

From First-Order Logic to SMT Solvers

David Walker
Princeton University

Last Time: First-Order Logic

A **signature** Σ specifies the function symbols and relation symbols, each with an arity.

Syntax: $t ::= x \mid f(t_1, \dots, t_n)$

$A ::= r(t_1, \dots, t_n) \mid t_1 = t_2 \mid \neg A \mid A \wedge B \mid A \vee B \mid A \rightarrow B \mid \forall x. A \mid \exists x. A$

Semantics:

- A **structure** M picks a *domain*, and a *meaning* for every symbol in Σ .
- An **assignment** ρ gives a value in the domain to every variable.
- $M, \rho \models F$ means that M *satisfies* F when the free variables are assigned values by ρ .
- F is **satisfiable** if there are *some* M and ρ with $M, \rho \models F$.

Last Time: Satisfaction: $M, \rho \models F$

$M, \rho \models F$: “M satisfies F when the free variables of F are interpreted by ρ .”

evaluate the subterms and check to see whether the tuple is in the relation r from the model

$M, \rho \models r(t_1, \dots, t_n)$ iff $(\llbracket t_1 \rrbracket(M, \rho), \dots, \llbracket t_n \rrbracket(M, \rho)) \in r^M$

$M, \rho \models s = t$ iff $\llbracket s \rrbracket(M, \rho) = \llbracket t \rrbracket(M, \rho)$

$M, \rho \models \text{true}$

always

$M, \rho \models \text{false}$

never

evaluate the terms s and t and check if they are the same value u from U

$M, \rho \models \neg F$

iff $M, \rho \not\models F$

$M, \rho \models F \wedge G$

iff $M, \rho \models F$ and $M, \rho \models G$

$M, \rho \models F \vee G$

iff $M, \rho \models F$ or $M, \rho \models G$

$M, \rho \models F \rightarrow G$

iff $M, \rho \not\models F$ or $M, \rho \models G$

is there some way to satisfy F by extending the valuation at v ?

$M, \rho \models \exists v. F$ iff for **some** $m \in U^M$, $M, \rho\{v \mapsto m\} \models F$

$M, \rho \models \forall v. F$ iff for **all** $m \in U^M$, $M, \rho\{v \mapsto m\} \models F$

is F satisfied for all extensions (at v) of the valuation?

Using the Semantics

Are the integers a model of this formula?

$$\mathbb{Z}, \{ \} \models \forall x. (\exists y. x < y)$$

step 1 (use definition of $\forall$)

must show **for all** $m \in \mathbb{Z}$, ($m = 0, 1, 2, -1, \dots$)
 $M, \{x \mapsto m\} \models (\exists y. x < y)$

step 2 (use definition of $\exists$)

Given an arbitrary $m \in \mathbb{Z}$,
must choose **some** $m' \in \mathbb{Z}$ compatible with m
 $M, \{x \mapsto m, y \mapsto m'\} \models (x < y)$

Is there a choice for m' that works?

Let m' be $m+1$:

$$M, \{x \mapsto m, y \mapsto m+1\} \models (x < y)$$

Now we can see the formula is satisfied:

$$M, \{x \mapsto m, y \mapsto m+1\} \models (x < y) \text{ iff } m < m+1$$

use definition of **variables** x, y
and **facts** about the model of $<$

Using the Semantics

Are the integers a model of this formula?

$$\mathbb{Z}, \{ \} \models \exists y. \forall x. (x < y)$$

must choose a single $m \in \mathbb{Z}$. Which one? 0? 17?
 $M, \{x \mapsto m\} \models (\forall x. x < y)$

step 1 (use definition of $\exists$)

which integer is bigger than all of the others?

There is none!

If our model was $\mathbb{Z} \cup \{ \infty \}$ we would be able to satisfy the formula.

Moral: Use the semantics carefully to determine if your model satisfies the formula

Moral: Quantifier order matters. $\exists y. \forall x. F$ and $\forall x. \exists y. F$ are not equivalent!

Terminology: Entailment

Let Γ be a set of formula.

“ Γ entails G ” $\Gamma \models G$

Intuition: “for any model, if all the formula in Γ are true then G is true”

$\Gamma \models G$ whenever:

for all models M , and valuations p ,

if $(M, p \models F \text{ for all } F \text{ in } \Gamma)$ then $M, p \models G$

Simple Examples:

$p, q, r \models p$

$p_1, \dots, p_n \models \mathbf{true}$

From FOL to Satisfiability Modulo Theories

First-order logic is a tempting logic to work in, but it has some practical problems:

- we can't automate it reliably (satisfiability and validity are undecidable in general!)
- certain concepts come up over and over again:
 - integers, reals, strings ...
 - First-order logic has nothing "built in" -- we don't want to have to explain what properties $<$ or $+$ have every time we want to use them.

Satisfiability Modulo Theories is, approximately, the "practical subset of FOL"

- structured terms and relations like FOL
- *but has built-in operations with built-in models* ($+$, $<$, 0, 1, 2, 3, etc)
 - these are the *theories*
- usage: we construct formula with free variables:
 - $\neg(\text{start_logic} = \text{start_algebra}) \wedge (9 \leq \text{start_logic} \wedge \text{start_logic} < 17)$
 - we ask, *does there exist* values from the *built-in models* that we can substitute for variables that satisfy the formula?

Example Theories Supported by SMT Solvers

A **theory** fixes the meaning of some symbols. Example: Linear Arithmetic (LA)

$\Sigma = \{ 0, 1, 2, \dots, +, -, =, <, \leq \}$

M = domain is the integers $\mathbb{Z}$, with functions interpreted as usual

SMT Question: What substitution ρ allows the LA model to satisfy a formula F ?

$LA, \rho \models x < y \wedge y < 3$

Answer: $\{x \mapsto 0, y \mapsto 1\}$

In plain SAT, we look for Boolean assignments to propositional variables p, q

In SMT, we look for (eg: Integer) assignments to variables using specific models like LA.

Other theories an SMT solver knows:

bitvectors

arrays

reals

strings

functions

The Exam Scheduling Problem

Example Problem: Exam Scheduling

At the end of every semester, the registrar must **give every final exam a time**.

- Exams have different lengths.
- Every exam must fit inside the exam day (or exam period).
- No student may be asked to sit two exams at once.



Ask: Is it possible to find a start time for each class?

The same kind of problem appears wherever jobs compete for time:

timetables

meetings

debriefs

shifts

cloud jobs

Formalizing: Courses and Exams

We are given a finite set of **courses** $C = \{ c_1, \dots, c_n \}$, each with one exam.

```
C = { logic, algebra, poetry, chemistry }
```

Each exam has a **duration**, in hours: $d : C \rightarrow \{1, 2, 3\}$

```
d(logic)   = 3      d(poetry)   = 1
d(algebra) = 2      d(chemistry) = 2
```

Exams happen on one **day**, from T_{start} to T_{end} :

```
T_start = 9      T_end = 17
```

Formalizing: Students and Enrollment

We are given a finite set of **students** S , and an **enrollment** E :

for each student t , the set $E(t) \subseteq C$ of exams that t must sit.

```
E(alice) = { logic, algebra }  
E(bob)   = { algebra, poetry }  
E(carol) = { logic, chemistry }  
E(dave)  = { poetry, chemistry }
```

Two courses **conflict** when some student takes both:

```
 $c_1 \sim c_2 \quad \text{iff} \quad c_1 \neq c_2 \quad \text{and} \quad \{c_1, c_2\} \subseteq E(t) \quad \text{for some } t \in S$ 
```

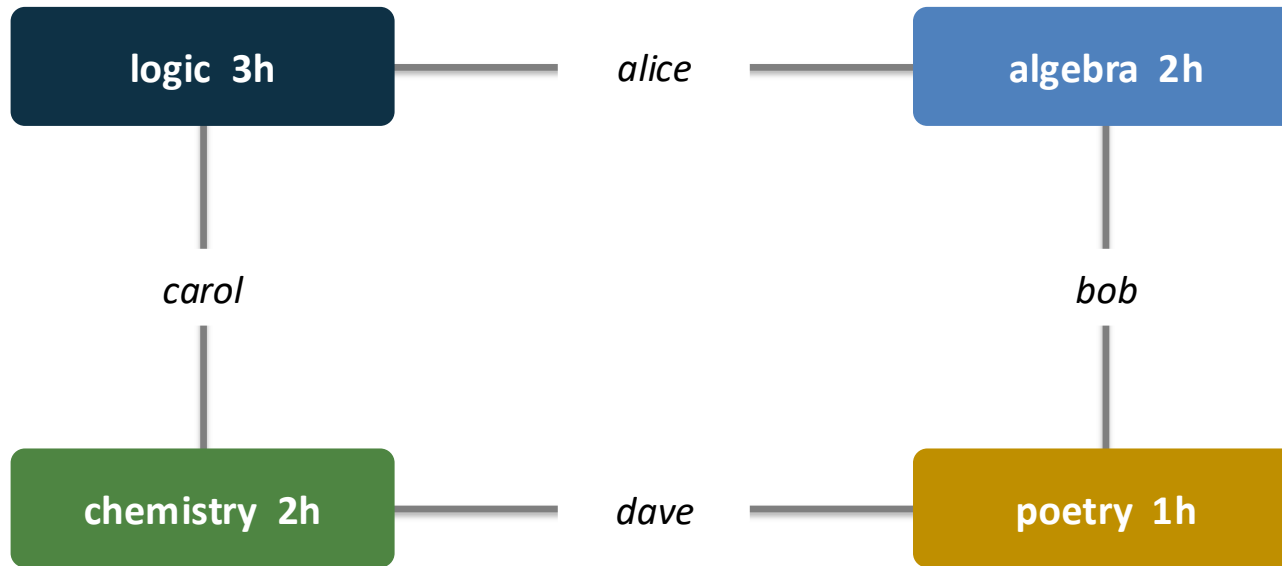
Conflicting exams must not be held at the same time.

Courses that do *not* conflict may share a time slot.

The Conflict Graph

One node per course.

One edge for each conflicting pair, labelled by a student who creates it.



logic and poetry are not directly connected: they may run at the same time.

algebra and chemistry are not directly connected

Schedules, and Validity

A **schedule** σ maps each course to the hour its exam begins: $\sigma : C \rightarrow \mathbb{Z}$

```
 $\sigma = \{ \text{logic} \mapsto 9, \text{algebra} \mapsto 12, \text{poetry} \mapsto 14, \text{chemistry} \mapsto 15 \}$ 
```

Exam c occupies the half-open interval $[\sigma(c), \sigma(c) + d(c))$



σ is **valid** when:

- every exam fits in the day: $T_{\text{start}} \leq \sigma(c)$ and $\sigma(c) + d(c) \leq T_{\text{end}}$
- if $c_1 \sim c_2$, then the intervals of c_1 and c_2 are disjoint

The Exam Scheduling Problem

Given

courses C , durations d , students S , enrollment E , and a day $[T_{\text{start}}, T_{\text{end}}]$

Ask

is there a valid schedule $\sigma : C \rightarrow \mathbb{Z}$?

If yes, then What is that schedule?

Encoding Exam Scheduling as an SMT Problem

One Integer Variable per Course

For each course c , introduce one integer variable named `start_c`.

`start_c` is the hour at which the exam for c begins

For example, if courses are logic, algebra, poetry, chemistry:

`start_logic` `start_algebra` `start_poetry` `start_chemistry`

Constraint 1: Every Exam Fits in the Day

An exam may not begin before T_{start} , or end after T_{end} .

for each course $c \in C$:

$$T_{\text{start}} \leq \text{start}_c \quad \wedge \quad \text{start}_c + d(c) \leq T_{\text{end}}$$

For our example:

$$\begin{aligned} & (9 \leq \text{start_logic} \quad \wedge \quad \text{start_logic} + 3 \leq 17) \\ \wedge & (9 \leq \text{start_algebra} \quad \wedge \quad \text{start_algebra} + 2 \leq 17) \\ \wedge & (9 \leq \text{start_poetry} \quad \wedge \quad \text{start_poetry} + 1 \leq 17) \\ \wedge & (9 \leq \text{start_chemistry} \quad \wedge \quad \text{start_chemistry} + 2 \leq 17) \end{aligned}$$

Constraint 2: No Overlapping Exams

No student sits two exams at once. Intervals $[a_1, b_1)$ and $[a_2, b_2)$ are disjoint exactly when one ends before the other begins:



for each conflicting pair $c_1 \sim c_2$:

$$\text{start}_{c_1} + d(c_1) \leq \text{start}_{c_2} \quad \vee \quad \text{start}_{c_2} + d(c_2) \leq \text{start}_{c_1}$$

Constraint 2: For Our Example

One disjunction per edge of the conflict graph:

```
(start_logic + 3 ≤ start_algebra  
∨ start_algebra + 2 ≤ start_logic)
```

alice

```
∧ (start_algebra + 2 ≤ start_poetry  
    ∨ start_poetry + 1 ≤ start_algebra)
```

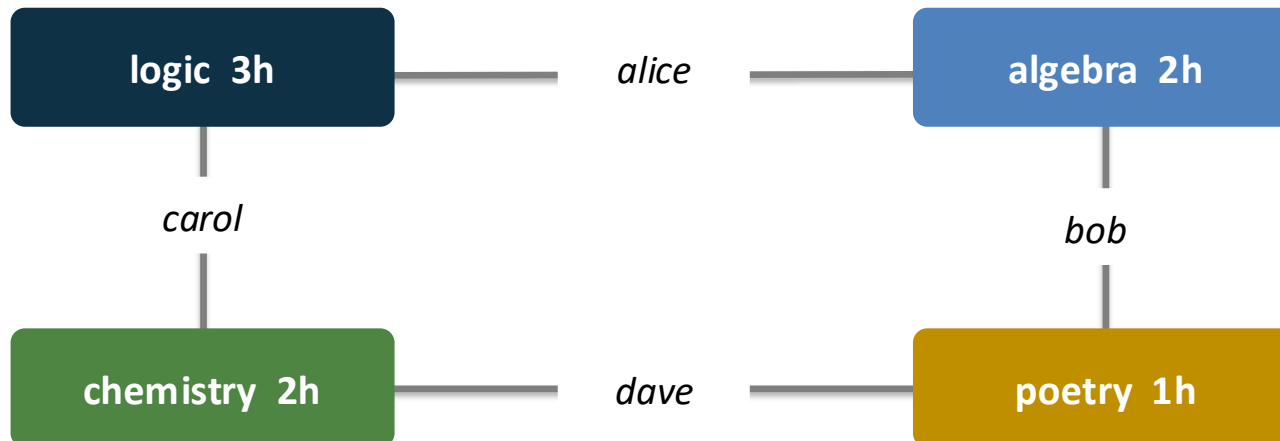
bob

```
∧ (start_logic + 3 ≤ start_chemistry  
    ∨ start_chemistry + 2 ≤ start_logic)
```

carol

```
∧ (start_poetry + 1 ≤ start_chemistry  
    ∨ start_chemistry + 2 ≤ start_poetry)
```

dave



Summary: The Encoding

Within the day

one per course

$\wedge$

No overlap

one per conflicting pair

SMT solver will return:

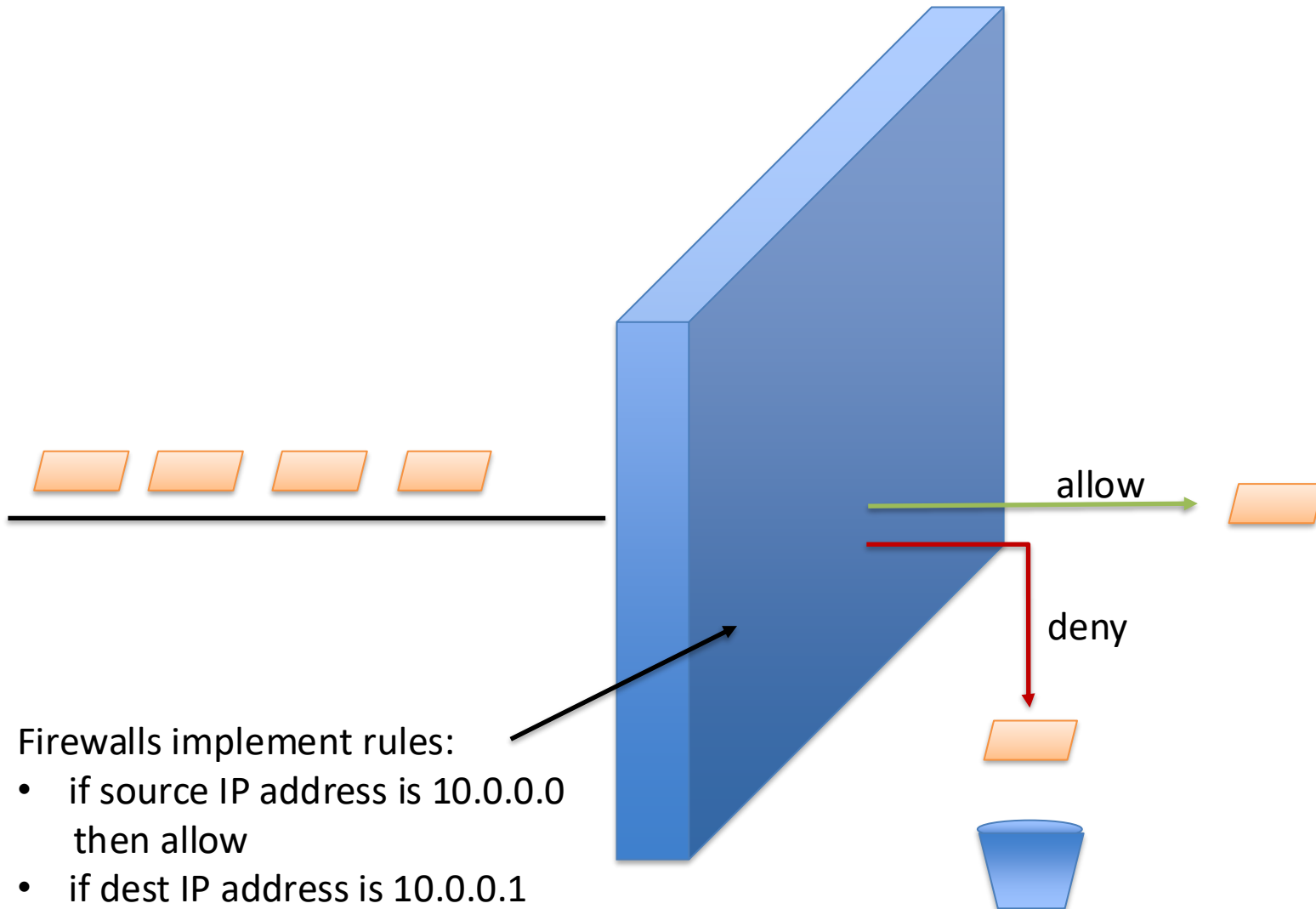
- SAT and supply a number for every variable – the start times of the exams
- UNSAT – there does not exist a substitution of values for variables that allows the built-in models to satisfy the constraints

Assignment 5:

Firewall Translator and Verifier

Using an SMT Solver

Network Firewall



Firewalls implement rules:

- if source IP address is 10.0.0.0
then allow
- if dest IP address is 10.0.0.1
then deny

Goal: Build a Firewall Translator

Many formats for firewall rules

- **Cisco ACLs** (used by routers)
- **AWS Security Groups** (used in Amazon's cloud).

Translate

Cisco ACL → AWS JSON
using an LLM (Ollama).

The LLM can be wrong!



Check

Task 1

Are two policies
equivalent?

*Encode policies as
formula and check for
equivalence.*



Repair

Tasks 2,3

Fix a broken AWS policy
so it matches a reference.

A series of SMT queries.

Background: Packets

Packets

Internet communication breaks data into small chunks called **packets**. Packets include:

- a **header**: where it is from, where it is going, and what kind of traffic it is
- a **payload**: the data transmitted

A firewall looks only at the header. Standard IP header fields are:

proto	src_ip	dst_ip	src_port	dst_port
kind of traffic	who sent it	who receives it	sender's port	receiver's port

For our purposes:



Field	Values	Meaning
<code>proto</code>	TCP or UDP	the kind of traffic
<code>src_ip</code>	32-bit unsigned integer	source IP address (who sent it)
<code>dst_port</code>	16-bit: 0 – 65535	which service should receive it. eg: • port 22: ssh (remote log in) • port 80: http (web request)

A firewall *policy* is therefore a function from these triples to a decision:

```
policy : proto × src_ip × dst_port → { permit, deny }
```

Two Policy Languages

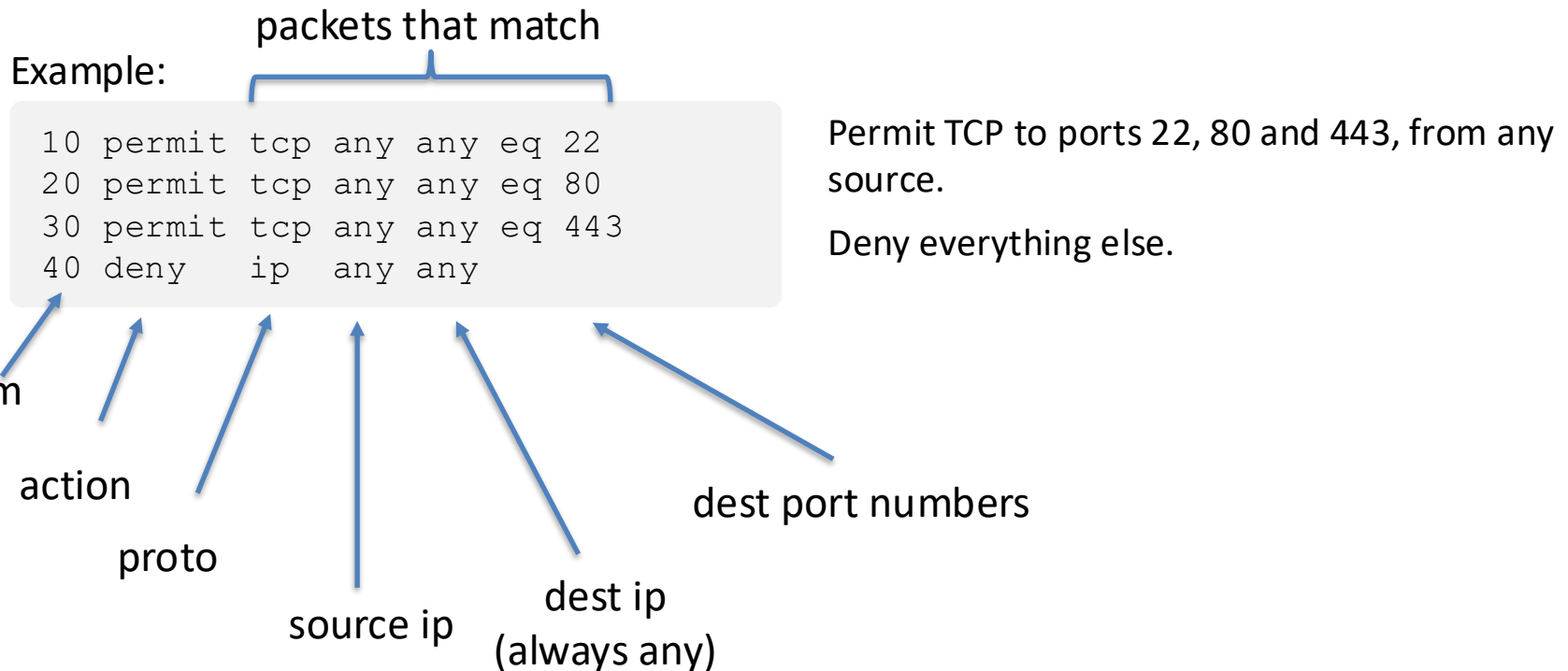
Cisco ACLs and AWS Security Groups

Format 1: Cisco Access Control Lists (ACL)

An ACL is a list of rules.

Each rule has

- a sequence number (tells us the order rules match)
- an action (permit/deny)
- information about the packets that match



ACL Semantics: First Match Wins

Try rules in sequence-number order. The **first rule that matches** decides on packet. If no rule matches, the packet is **denied** (the *implicit default deny*).

example 1:

```
10 deny    tcp any any eq 23
20 permit  tcp any any range 1 1024
```

all TCP packets permitted for ports 1–1024
except port 23

example 2:

```
10 permit  tcp any any range 1 1024
20 deny    tcp any any eq 23
```

all TCP packets permitted
for ports 1–1024 (*including port 23*)

In example 2, both rule 10 and rule 20 match tcp packets for port 23. Rule 10 comes first ($10 < 20$) so rule 10 takes precedence.

More details of the format (how sets of IP addresses and ports are specified in assignment)

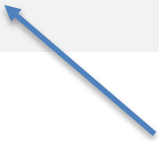
Format 2: AWS Security Groups

An AWS policy is a JSON *array* of permission objects.

Every object is an allow rule. (There are no deny rules.)

Example Permission Object:

```
{
  "IpProtocol": "tcp" ,
  "FromPort": 80,
  "ToPort": 100,
  "IpRanges": [ {"CidrIp": "0.0.0.0/0"} ]
}
```



this is a set of IP addresses,
described in more detail
in the assignment

Deny rules allow for some compact encodings (they can “punch holes” in a space of accept packets), but we can’t use them in this format.

Because there are no deny rules, translation from ACL to AWS formats can be tricky!

One Policy, Two Formats

ACL:

```
10 permit tcp any any eq 22
20 permit tcp any any eq 80
30 permit tcp any any eq 443
```

AWS:

```
[ {"IpProtocol": "tcp", "FromPort": 22, "ToPort": 22, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
  {"IpProtocol": "tcp", "FromPort": 80, "ToPort": 80, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
  {"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]} ]
```

	Cisco ACL	AWS Security Group
rule order	matters: first match wins	irrelevant: any match permits
actions	permit and deny	permit only
source	wildcard mask	list of CIDR blocks
dest. port	eq / range / lt / gt	FromPort ... ToPort
no match	deny	deny

Task 1: Checking Equivalence

lib/equiv.ml

When Are Two Policies Equivalent?

$P_1 \equiv P_2$ iff for every packet p : P_1 permits $p \Leftrightarrow P_2$ permits p

Total number of packets: 2^{49} Complete packet-by-packet testing is impossible!

When Are Two Policies Equivalent?

Solution:

(1) Generate variables that represent a packet:

- `is_tcp` - true if TCP; false if UDP
- `src_ip` - bitvector representing `src_ip`
- `port` - port of the packet

(2) Compute formula F1 and F2 involving those variables such that

- F1 is true whenever the packet is permitted by policy P1
- F2 is true whenever the packet is permitted by policy P2

(3) Ask: $\neg (F1 \iff F2)$ satisfiable?

UNSAT

No such packet exists: the policies are **equivalent**.

SAT

The model is a packet F1 and F2 disagree on: a **counterexample**.

Your Job: `encode_acl` and `encode_aws`

```
let encode_acl (pkt : packet) (acl : Acl.t) : formula = ...  
let encode_aws (pkt : packet) (sg : Aws.t) : formula = ...
```

Spec: the formula is **true if and only if the policy permits the packet** `pkt`.

Building Formula (formula.mli)

Formula/Term	Meaning	Formula/Term	Meaning
<code>tru, fls</code>	true, false	<code>int n</code>	$\text{int} \rightarrow \text{integer}$
<code>not f</code>	negation	<code>x =. y</code>	$x = y$
<code>ands fs</code>	conjunction of a list	<code>x <. y</code>	$x < y$
<code>ors fs</code>	disjunction of a list	<code>x <=. y</code>	$x \leq y$
<code>ite c t e</code>	if c then t else e	<code>x >. y</code>	$x > y$
<code>bits n</code>	$\text{int32} \rightarrow \text{bits}$	<code>x >=. y</code>	$x \geq y$
<code>bits_and x y</code>	bitwise AND, OR	<code>&&& </code>	and, or (binary)
<code>bits_eq x y</code>	equal 32-bit words	<code>==> <=></code>	implies, iff

Running the Checker

```
10 permit tcp any any eq 22
20 permit tcp any any eq 80
30 permit tcp any any eq 443
```

simple.acl

```
[ {"IpProtocol":"tcp","FromPort":80, ...},
  {"IpProtocol":"tcp","FromPort":443,...} ]
```

simple-no-ssh.json

```
$ dune exec fw -- check examples/simple.acl examples/simple-no-ssh.json
```

NOT EQUIVALENT

Found a packet on which the two policies disagree:

Protocol : TCP

Src IP : 0.0.0.0

Dst Port : 22

Policy 1 (ACL) : PERMIT

Policy 2 (AWS) : DENY

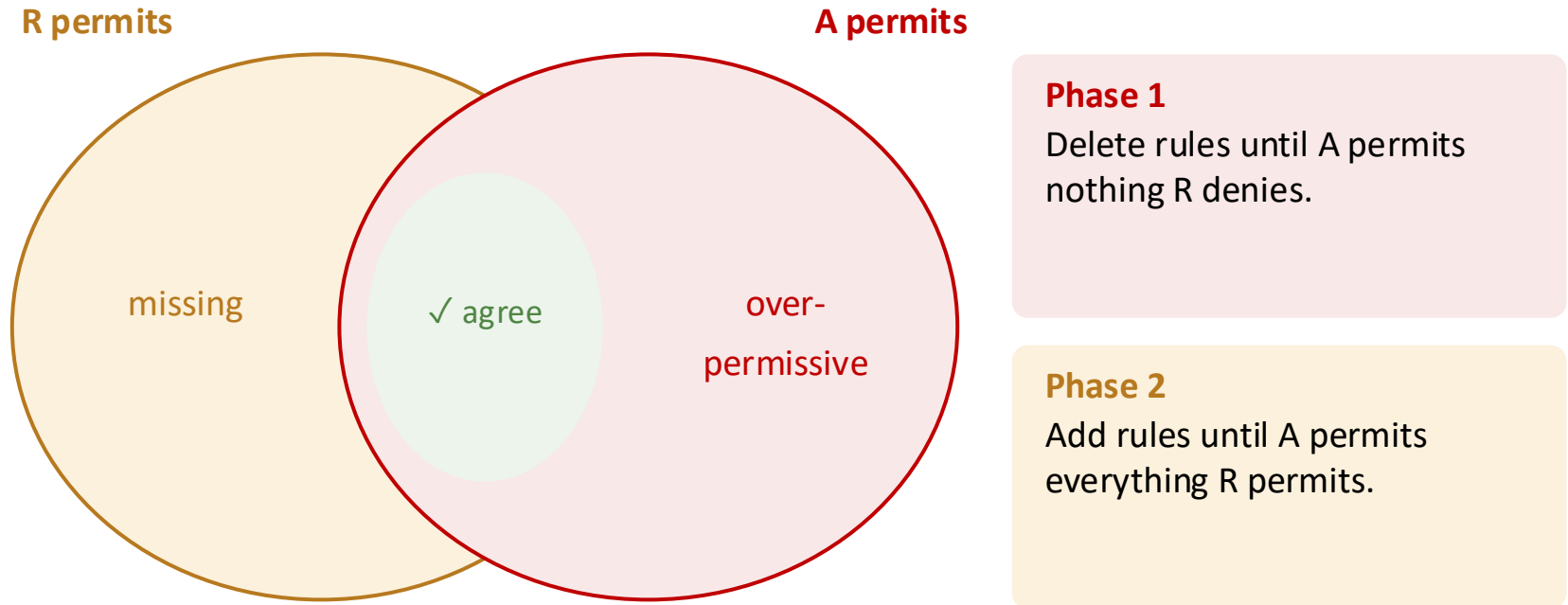
The JSON forgot SSH. Equivalent policies print EQUIVALENT.

Task 2: Repair

lib/repair.ml

The Repair Problem

Given a **reference ACL R** and a **broken AWS policy A**, change A until $A \equiv R$.



Both phases are driven by solver queries that return **witness packets**.

Output: the repaired policy and the list of patches (Delete r , Add r) applied.

Phase 1: Delete Over-permissive rules

Repeat the following until done:

Let $R(p)$ be the formula for the reference (ie: good) firewall on packet variables p

Let $A(p)$ be the possibly broken firewall on packet variables p

Let $\text{bounds}(p)$ mean that p is a valid packet (specifically, it's ports are in a 16 bit range)

Check: $\text{bounds}(p) \wedge A(p) \wedge \neg R(p)$

- A satisfying assignment is a specific (concrete) packet c that the reference blocks and A permits.
 - Delete the first rule in A that we find that accepts c .
- If UNSAT, then we are done this phase

Implementation Provided: `lib/repair.ml`: functions **phase1** and **find_over_permissive**

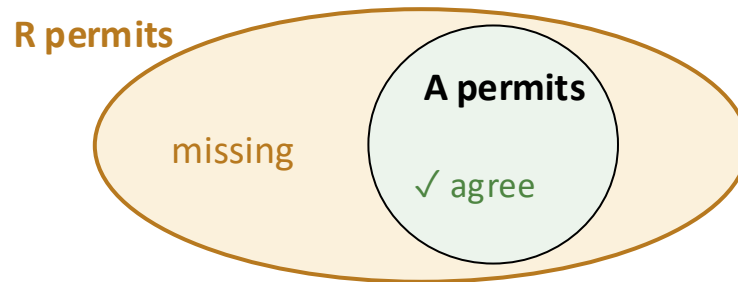
Example: `ssh-web-extra.json` has the right rules plus one extra:

```
$ dune exec fw -- repair examples/ssh-web.acl examples/ssh-web-extra.json
Repair found (1 operation):
  Step 1 - DELETE rule:  tcp 8080-8080 from 0.0.0.0/0
```

Phase 2: Add Rules to Increase Coverage

After Phase 1, the situation looks like the following

We need to expand the green so it matches the yellow area exactly.



Repeat the following until done:

Let $R(p)$ be the formula for the good firewall on packet p

Let $A(p)$ be the possibly broken firewall on packet p

Check: $\text{bounds}(p) \wedge \neg A(p) \wedge R(p)$

- A satisfying assignment is a packet c that A blocks and the reference permits.
 - **Add a new rule to cover that packet**
- If UNSAT, then we are done this phase

Simple example

Reference ACL

```
10 permit tcp host 10.0.0.1 any eq 1
20 deny   ip  any          any
```

one-host-one-port.acl

Attempted translation (broken)

```
[ tcp 2-2 from 10.0.0.1/32 ]
```

one-host-one-port-wrong.json

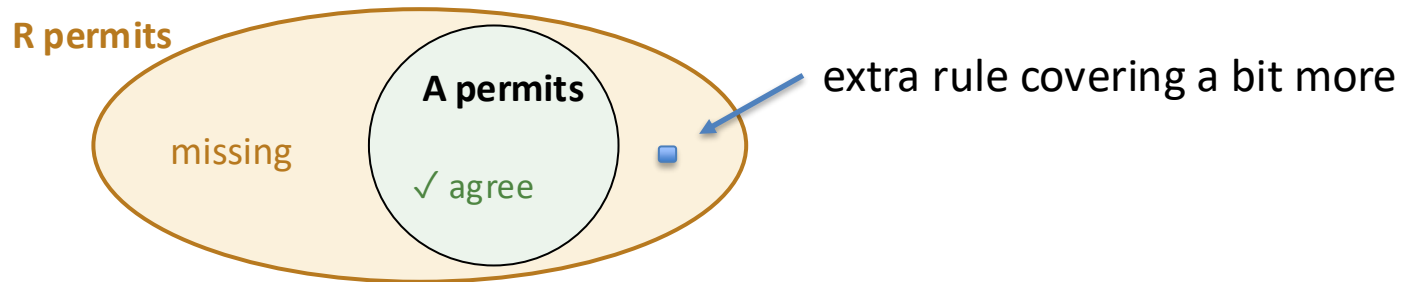
Repair Patch List

```
Step 1 - DELETE rule:  tcp 2-2 from 10.0.0.1/32      (Phase 1)
Step 2 - ADD      rule:  tcp 1-1 from 10.0.0.1/32      (Phase 2)
```

Your task 2 job: understand `repair_concrete` well enough to explain it on a whiteboard and explain which parts of the code do what.

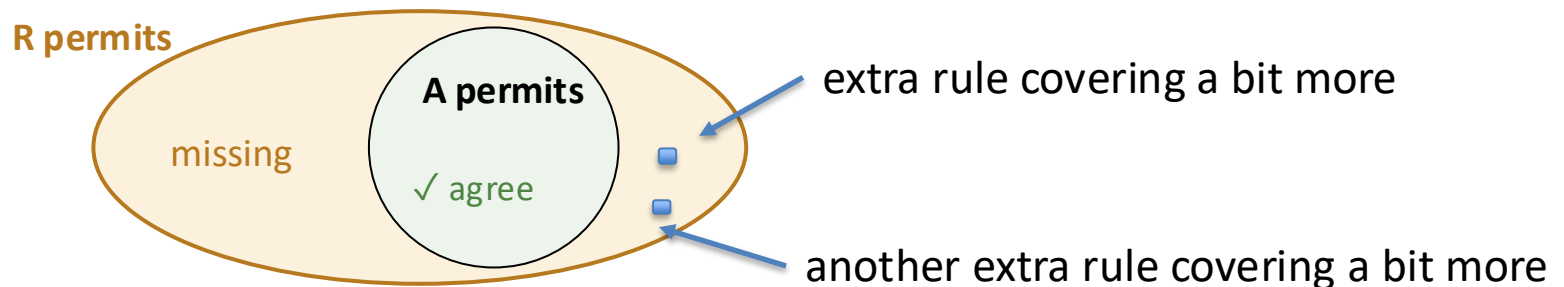
Phase 2: repair_concrete

When adding a rule to our firewall, we could add a rule that matches **exactly one packet**: the packet that we found. This is what the `repair_concrete` algorithm we give you does



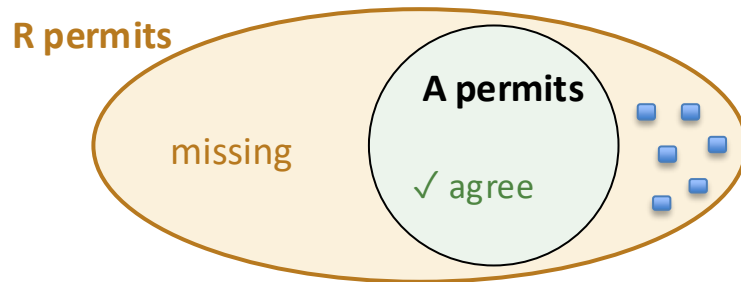
Phase 2: repair_concrete

When adding a rule to our firewall, we could add a rule that matches **exactly one packet**: the packet that we found. This is what the `repair_concrete` algorithm we give you does



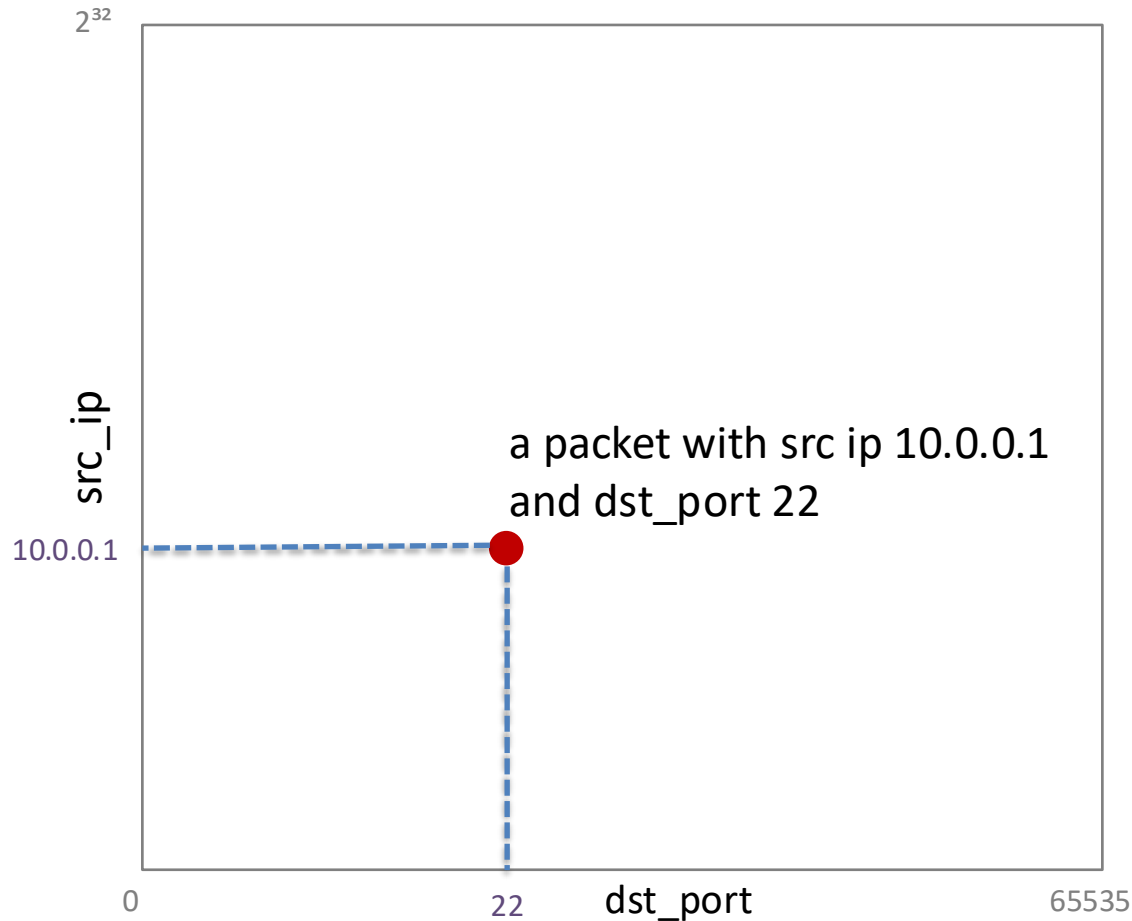
Phase 2: repair_concrete

When adding a rule to our firewall, we could add a rule that matches **exactly one packet**: the packet that we found. This is what the repair_concrete algorithm we give you does



If the missing space is large, then adding one concrete rule at a time takes way too long! Must compute new rules that cover many packets at once!

Packets are Points in Packet Space



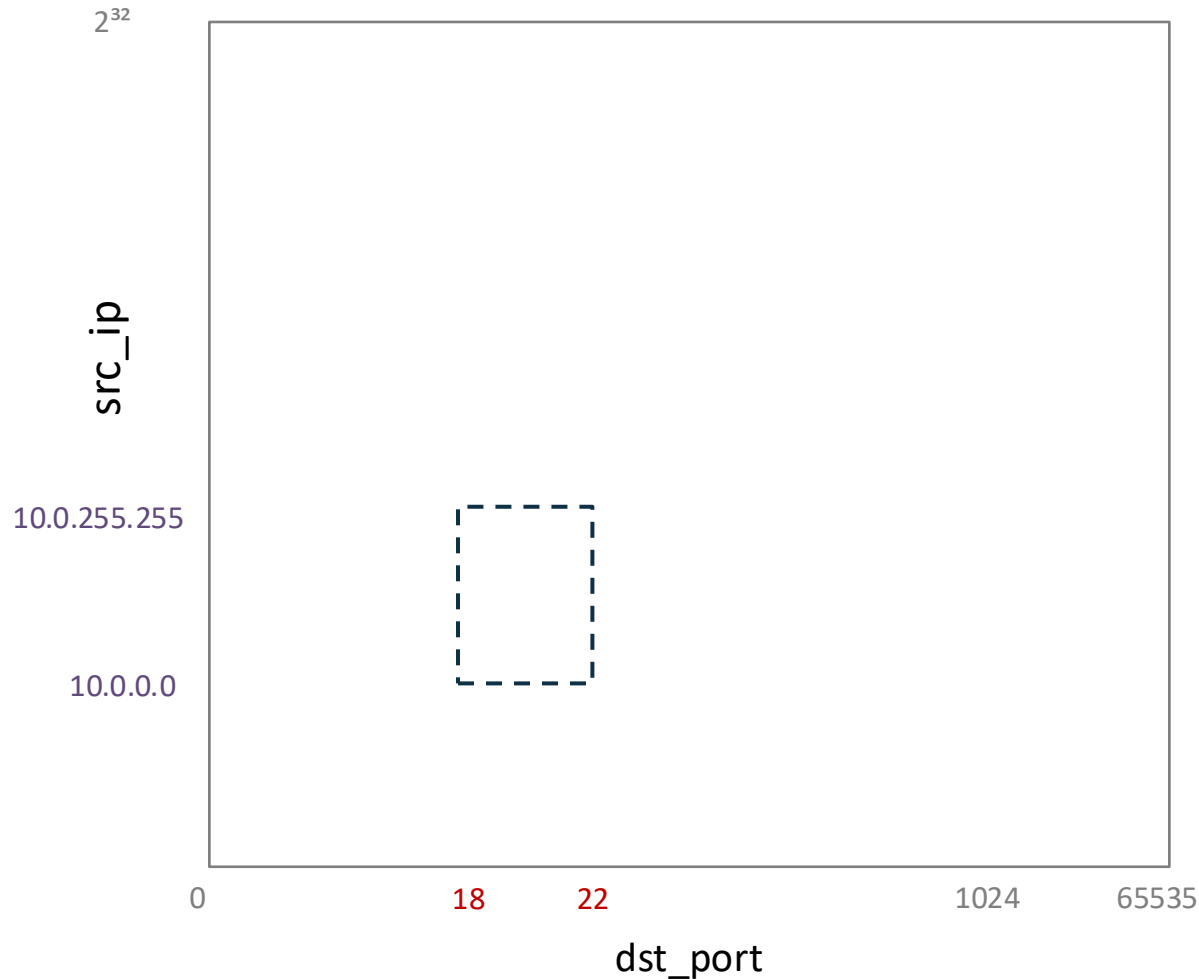
The diagram on the left represents the space of packets.

Dimensions of the space:

- x-axis: ports from 0 .. 65535
- y-axis: IPs from 0 .. 2^{32}

(There are 2 copies of the x-y space. One copy for TCP protocol and one copy for UDP protocol. Pretend the TCP copy is shown.)

Rules Can be Boxes in Packet Space

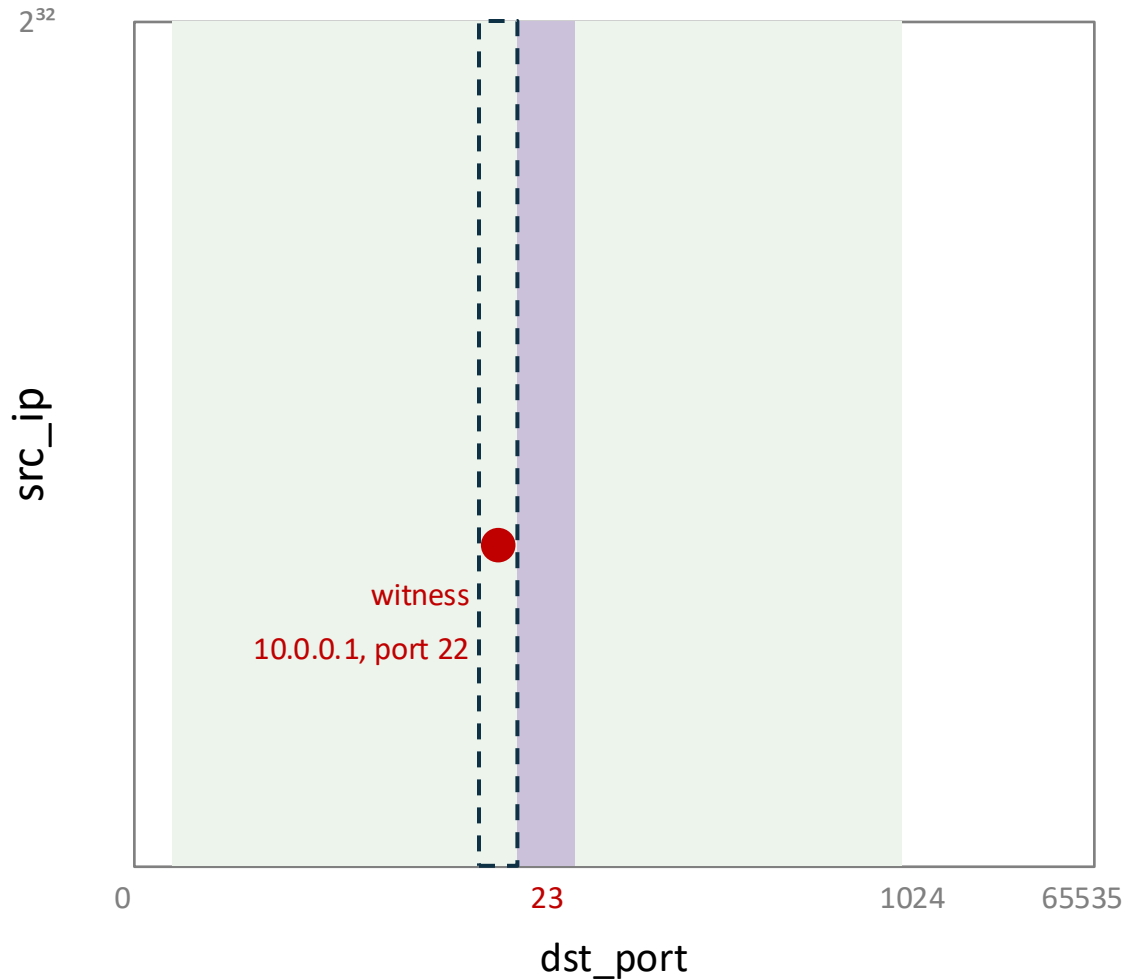


An AWS rule covers an area:

- a range of ports along the x-axis
- a set of `src_ips` on the y-axis

(3rd dimension not shown in picture: protocol is either TCP or UDP)

Goal Find a Big Legal Box that Covers a Packet



Situation:

Green: reference permits

Purple: reference denies

Concrete rule: just the red dot.

Dashed:

`tcp 22-22 from 0.0.0.0/0.`

Goal: Find a big box around the red witness, while staying out of purple.

An algorithm to answer that is your job for task 3.

Two Strategies Worth Exploring

Start big and shrink

- Begin with the rule that matches **every** packet.
- Cut it down one dimension at a time until it matches nothing the reference denies.
- Never cut away the witness.
- Where to cut? Ask what each reference rule says about packets near the witness.

Start small and grow

- Begin with the rule that matches **only the witness**: the concrete rule.
- Grow it one dimension at a time.
- Stop before it matches a packet it should not.
- How far is too far? The solver can tell you: `find_over_permissive` asks almost exactly this question.

Trade-offs to think about: how many solver calls? How much does it depend on the shape of the reference? How large are the rules it finds?

What We Expect for Task 3

There are many possible solutions. A perfect one is **not** required. You should:

- explore one or more ideas;
- repair **some** examples that `repair_concrete` cannot;
- be able to show a step-by-step run of your algorithm on a small example;
- know when it works well, and when it works poorly, and why.

The **hard** test suite has references with any-source rules: the cases where concrete repair runs out of budget.

Challenge: what is the hardest rule set you can repair in one minute of compute? Share your examples with the class.

The Debrief

Teams of 2–3. After hand-in, each team meets a TA.

Every member must be able to explain the code, yours and ours, without an LLM.

Questions might include:

- Encode a small ACL as a formula, on the whiteboard.
- Walk through `check_equivalence`, line by line.
- Run `repair_concrete` by hand on a small example.
- Give an example where `repair_concrete` fails, and explain why.
- Show where your algorithm beats it, and why.

Scale	
4	exceptional
3	solid, well prepared
2	fuzzy or missing details
1	not enough preparation
0	no reasonable answer

*Clear and concise beats long.
Practise your explanations in
advance.*

Guidance

Intended as a project to work on in teams of 2-3.

Step 1: Read STUDENTS.md

Step 2: Investigate the code base yourself first.

If you are having trouble understanding what a function does, an LLM may help.

Step 3: Equivalence checking

Step 4: Repair

- Think for yourself how you might craft a solution at a high level of abstraction.
- Try working through how your algorithm idea might work on a specific, concrete example
- Try coding it yourself. Fall back on LLM support when you need it.
- Understand deeply any LLM-generated code. See what it does on a specific example.
- Reach out to a TA and your prof for help!
- We are unsure how difficult this task will turn out to be!
- Practice explaining how your algorithms work to your teammates
- Practice running through what happens on a specific example