_i$

...

constraint Nc: $(a_0^{Nc}+\sum_i a_i^{Nc} y_i) * (b_0^{Nc}+\sum_i b_i^{Nc} y_i) = c_0^{Nc}+\sum_i c_i^{Nc} y_i$

$$\begin{pmatrix} 1 & y_1 & y_2 & y_{..} & y_{Nv} \end{pmatrix} \boxed{\begin{pmatrix} a_0^1 \\ a_1^1 \\ a_2^1 \\ a_{..}^1 \\ a_{Nv}^1 \end{pmatrix} \begin{pmatrix} b_0^1 & b_1^1 & b_2^1 & b_{..}^1 & b_{Nv}^1 \end{pmatrix}} \begin{pmatrix} 1 \\ y_1 \\ y_2 \\ y_{..} \\ y_{Nv} \end{pmatrix}$$

This kind of matrix is of rank 1



Polynomial encoding (for intuition only)

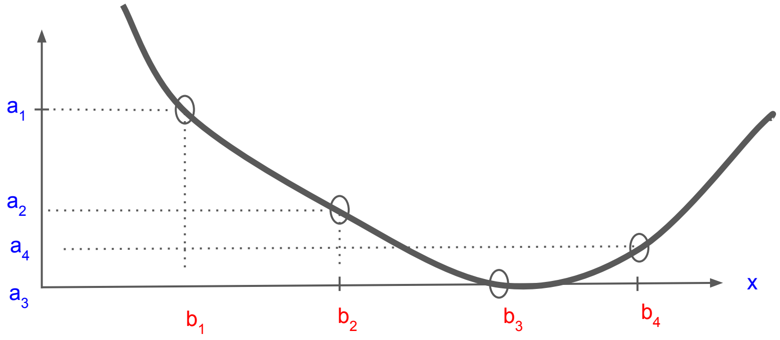
message: $a_1, a_2, \ldots, a_k$

pick arbitrary numbers: $b_1, b_2, \ldots, b_k$

Find polynomial p(x) such that:
$p(b_1) = a_1; p(b_2) = a_2; \ldots; p(b_k) = a_k$

add “robust” redundancy
$c(x) = p(x)\,(x-b_1)\,(x-b_2) \ldots (x-b_k)$

codeword = coefficients of c(x)



c(x) + errors?

c(x) divisible by $(x-b_1)(x-b_2) \ldots (x-b_k)$?

yes

Reconstruct p(x) by polynomial division, compute $a_1, a_2, \ldots, a_k$ by applying $b_1, b_2, \ldots, b_k$ to p(x)

NO

Checking divisibility for the codeword

$c(x)$ divisible by $(x-b_1)(x-b_2) \ldots (x-b_k)$?

Euclid algorithm for polynomials

Sample random $\tau \notin \{b_1, b_2, \ldots, b_k\}$
check that $c(\tau)$ divisible by $(\tau-b_1)(\tau-b_2) \ldots (\tau-b_k)$
It is meaningful if we are in a finite field

Repeat until confident

Interpolation of a, b, c-s: 4 variables, 3 constraints

Pick arbitrary $\alpha_1,\alpha_2,\alpha_3$

constraint 1: $(a_0^1+a_1^1y_1+a_2^2y_2+a_3^1y_3+a_4^2y_4) * (b_0^1+b_1^1y_1+b_2^1y_2+b_3^1y_3+b_4^1y_4) = c_0^1+c_1^1y_1+c_2^1y_2+c_3^1y_3+c_4^1y_4$

constraint 2: $(a_0^2+a_1^2y_1+a_2^2y_2+a_3^2y_3+a_4^2y_4) * (b_0^2+b_1^2y_1+b_2^2y_2+b_3^2y_3+b_4^2y_4) = c_0^2+c_1^2y_1+c_2^2y_2+c_3^2y_3+c_4^2y_4$

constraint 3: $(a_0^3+a_1^3y_1+a_2^3y_2+a_3^3y_3+a_4^3y_4) * (b_0^3+b_1^3y_1+b_2^3y_2+b_3^3y_3+b_4^3y_4) = c_0^3+c_1^3y_1+c_2^3y_2+c_3^3y_3+c_4^3y_4$

---

constraint 1: $[A_0(\alpha_1)+A_1(\alpha_1)y_1+A_2(\alpha_1)y_2+A_3(\alpha_1)y_3+A_4(\alpha_1)y_4] * [B_0(\alpha_1)+B_1(\alpha_1)y_1+B_2(\alpha_1)y_2+B_3(\alpha_1)y_3+B_4(\alpha_1)y_4] =$
$C_0(\alpha_1)+C_1(\alpha_1)y_1+C_2(\alpha_1)y_2+C_3(\alpha_1)y_3+C_4(\alpha_1)y_4$

constraint 2: $[A_0(\alpha_2)+A_1(\alpha_2)y_1+A_2(\alpha_2)y_2+A_3(\alpha_2)y_3+A_4(\alpha_2)y_4] * [B_0(\alpha_2)+B_1(\alpha_2)y_1+B_2(\alpha_2)y_2+B_3(\alpha_2)y_3+B_4(\alpha_2)y_4] =$
$C_0(\alpha_2)+C_1(\alpha_2)y_1+C_2(\alpha_2)y_2+C_3(\alpha_2)y_3+C_4(\alpha_2)y_4$

constraint 3: $[A_0(\alpha_3)+A_1(\alpha_3)y_1+A_2(\alpha_3)y_2+A_3(\alpha_3)y_3+A_4(\alpha_3)y_4] * [B_0(\alpha_3)+B_1(\alpha_3)y_1+B_2(\alpha_3)y_2+B_3(\alpha_3)y_3+B_4(\alpha_3)y_4] =$
$C_0(\alpha_3)+C_1(\alpha_3)y_1+C_2(\alpha_3)y_2+C_3(\alpha_3)y_3+C_4(\alpha_3)y_4$

Encoding of all constraints

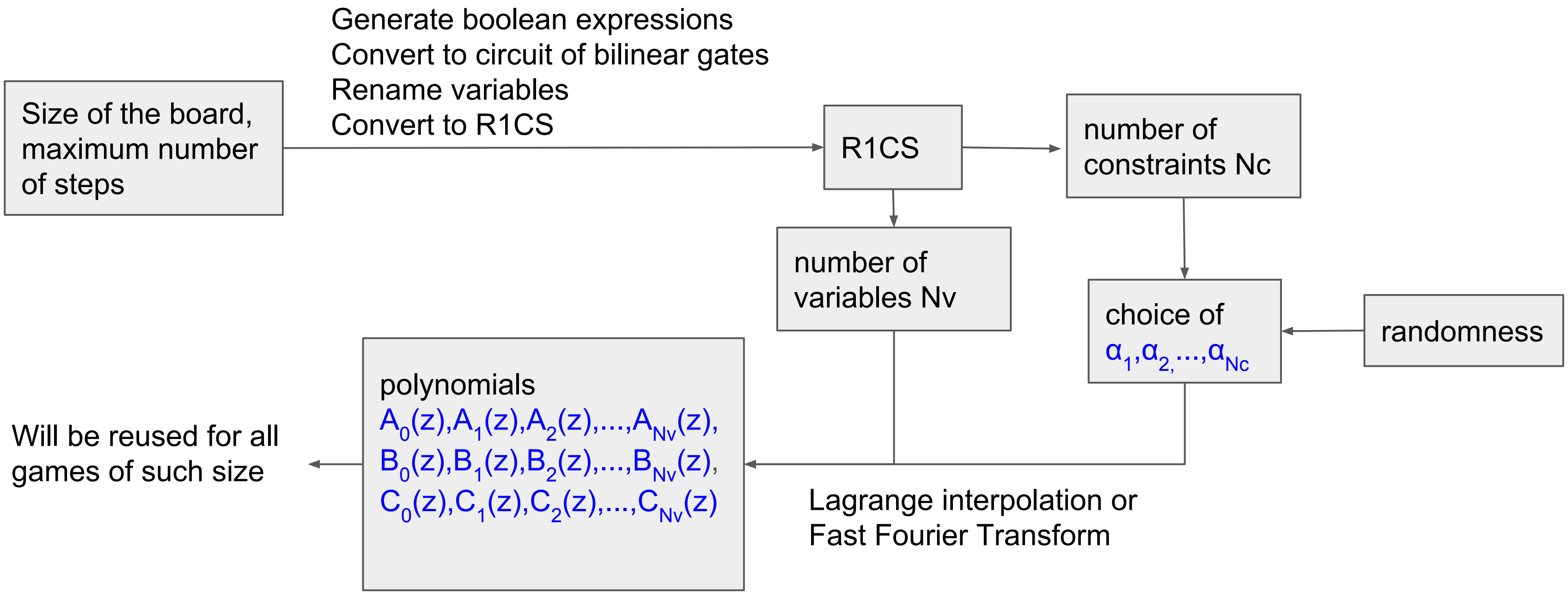
constraint j: $(A_0(\alpha_j)+\sum_i A_i(\alpha_j)y_i) * (B_0(\alpha_j)+\sum_i B_i(\alpha_j)y_i) = C_0(\alpha_j)+\sum_i C_i(\alpha_j)y_i$

Codeword for the whole solution: $(A_0(z)+\sum_i A_i(z)y_i) * (B_0(z)+\sum_i B_i(z)y_i) - C_0(z)+\sum_i C_i(z)y_i$

Codeword polynomial turns 0 at points $\{\alpha_1,\alpha_2,..., \alpha_{Nc}\}$, if all y_i are assigned to correct values.

Therefore, $\alpha_1,\alpha_2,..., \alpha_{Nc}$ are its roots, and it is divisible by $(z-\alpha_1)(z-\alpha_2)...(z-\alpha_{Nc})$!

Setup (before initial game position is known)



Prover’s preparation (initial position known)

initial position of the board x

sequence of moves leading to win: a

Execute the game logic

Assignment of all variables $y_1, y_2, \ldots, y_{Nv}$

Polynomials

$A(z) = A_0(z) + \sum_i A_i(z) y_i$

$B(z) = B_0(z) + \sum_i B_i(z) y_i$

$C(z) = C_0(z) + \sum_i C_i(z) y_i$

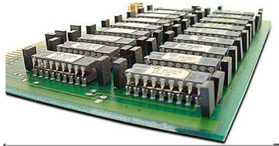
polynomials from setup

$A_0(z), A_1(z), A_2(z), \ldots, A_{Nv}(z),$

$B_0(z), B_1(z), B_2(z), \ldots, B_{Nv}(z),$

$C_0(z), C_1(z), C_2(z), \ldots, C_{Nv}(z)$

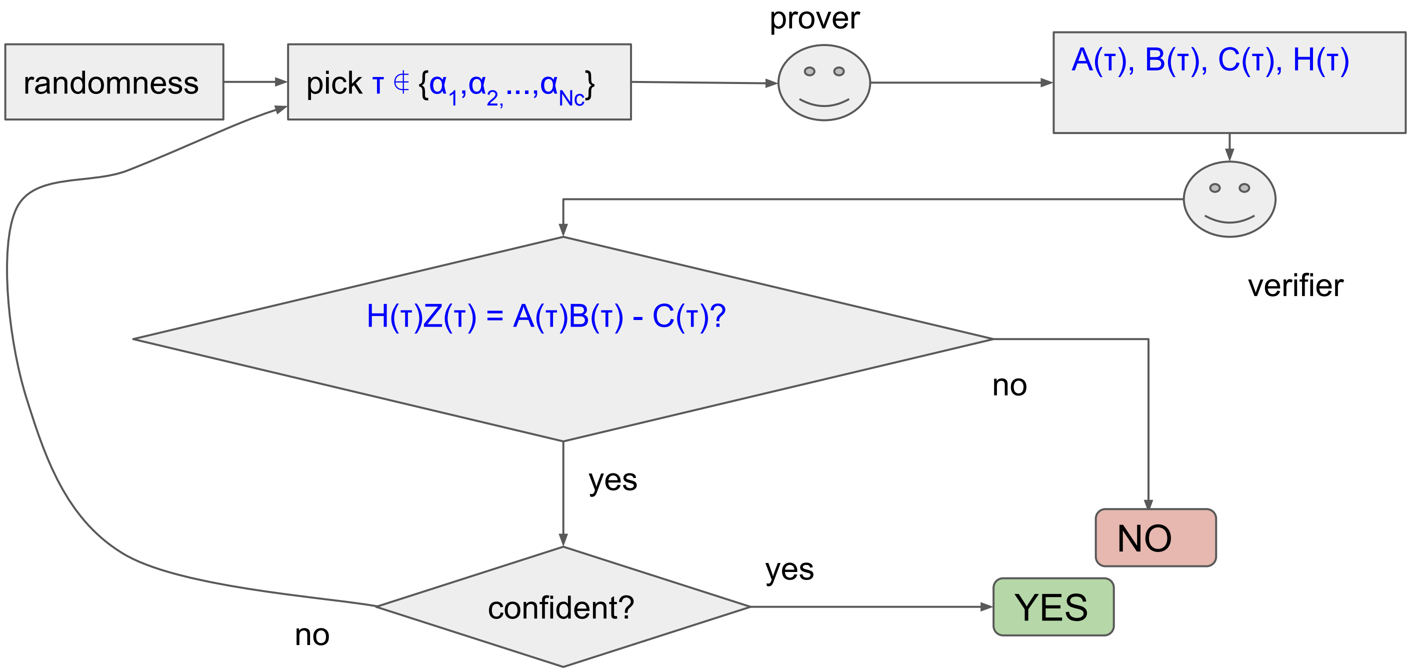
memory-intensive



choice of $\alpha_1, \alpha_2, \ldots, \alpha_{Nc}$

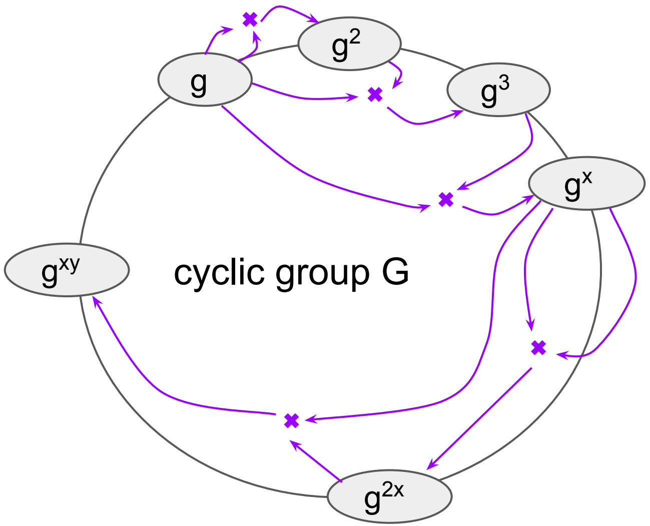
$$H(z) = \frac{A(z)B(z) - C(z)}{Z(z) = (z-\alpha_1)(z-\alpha_1)\ldots(z-\alpha_{Nc})}$$

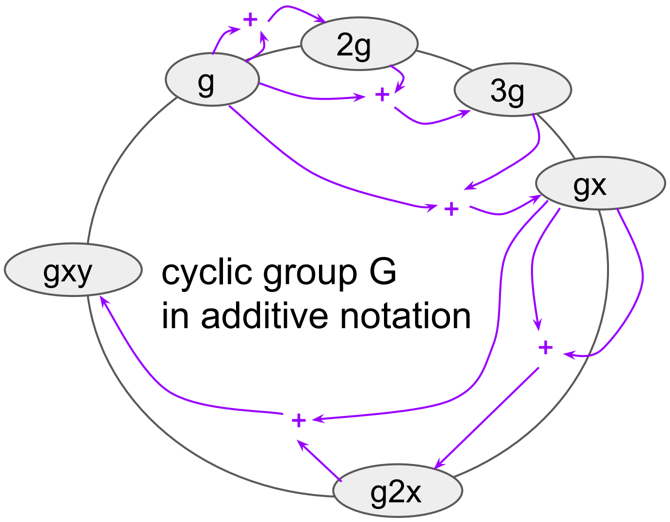
Prover cheats: $H(\tau) = \dfrac{A(\tau)B(\tau)-C(\tau)}{Z(\tau)}$



Linear PCP based on CDH

"Computational Diffie-Hellman" is the assumption that given element of the group: g, g^x, and g^y, it is computationally hard to find g^{xy}





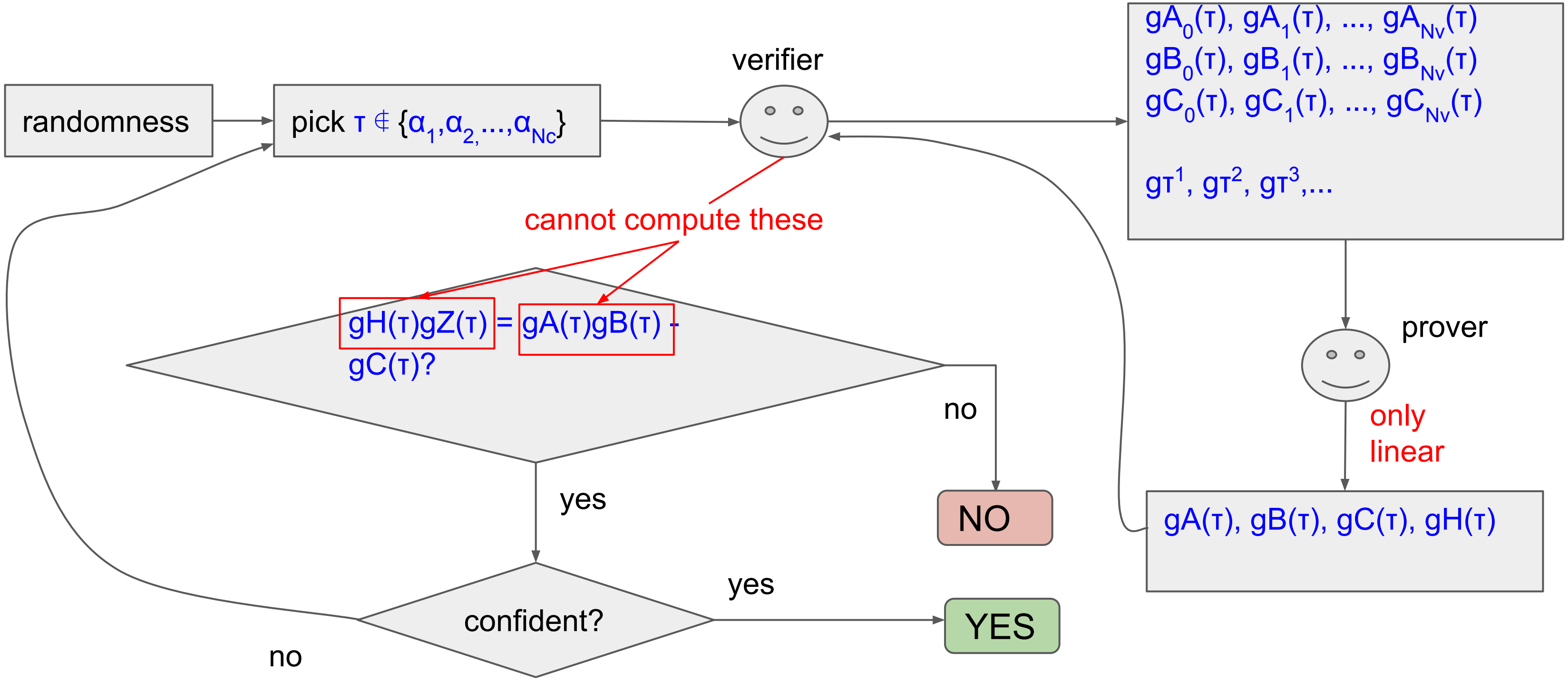
Linear PCP based on CDH

If we accept CDH assumption, we can "hide" numbers by representing them as group elements (kind of "wrapping" them). To "hide" numbers $\tau^1, \tau^2, \ldots$, verifier computes $g\tau^1, g\tau^2, \ldots$ The same applies to "hiding" $A_0(\tau), A_1(\tau), \ldots, A_{Nv}(\tau), B_0(\tau), B_1(\tau), \ldots, B_{Nv}(\tau), C_0(\tau), C_1(\tau), \ldots, C_{Nv}(\tau)$.

Because the prover does not know the multipliers of $g\tau^1, g\tau^2, \ldots, gA_0(\tau), gA_1(\tau), \ldots, gA_{Nv}(\tau), gB_0(\tau), gB_1(\tau), \ldots, gB_{Nv}(\tau), gC_0(\tau), gC_1(\tau), \ldots, gC_{Nv}(\tau)$, computationally bounded prover cannot multiply them with each other, but it can multiply them by number and add them together. It cannot solve this equation:

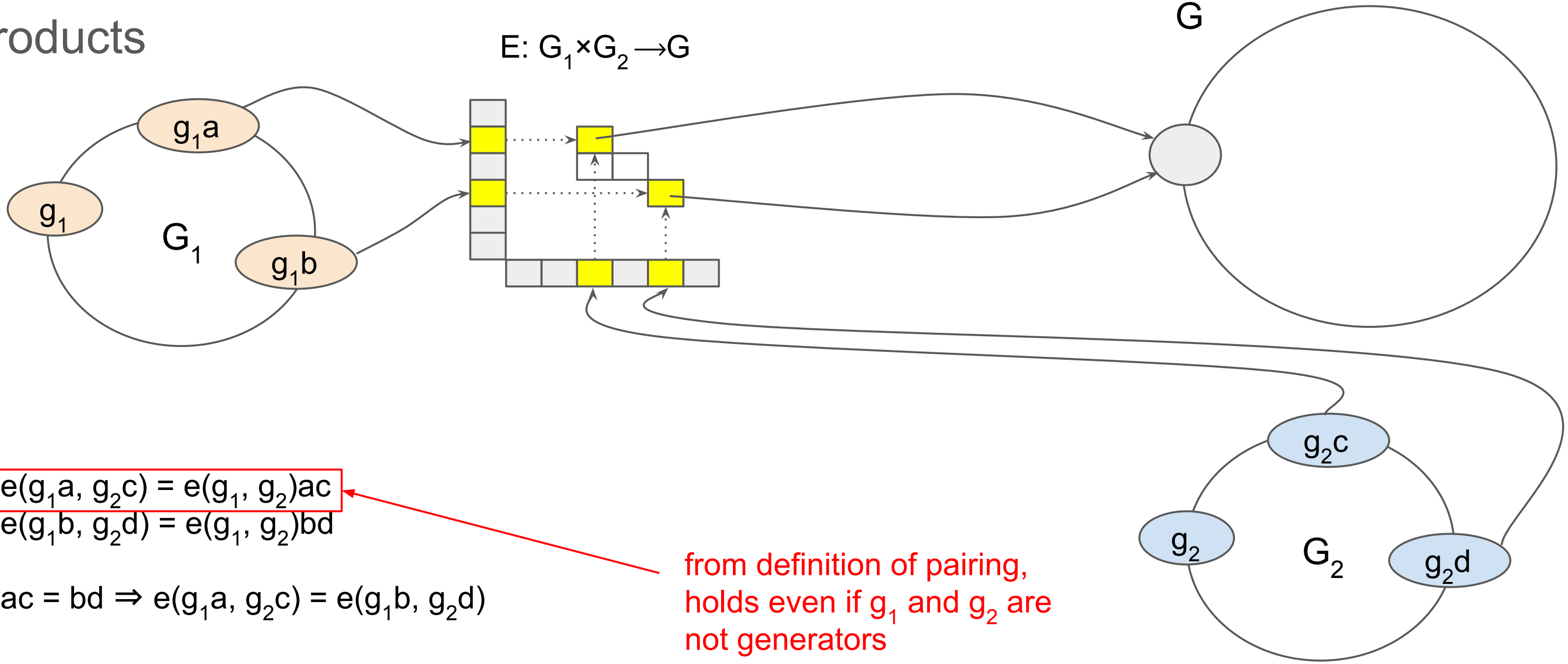
$$H(\tau) = \frac{A(\tau)B(\tau)-C(\tau)}{Z(\tau)}$$

Verifier is restricted too!



Solution - pairing

It does not allow multiplying group elements, but allows checking equality of two products



$\boxed{e(g_1a, g_2c) = e(g_1, g_2)ac}$

$e(g_1b, g_2d) = e(g_1, g_2)bd$

$ac = bd \Rightarrow e(g_1a, g_2c) = e(g_1b, g_2d)$

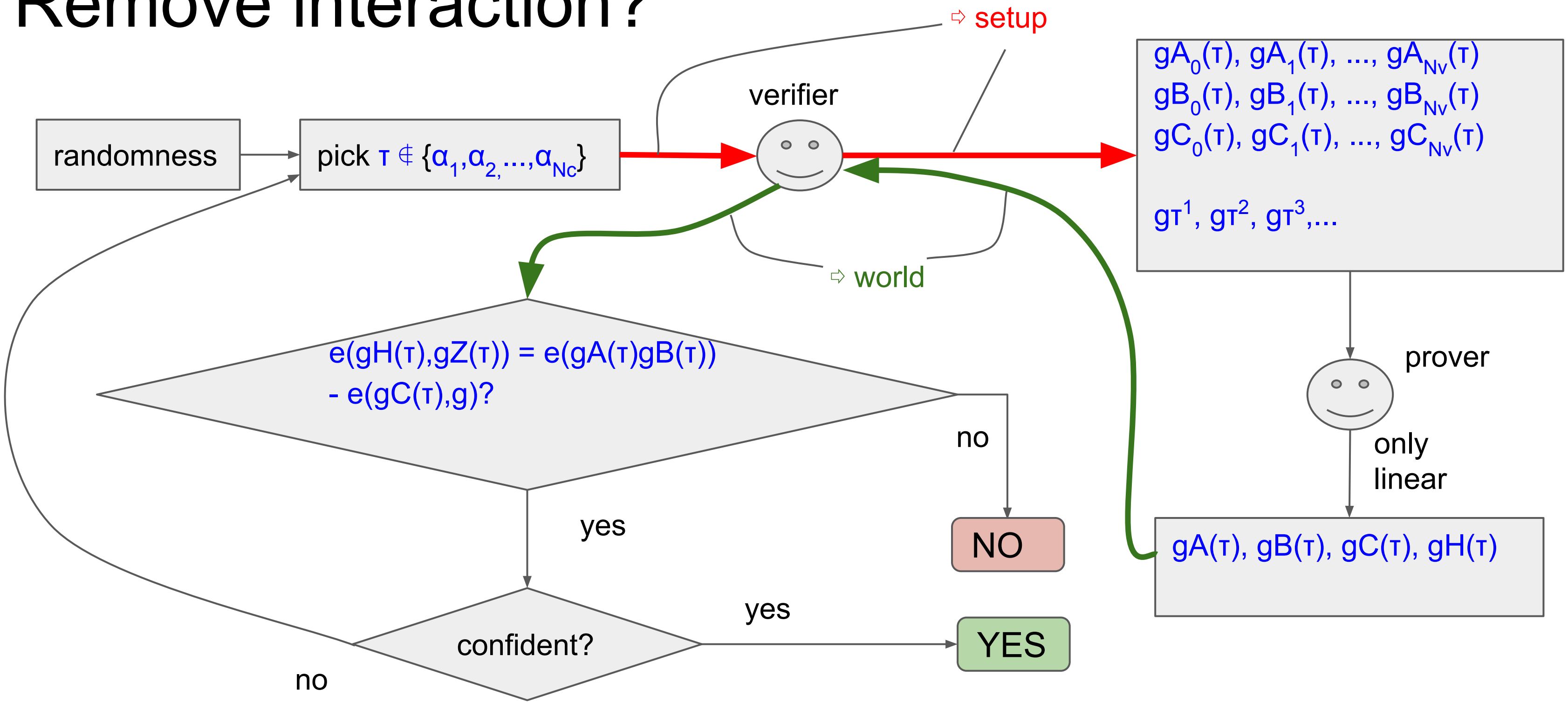
from definition of pairing, holds even if g_1 and g_2 are not generators

Pairings and elliptic curves

Pairings exist for some classes of groups induced by point arithmetics on elliptic curves. That is one of the reason elliptic curves are used in these constructions

Another reason - discrete logarithm is believed to be solved more easily groups of numbers (versus groups induced by elliptic curves), and solving discrete logarithm is enough to break CDH assumption

Remove interaction?



Trusted setup

must be destroyed

setup continued...

randomness

pick $\tau \notin \{\alpha_1, \alpha_2, \ldots, \alpha_{Nc}\}$

polynomials

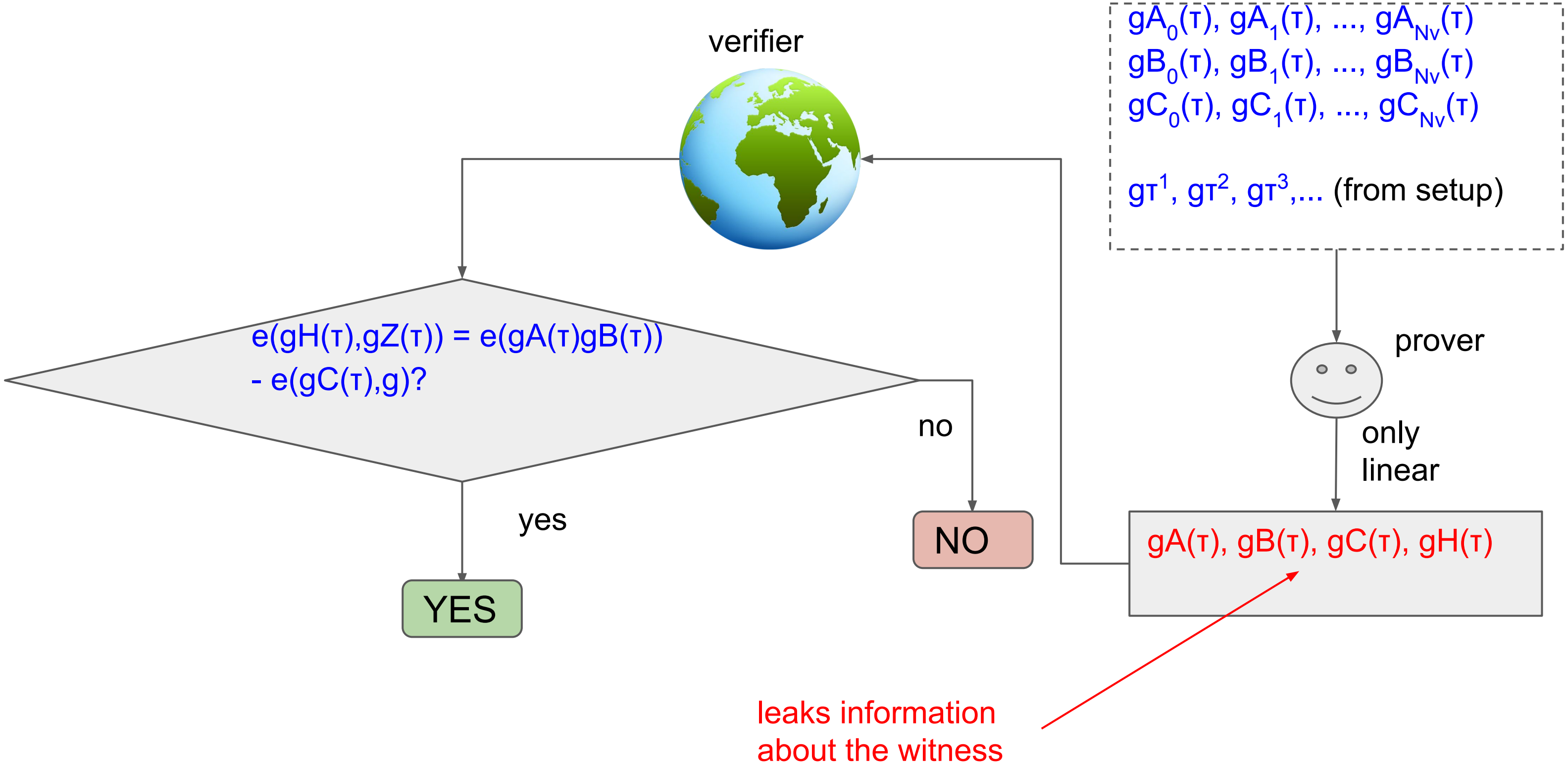
$A_0(z), A_1(z), A_2(z), \ldots, A_{Nv}(z),$
$B_0(z), B_1(z), B_2(z), \ldots, B_{Nv}(z),$
$C_0(z), C_1(z), C_2(z), \ldots, C_{Nv}(z)$

evaluate

$A_0(\tau), A_1(\tau), A_2(\tau), \ldots, A_{Nv}(\tau),$
$B_0(\tau), B_1(\tau), B_2(\tau), \ldots, B_{Nv}(\tau),$
$C_0(\tau), C_1(\tau), C_2(\tau), \ldots, C_{Nv}(\tau)$

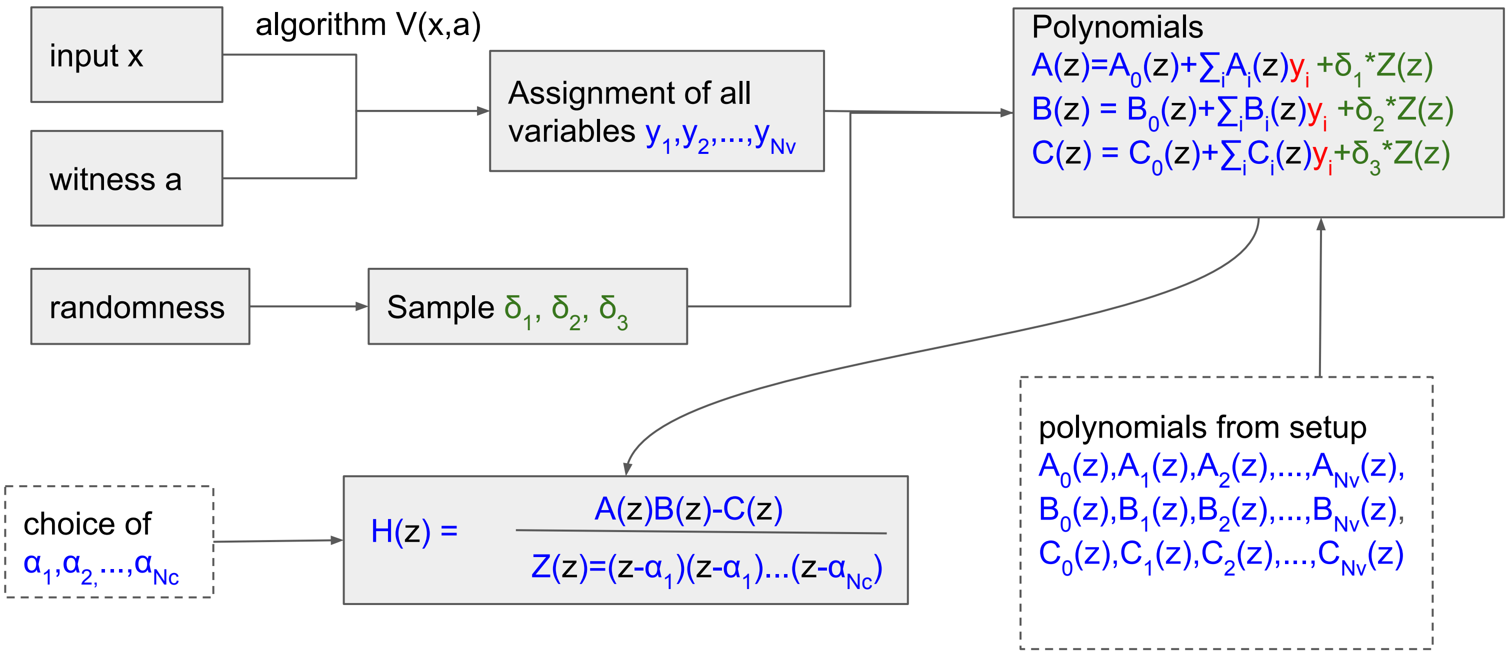
$gA_0(\tau), gA_1(\tau), \ldots, gA_{Nv}(\tau)$
$gB_0(\tau), gB_1(\tau), \ldots, gB_{Nv}(\tau)$
$gC_0(\tau), gC_1(\tau), \ldots, gC_{Nv}(\tau)$

$g\tau^1, g\tau^2, g\tau^3, \ldots$


verifier
gA0(τ), gA1(τ), ..., gANv(τ)
gB0(τ), gB1(τ), ..., gBNv(τ)
gC0(τ), gC1(τ), ..., gCNv(τ)
gτ1, gτ2, gτ3,... (from setup)
e(gH(τ),gZ(τ)) = e(gA(τ)gB(τ))
- e(gC(τ),g)?
no
yes
YES
NO
prover
only
linear
gA(τ), gB(τ), gC(τ), gH(τ)
leaks information
about the witness

Prover's preparation (zero knowledge)

Preserve divisibility, but make results appear random



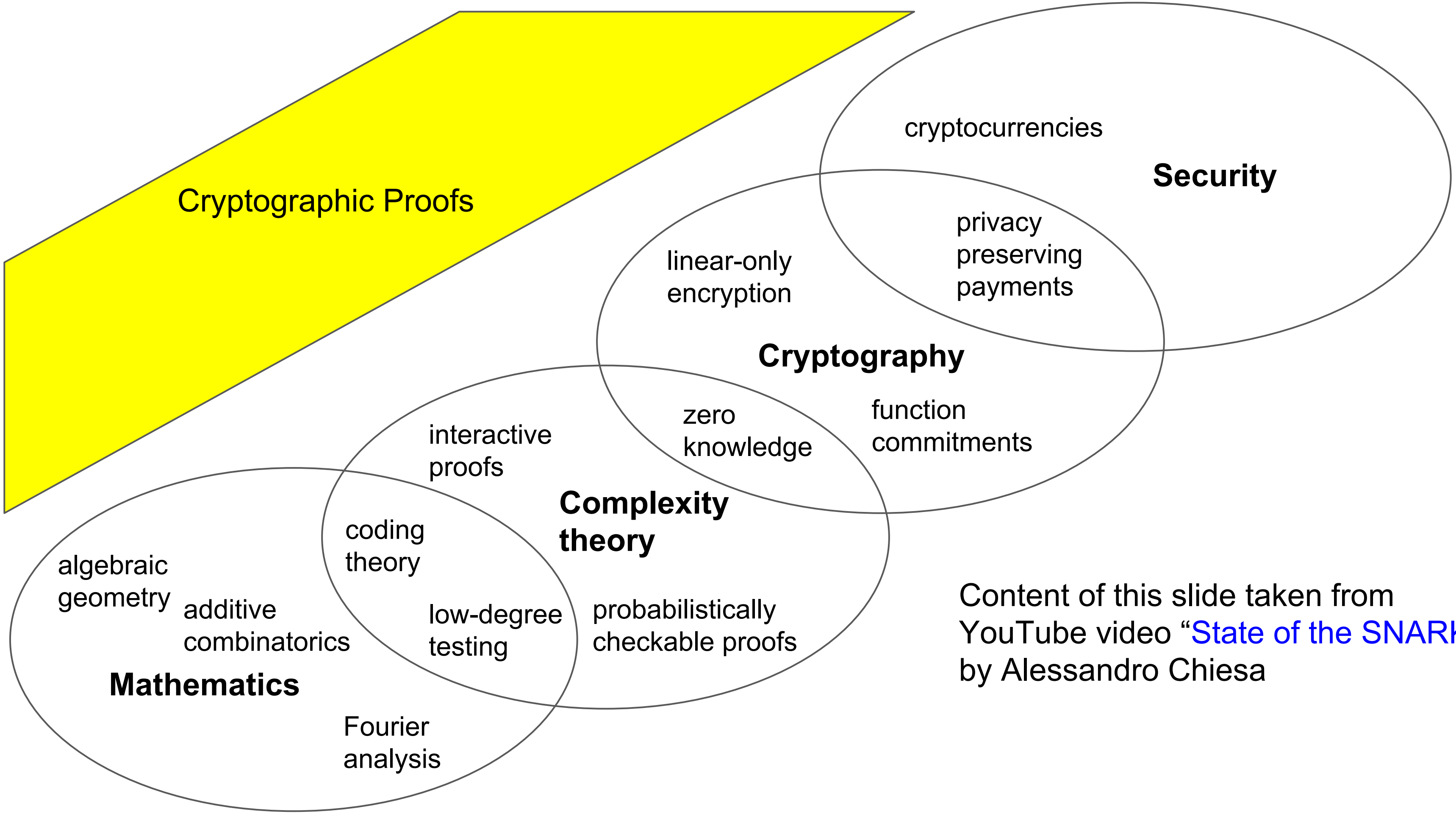
Succinctness

Proof is very short - few points of some elliptic curve: $g^{A(\tau)}, g^{B(\tau)}, g^{C(\tau)}, g^{H(\tau)}$

Verification is very efficient - couple of pairings - $e(g^{H(\tau)}, g^{Z(\tau)}) = e(g^{A(\tau)} g^{B(\tau)}) - e(g^{C(\tau)}, g)$?

Prover's work is proportional to the runtime of verification algorithm $V(x,a)$

Setup is expensive and has to be trusted - major drawback



Content of this slide taken from YouTube video “State of the SNARK” by Alessandro Chiesa

THANK YOU