# Event-B: Introduction and First Steps[1]

Manuel Carro
manuel.carro@upm.es

IMDEA Software Institute &
Universidad Politécnica de Madrid

[1] Many slides borrowed from J. R. Abrial



## Conventions

I will sometimes use boxes with different meanings.

- Quiz to do together during the lecture.

Q: What happens in this case? (1)

aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa

- Material / solutions that I want to develop during the lecture.

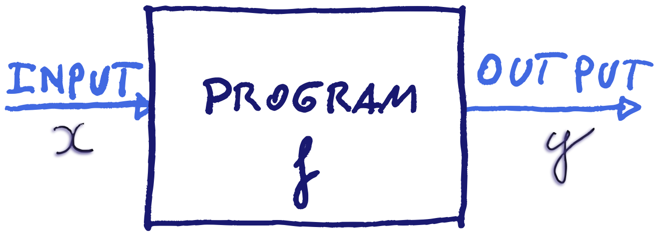
Something to complete here (2)

aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa

**Event B**

*An industry-oriented method, language, and set of supporting tools to describe systems of interacting, reactive software, hardware components, and their environment, and to reason about them.*

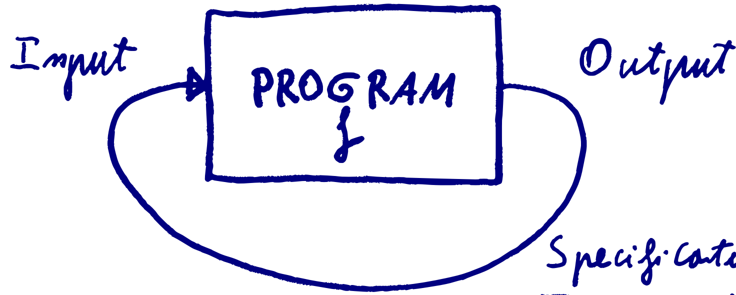
## Sequential vs. reactive software




$y = f(x)$

(or "$R_f(x, y)$ is true" for a logic-based view of computation)

## Sequential vs. reactive software



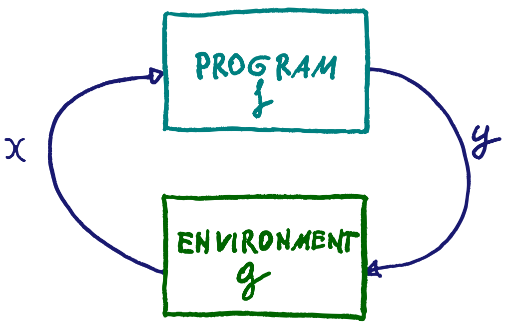

$x_{i+1} = f(x_i)$

Specification?

Termination?

Correctness?

## Sequential vs. reactive software

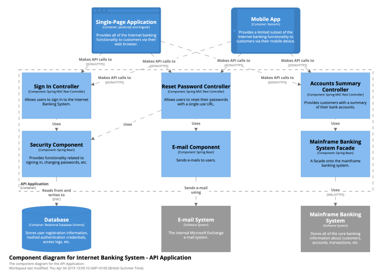



$y_0 = f(x_0),\ x_1 = g(y_0),\ y_1 = f(x_1),\ x_2 = g(y_1), \ldots$

Effects of environment?

## Industrial systems: usual characteristics

- Functionality often not too complex.
  - Algorithms / data structures relatively simple.
  - Underlying maths of reasonable complexity.
- Requirements document usually poor.
- Reactive and concurrent by nature.
  - But often coarse: protecting (large) critical regions often enough.




- Many special cases.
- Communication with hardware / environment involved.
- Many details ($\approx$ properties to ensure) to be taken into account.
- Large (in terms of LOCs).

Producing correct (software) systems hard — but not necessarily from a theoretical point of view.

## Typical approaches and problems

### Usual approach

- Choose a platform.
- Write software specifications (which often neglect or under-represent the environment).
- Design by cutting in small pieces with well-defined communication.
- Code and test / verify units.
- Integrate and test.

## Typical approaches and problems

### Usual approach

- Choose a platform.
- Write software specifications (which often neglect or under-represent the environment).
- Design by cutting in small pieces with well-defined communication.
- Code and test / verify units.
- Integrate and test.

### Pitfalls

- Often too many details / interactions / properties to prove.
- Cutting in pieces: poor job in taming complexity.
  - Small pieces: easy to prove them right.
  - Additional relationships created!
  - Overall complexity not reduced.
- Modeling environment?
  E.g., we expect a car driver to stop at a red light.
- Result: system as a whole never verified.

## The Event B approach

### Complexity: Model Refinement

- System built incrementally, monotonically.
  - Take into account subset of requirements at each step.
  - Build model of a *partial* system.
  - Prove its correctness.
- **Add** requirements to the model, ensure correctness:
  - The requirements correctly captured by the new model.
  - New model preserves properties of previous model.

### Details: Tool Support

- Tool to edit Event B models (Rodin).
- Generates *proof obligations*: theorems to be proved to ensure correctness.
- Interfaced with (interactive) theorem provers.
- Extensible.

## Basic ideas

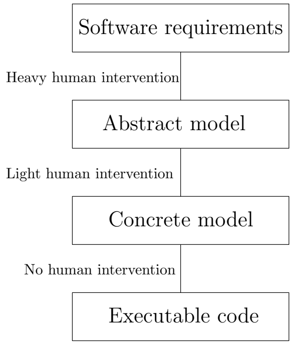
- Model: formal description of a discrete system.
  - Formal: mechanism to decide whether some properties hold
  - Discrete: can be represented as a transition system

## Basic ideas

- Model: formal description of a discrete system.
  - Formal: mechanism to decide whether some properties hold
  - Discrete: can be represented as a transition system
- Formalization contains models of:
  - The future software components
  - The future equipments surrounding these components

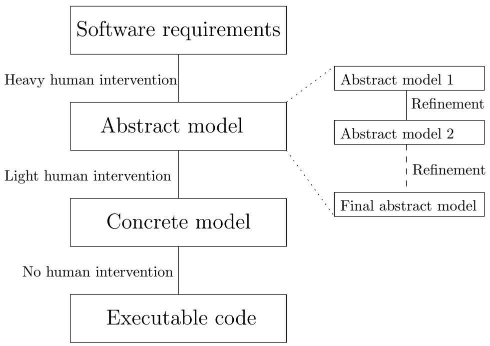
## Refinement

- Refinement allows us to build model gradually.
- Ordered sequence of more precise models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.
- Useful analogy: looking through a microscope.



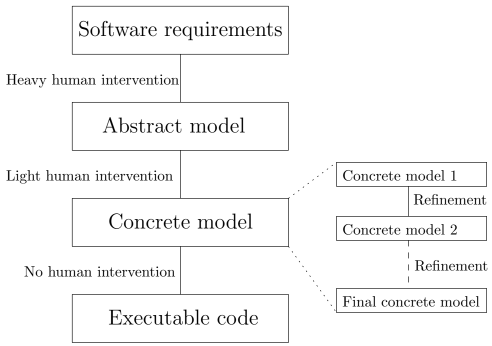

## Refinement

- Refinement allows us to build model gradually.
- Ordered sequence of more precise models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.
- Useful analogy: looking through a microscope.



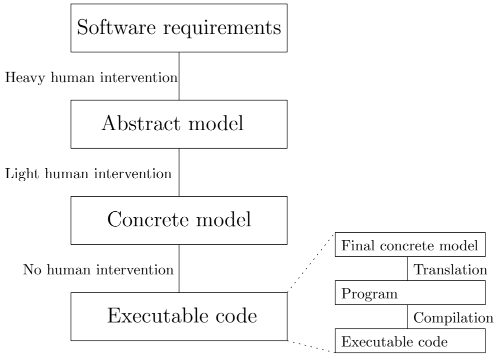

## Refinement

- Refinement allows us to build model gradually.
- Ordered sequence of more precise models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.
- Useful analogy: looking through a microscope.



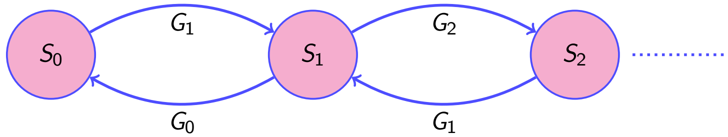
## Refinement

- Refinement allows us to build model gradually.
- Ordered sequence of more precise models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.
- Useful analogy: looking through a microscope.




## Models and states

A discrete model is made of states




What is its relationship with a regular program?

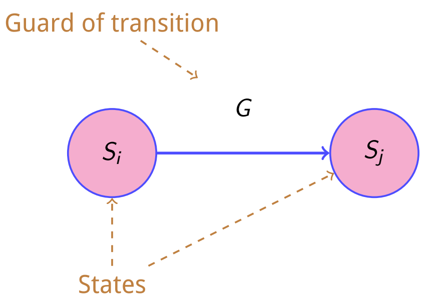
- States are represented by constants and variables

$$S_i = \langle c_1, \ldots, c_n, v_1, \ldots, v_m \rangle$$

- Relationships among constants and variables written using set-theoretic expressions

## States and transitions

- Transitions between states: triggered by events
- Events: made of guards and actions
  - Guard ($G_i$) denote enabling conditions of events
  - Actions denote how state is modified by event
- Guards and actions written with set-theoretic expressions (e.g., first-order, classical logic).
- Event B based on set theory.




Examples:

$S_i \equiv x = 0 \wedge y = 7$
$S_i \equiv x, y \in \mathbb{N} \wedge x < 4 \wedge y < 5 \wedge x + y < 7$

Write extensional definition for the latter

## A simple example – informal introduction!

Search for element `k` in array `f` of length `n`, assuming `k` is in `f`.

Constants / Axioms

```
CONST n ∈ ℕ
CONST f ∈ 1..n ⟶ ℕ
CONST k ∈ ran(f)
```

Variables / Invariants

```
VARIABLE i ∈ 1..n
```

```
Event Search
  when
    i < n ∧ f(i) ≠ k
  then
    i := i + 1
  end
```

```
Event Found
  when
    f(i) = k
  then
    skip
  end
```

(initialization of `i` not shown for brevity)

## Events

```
Event EventName
  when
    guard:  G(v, c)
  then
    action:  v := E(v, c)
  end
```

- Executing an event (normally) changes the system state.
- An event can fire when its guard evaluates to true.
- G(v, c) predicate that enables EventName
- v := E(v, c) is a state transformer.
  - Formally, a predicate $Act_E$(v, c, v')
  - v' is renamed to v after the predicate.

## Intuitive operational interpretation

```
Initialize;
while (some events have true guards) {
  Choose one such event;
  Modify the state accordingly;
}
```

```
Event EventName
  when
    guard:  G(v, c)
  then
    action:  v := E(v, c)
  end
```

- Now: **informal** Event B semantics.
- Actual Event B semantics based on set theory and invariants — Later!

- An event execution takes no time.
  - No two events occur simultaneously.
- If all guards false, system stops.
- Otherwise: choose one event with true guard, execute action, modify state.
- Previous phase repeated (if possible).

Fairness: what is it? What should we expect?

## Comments on the operational interpretation

- Stopping is not necessary: a discrete system may run forever.
- This interpretation is just given here for informal understanding
- The meaning of such a discrete system will be given by the proofs which can be performed on it (next lectures).[2]

**On using sequential code**

*To help understanding, we will now write some sequential code first, translate it into Event B, and then proving correctness. This does not follow Event B workflow, which goes in the opposite direction: write Event B models and derive sequential / concurrent code from them.*

[2] J. R. Abrial: *The B method: assigning programs to meanings.*

## Running example (sequential code)

$$a = \left\lfloor \frac{b}{c} \right\rfloor$$

- Characterize it: we want to define integer division, without using division.

Q: specification of division (3)

$$\forall b \forall c\, [b \in \mathbb{N} \wedge c \in \mathbb{N} \wedge c > 0 \Rightarrow \exists a \exists r\, [a \in \mathbb{N} \wedge r \in \mathbb{N} \wedge r < c \wedge b = c \times a + r]]$$

It is useful to categorize the specification as assumptions (preconditions)

$$b \in \mathbb{N} \wedge c \in \mathbb{N} \wedge c > 0$$

and results (postconditions)

$$a \in \mathbb{N} \wedge r \in \mathbb{N} \wedge r < c \wedge b = c \times a + r$$

Input / output / variables / constants / types?

## Two Math Notes

### Zero

*There is no universal agreement about whether to include zero in the set of natural numbers. Some authors begin the natural numbers with 0, corresponding to the non-negative integers 0, 1, 2, 3, ..., whereas others start with 1, corresponding to the positive integers 1, 2, 3, ... This distinction is of no fundamental concern for the natural numbers as such.*

I will assume that $0 \in \mathbb{N}$. That is the convention in computer science.

## Two Math Notes

### Zero

*There is no universal agreement about whether to include zero in the set of natural numbers. Some authors begin the natural numbers with 0, corresponding to the non-negative integers 0, 1, 2, 3, ..., whereas others start with 1, corresponding to the positive integers 1, 2, 3, ... This distinction is of no fundamental concern for the natural numbers as such.*

I will assume that $0 \in \mathbb{N}$. That is the convention in computer science.

.

If you write $\forall b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \cdot \exists a \in \mathbb{N}, r \in \mathbb{N}, r < c \cdot b = c \times a + r$ remember:

- Commas mean conjunction.
- Nesting may need disambiguation.
- $\forall x \in D \cdot P(x)$ means $\forall x[x \in D \Rightarrow P(x)]$
- $\exists x \in D \cdot P(x)$ means $\exists x[x \in D \land P(x)]$

See https://twitter.com/lorisdanto/status/1354128808740327425?s=20
and https://twitter.com/lorisdanto/status/1354214767590842369?s=20

## Programming integer division

- We have addition and substracion
- We have a simple procedural language
- Variables, assignment, loops, if-then-else, + & -, arith. operators, ...

Q: integer division code (4)

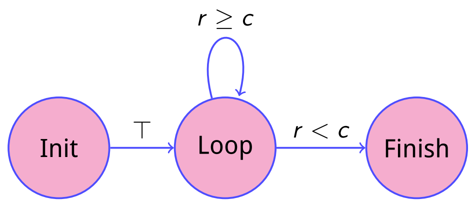
```
a := 0
r := b
while r >= c
   r := r - c
   a := a + 1
```

## Programming integer division

- We have addition and substracion
- We have a simple procedural language
- Variables, assignment, loops, if-then-else, + & -, arith. operators, ...

Q: integer division code (5)

```
a := 0
r := b
while r >= c
   r := r - c
   a := a + 1
```




Copy the code! We will need it!

This step is not taken in Event B. We are writing this code only for illustration purposes.

## Towards events

Template

```
Event EventName
  when
    G(v, c)
  then
    v := E(v, c)
  end
```

Code

```
a := 0
r := b
while r >= c
  r := r - c
  a := a + 1
end
```

- Special initialization event (**INIT**).
- Sequential program (special case):
  - *Finish* event, *Progress* events
  - Guards exclude each other (determinism) Prove!
  - Non-deadlock: some guard always true Prove!
  - A variable is reduced (termination) Prove!

Q: integer division events (6)

```
Event INIT            Event Progress                Event Finish
  a, r = 0, b           when                          when
end                       r >= c                        r < c
                        then                          then
                          r, a := r - c, a + 1          skip
                        end                           end
```

## Categorizing elements

| Constants | Axioms (Write them down separately!) |
|---|---|
| Q: constants (7)<br>b<br>c | Q: axioms (8)<br>$b \in \mathbb{N}$<br>$c \in \mathbb{N}$<br>$c > 0$ |
| **Variables** | **Invariants** |
| Q: variables (9)<br>a<br>r | Later! |

```
Event INIT          Event Progress              Event Finish
  a, r = 0, b         when r >= c                 when r < c
end                   then                        then
                        r, a := r - c, a + 1        skip
                      end                         end
```

## Proving correctness

How do **you** prove your programs correct?

## Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

```
x := x + y;
y := x − y;
x := x − y;
```

## Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

$\{x = a, y = b\}$

```
x := x + y;
y := x - y;
x := x - y;
```

$\{x = b, y = a\}$

## Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

$\{x = a, y = b\}$

`x := x + y;` $\{x = a + b, y = b\}$
`y := x - y;`
`x := x - y;`
$\{x = b, y = a\}$

## Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

$\{x = a, y = b\}$

`x := x + y;` $\{x = a + b, y = b\}$
`y := x - y;` $\{x = a + b, y = a\}$
`x := x - y;`
$\{x = b, y = a\}$

## Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

$\{x = a, y = b\}$

`x := x + y;` $\{x = a + b, y = b\}$
`y := x - y;` $\{x = a + b, y = a\}$
`x := x - y;` $\{x = b, y = a\}$
$\{x = b, y = a\}$

## Proving correctness: invariants in a nutshell

**Loops:** much more difficult

- # iterations unknown. (remember Collatz's conjecture)

```
while r >= c do

    r := r − c
    a := a + 1

end
```

**Invariant:** formula that is "always" true.

- Procedural code: beginning and end of every loop iteration.
- Event-B: after initialization, after every event (essentially same idea).

**Intuitition:**

- If invariant implies postcondition, then we can prove postcondition.
- Nobody gives us invariants.
  - We have to find them.
  - We have to prove they are invariants.

## Proving correctness: invariants in a nutshell

**Loops:** much more difficult

- # iterations unknown. (remember Collatz's conjecture)

```
while r >= c do
    {I(a, r)}
    r := r − c
    a := a + 1
    {I(a, r)}
end
```

$\{I(a,r) \wedge r < c \Rightarrow a = \lfloor \frac{b}{c} \rfloor\}$

**Invariant:** formula that is "always" true.

- Procedural code: beginning and end of every loop iteration.
- Event-B: after initialization, after every event (essentially same idea).

**Intuitition:**

- If invariant implies postcondition, then we can prove postcondition.
- Nobody gives us invariants.
  - We have to find them.
  - We have to prove they are invariants.

## Finding invariants

Which assertions are invariant in our model?

Q: model invariants (10)

$I_1$: $a \in \mathbb{N}$ *// Type invariant*
$I_2$: $r \in \mathbb{N}$ *// Type invariant*
$I_3$: $b = a \times c + r$

One formula that is an invariant for **any** Event-B model / loop.

Q: trivial invariant (11)

$\top$

```
Event INIT          Event Progress                 Event Finish
  a, r = 0, b         when r >= c                    when r < c
end                   then                           then
                        r, a := r - c, a + 1           skip
                      end                            end
```

## Finding invariants

Which assertions are invariant in our model?

Q: model invariants (12)

$I_1$: $a \in \mathbb{N}$ *// Type invariant*
$I_2$: $r \in \mathbb{N}$ *// Type invariant*
$I_3$: $b = a \times c + r$

One formula that is an invariant for **any** Event-B model / loop.

Q: trivial invariant (13)

$\top$

```
Event INIT          Event Progress                 Event Finish
  a, r = 0, b         when r >= c                    when r < c
end                   then                           then
                        r, a := r - c, a + 1           skip
                      end                            end
```

Copy invariants somewhere else – we will need to have them handy

## Invariant preservation in Event B

- Invariants must be true before and after event execution.
- For all event $i$, invariant $j$:

  Establishment:

  $$A(c) \vdash I_j(E_{\mathrm{init}}(v,c),c)$$

  Preservation:

  $$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

  - $A(c)$ axioms
  - $G_i(v,c)$ guard of event $i$
  - $I_j(v,c)$ invariant $j$
  - $I_{1\ldots n}(v,c)$ all the invariants
  - $E_i(v,c)$ result of action $i$

**Sequent**

$$\Gamma \vdash \Delta$$

Show that $\Delta$ can be proved using assumptions $\Gamma$

**Invariant preservation**

If an invariant holds and the guards of an event are true and we execute the event's action, the invariant should hold.

## Invariant preservation proofs

- Invariant preservation proven using model and math axioms.
- Three invariants & three events: nine proofs
- Named as e.g. $\mathrm{E_{Progress}/I_2/INV}$
  - Other proofs necessary later

$\mathrm{E_{INIT}}$ / $\mathrm{I_1}$ / INV

INIT I1 invariant proof (14)

$$\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash 0 \in \mathbb{N}}\ \text{MON}$$

$\mathrm{E_{INIT}}$ / $\mathrm{I_2}$ / INV

INIT I2 invariant proof (15)

$$\dfrac{\dfrac{}{b \in \mathbb{N} \vdash b \in \mathbb{N}}\ \text{HYP}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash b \in \mathbb{N}}\ \text{MON}$$

```
Event INIT
  a, r = 0, b
end
```

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$\mathrm{E_{INIT}}$ / $\mathrm{I_3}$ / INV

INIT I3 invariant proof (16)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash b = b}\ \text{EQL}}{\vdash b = 0 + b}\ \text{Arith}}{\vdash b = 0 \times c + b}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash b = 0 \times c + b}\ \text{MON}$$

$\mathrm{E_{Progress}}$ / $\mathrm{I_1}$ / INV

Progress I1 invariant proof (17)

$$\dfrac{\dfrac{}{a \in \mathbb{N} \vdash a + 1 \in \mathbb{N}}\ \text{P1}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, r \in \mathbb{N}, b = a \times c + r, a \in \mathbb{N} \vdash a + 1 \in \mathbb{N}}\ \text{MON}$$

```
Event INIT
  a, r = 0, b
end
```

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Sequents

- Mechanize proofs
  - Humans "understand"; proving is tiresome and error-prone
  - Computers manipulate symbols
- How can we mechanically construct correct proofs?
  - Every step crystal clear
  - For a computer to perform
- Several approaches
- For Event B: sequent calculus
  - To read: [Pau] (available at course web page), at least Sect. 3.3 to 3.5 , 6.4, and 6.5. Note: when we use $\Gamma \vdash \Delta$, Paulson uses $\Gamma \Rightarrow \Delta$.
  - Also: [Oric, Orib], available at the course web page.
- Admissible deductions: inference rules.

## Inference rules

- An inference rule is a tool to build a formal proof.
  - It not only tells you whether $\Gamma \vdash \Delta$: it tells you how.
- It is denoted by:

$$\frac{A}{C}\ \mathsf{R}$$

- A is a (possibly empty) collection of sequents: the antecedents.
- C is a sequent: the consequent.
- R is the name of the rule.

The proofs of each sequent of A

——— together give you ———

a proof of sequent C

## An example of inference rule

**Note:** not exactly the inference rules we will use.
Only an intuitive example.

- A(lice) and B(ob) are siblings:

$$\frac{\text{C is mother of A} \qquad \text{C is mother of B}}{\text{A and B are siblings}}\ \text{Sibling-M}$$

$$\frac{\text{C is father of A} \qquad \text{C is father of B}}{\text{A and B are siblings}}\ \text{Sibling-F}$$

- Note: we do not consider the case that, e.g., C is a father and a mother.

## Proof of sequent $S1$



$$\frac{}{S2}\mathbf{r1} \quad \frac{S7}{S4}\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\mathbf{r3} \quad \frac{}{S5}\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\mathbf{r5} \quad \frac{}{S6}\mathbf{r6} \quad \frac{}{S7}\mathbf{r7}$$

$S1$
**?**

## Proof of Sequent $S1$



$$\frac{}{S2}\mathbf{r1} \quad \frac{S7}{S4}\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\mathbf{r3} \quad \frac{}{S5}\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\mathbf{r5} \quad \frac{}{S6}\mathbf{r6} \quad \frac{}{S7}\mathbf{r7}$$

$S1$
**r3**
$\nearrow \uparrow \nwarrow$
$S2$ $S3$ $S4$
**?** **?** **?**

$$\frac{}{S2}\,\mathbf{r1} \quad \frac{S7}{S4}\,\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\,\mathbf{r3} \quad \frac{}{S5}\,\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\,\mathbf{r5} \quad \frac{}{S6}\,\mathbf{r6} \quad \frac{}{S7}\,\mathbf{r7}$$

$S1$
**r3**
↗ ↑ ↖
$S2$ $S3$ $S4$
**r1** **?** **?**



$$\frac{}{S2}\,\mathbf{r1} \quad \frac{S7}{S4}\,\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\,\mathbf{r3} \quad \frac{}{S5}\,\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\,\mathbf{r5} \quad \frac{}{S6}\,\mathbf{r6} \quad \frac{}{S7}\,\mathbf{r7}$$

$S1$
**r3**
↗ ↑ ↖
$S2$ $S3$ $S4$
**r1** **r5** **?**
↗ ↑
$S5$ $S6$
**?** **?**



$$\frac{}{S2}\,\mathbf{r1} \quad \frac{S7}{S4}\,\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\,\mathbf{r3} \quad \frac{}{S5}\,\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\,\mathbf{r5} \quad \frac{}{S6}\,\mathbf{r6} \quad \frac{}{S7}\,\mathbf{r7}$$

$S1$
**r3**
↗ ↑ ↖
$S2$ $S3$ $S4$
**r1** **r5** **?**
↗ ↑
$S5$ $S6$
**r4** **?**



$$\frac{}{S2}\,\mathbf{r1} \quad \frac{S7}{S4}\,\mathbf{r2} \quad \frac{S2 \quad S3 \quad S4}{S1}\,\mathbf{r3} \quad \frac{}{S5}\,\mathbf{r4} \quad \frac{S5 \quad S6}{S3}\,\mathbf{r5} \quad \frac{}{S6}\,\mathbf{r6} \quad \frac{}{S7}\,\mathbf{r7}$$

$S1$
**r3**
↗ ↑ ↖
$S2$ $S3$ $S4$
**r1** **r5** **?**
↗ ↑
$S5$ $S6$
**r4** **r6**

## Proof of Sequent $S1$



$$\frac{}{S2}\textbf{r1} \quad \frac{S7}{S4}\textbf{r2} \quad \frac{S2\ \ S3\ \ S4}{S1}\textbf{r3} \quad \frac{}{S5}\textbf{r4} \quad \frac{S5\ \ S6}{S3}\textbf{r5} \quad \frac{}{S6}\textbf{r6} \quad \frac{}{S7}\textbf{r7}$$

$$\begin{array}{ccc}
 & S1 & \\
 & \textbf{r3} & \\
\nearrow & \uparrow & \nwarrow \\
S2 & S3 & S4 \\
\textbf{r1} & \textbf{r5} & \textbf{r2} \\
\nearrow & \uparrow & \uparrow \\
S5 & S6 & S7 \\
\textbf{r4} & \textbf{r6} & \textbf{?}
\end{array}$$

## Proof of Sequent $S1$



$$\frac{}{S2}\textbf{r1} \quad \frac{S7}{S4}\textbf{r2} \quad \frac{S2\ \ S3\ \ S4}{S1}\textbf{r3} \quad \frac{}{S5}\textbf{r4} \quad \frac{S5\ \ S6}{S3}\textbf{r5} \quad \frac{}{S6}\textbf{r6} \quad \frac{}{S7}\textbf{r7}$$

$$\begin{array}{ccc}
 & S1 & \\
 & \textbf{r3} & \\
\nearrow & \uparrow & \nwarrow \\
S2 & S3 & S4 \\
\textbf{r1} & \textbf{r5} & \textbf{r2} \\
\nearrow & \uparrow & \uparrow \\
S5 & S6 & S7 \\
\textbf{r4} & \textbf{r6} & \textbf{r7}
\end{array}$$

## Recording the Proof of Sequent $S1$



$$\frac{}{S2}\textbf{r1} \quad \frac{S7}{S4}\textbf{r2} \quad \frac{S2\ \ S3\ \ S4}{S1}\textbf{r3} \quad \frac{}{S5}\textbf{r4} \quad \frac{S5\ \ S6}{S3}\textbf{r5} \quad \frac{}{S6}\textbf{r6} \quad \frac{}{S7}\textbf{r7}$$

$$\begin{array}{ccc}
 & S1 & \\
 & \textbf{r3} & \\
\nearrow & \uparrow & \nwarrow \\
S2 & S3 & S4 \\
\textbf{r1} & \textbf{r5} & \textbf{r2} \\
\nearrow & \uparrow & \uparrow \\
S5 & S6 & S7 \\
\textbf{r4} & \textbf{r6} & \textbf{r7}
\end{array}$$

- The proof is a tree

## Deduction systems

- There are many formal deduction systems [Ben12, Sect. 3.9].
- We will use a variant of the so-called *Gentzen* deduction systems.

### Sequent $\Gamma \vdash \Delta$ in a Gentzen system

- $\Gamma$: (possibly empty) collection of formulas (the hypotheses)
- $\Delta$: collection of formulas (the goal)
- Objective: show that, under hypotheses $\Gamma$, some formula(s) in $\Delta$ can be proven.

$\Gamma \equiv P_1, P_2, \ldots, P_n$ stands for $P_1 \wedge P_2 \wedge \ldots \wedge P_n$

$\Delta \equiv Q_1, Q_2, \ldots, Q_m$ s.f. $Q_1 \vee Q_2 \vee \ldots \vee Q_m$

$\boxed{P_1, P_2, \ldots, P_n \vdash Q_1, Q_2, \ldots, Q_m}$
is
$\boxed{P_1 \wedge P_2 \wedge \ldots \wedge P_n \vdash Q_1 \vee Q_2 \vee \ldots \vee Q_m}$

- We will use a proof calculus where the goal is a single formula.
- More constructive proofs — but see [Orib, Section 11.2] for interesting remarks.

## Inside a sequent

- We need a language to express hypothesis and goals.
  - Not formally defined yet
  - We will assume it is first-order, classical logic
  - Recommended references: [Pau, HR04, Ben12]
- We need a way to determine if (and how) $\Delta$ can prove $\Gamma$.
  - Inference rules.

## Logic and inference rules

### Inference rules

**Structural**

- Hypothesis
- Monotony
- Cut

**Depending on logic**

- Propositional
- First order
- Temporal
- Higher order
- ...

**For specific theories**

- Sets
- Relations
- Functions
- (Linear) Arithmetic
- Reals
- Strings
- Arrays
- Bitvectors
- Records
- Difference logic
- Inductive data types
- Empty theory
- ...

## Structural inference rules

- Three structural inference rules, independent of the predicate language.

**HYPothesis**

$$\frac{}{H, P \vdash P} \text{ HYP}$$

If the goal is among the hypothesis, we are done.

**MONotony**

$$\frac{H \vdash Q}{H, P \vdash Q} \text{ MON}$$

If goal proven without hypothesis $P$, then can be proven with $P$.

**CUT**

$$\frac{H \vdash P \qquad H, P \vdash Q}{H \vdash Q} \text{ CUT}$$

A goal can be proven with an intermediate deduction $P$. Nobody tells us what is $P$ or how to come up with it. It *cuts* the proof into smaller pieces. (*Cut Elimination Theorem*)

## More rules

- There are many other inference rules for:
  - Logic itself (propositional / predicate logic)
    - Look at the slides / documents in the course web page
  - reasoning on arithmetic (Peano axioms),
  - reasoning on sets,
  - reasoning on functions,
  - ...
- We will not list all of them here (see online documentation).
- We may need to explain them as they appear.
- But a mechanical prover has them as “inside knowledge” (plus tactics, strategies)

## The propositional language: basic constructs

- Given predicates $P$ and $Q$, we can construct:
- NEGATION: $\neg P$
- CONJUNCTION: $P \wedge Q$
- IMPLICATION: $P \Rightarrow Q$

- Precedence: $\neg, \wedge, \Rightarrow$.
  - Examples
- Parenthesis added when needed.
  - If in doubt: add parentheses!
- Can you build the truth tables?
- $\vee, \Leftrightarrow$ are defined based on them.
  - Define them
  - Can we use a **single** connective?

## The propositional language: rules for conjunction

$$\frac{H \vdash Q \qquad H \vdash P}{H \vdash P \wedge Q} \text{ AND-R}$$

*A conjunction on the RHS needs both branches of the conjunction be proven independently of each other.*

$x \in \mathbb{N}1, y \in \mathbb{N}1, x + y < 5 \vdash x < 4 \wedge y < 4$

## The propositional language: rules for conjunction

$$\frac{H \vdash Q \qquad H \vdash P}{H \vdash P \wedge Q} \text{ AND-R}$$

*A conjunction on the RHS needs both branches of the conjunction be proven independently of each other.*

$x \in \mathbb{N}1, y \in \mathbb{N}1, x + y < 5 \vdash x < 4 \wedge y < 4$

$$\frac{H, P, Q \vdash R}{H, P \wedge Q \vdash R} \text{ AND-L}$$

*By definition of sequent.*

## The propositional language: rules for disjunction

$$\frac{H, Q \vdash R \qquad H, P \vdash R}{H, P \vee Q \vdash R} \text{ OR-L}$$

*A disjunction on the LHS needs both branches of the disjunction be discharged separately.*

$(x < 0 \wedge y < 0) \vee x + y > 0 \vdash x \times y > 0$

Counterxample?

LHS: **all** conditions in which RHS has to hold. Removing part of disjunction makes “condition space” smaller (removing part of conjunction makes the “condition space” larger, more general). Proofs with more general assumptions are valid for less general assumptions, not the other way around.

## The propositional language: rules for disjunction

$$\frac{H, Q \vdash R \qquad H, P \vdash R}{H, P \vee Q \vdash R} \text{ OR-L}$$

*A disjunction on the LHS needs both branches of the disjunction be discharged separately.*

$(x < 0 \wedge y < 0) \vee x + y > 0 \vdash x \times y > 0$

Counterexample?

LHS: **all** conditions in which RHS has to hold. Removing part of disjunction makes "condition space" smaller (removing part of conjunction makes the "condition space" larger, more general). Proofs with more general assumptions are valid for less general assumptions, not the other way around.

$$\frac{H \vdash P}{H \vdash P \vee Q} \text{ OR-R1} \qquad \frac{H \vdash Q}{H \vdash P \vee Q} \text{ OR-R2}$$

*A disjunction on the RHS only needs **one** of the branches to be proven. There is a rule for each branch.*

## The propositional language: rules for disjunction

$$\frac{H, Q \vdash R \qquad H, P \vdash R}{H, P \vee Q \vdash R} \text{ OR-L}$$

*A disjunction on the LHS needs both branches of the disjunction be discharged separately.*

$(x < 0 \wedge y < 0) \vee x + y > 0 \vdash x \times y > 0$

Counterexample?

LHS: **all** conditions in which RHS has to hold. Removing part of disjunction makes "condition space" smaller (removing part of conjunction makes the "condition space" larger, more general). Proofs with more general assumptions are valid for less general assumptions, not the other way around.

$$\frac{H \vdash P}{H \vdash P \vee Q} \text{ OR-R1} \qquad \frac{H \vdash Q}{H \vdash P \vee Q} \text{ OR-R2}$$

*A disjunction on the RHS only needs **one** of the branches to be proven. There is a rule for each branch.*

$$\frac{H, \neg P \vdash Q}{H \vdash P \vee Q} \text{ NEG}$$

*Part of a disjunctive goal can be negated, moved to the hypotheses, and used to discharge the proof. Related to $\neg P \vee Q$ being $P \Rightarrow Q$.*

$x \in \mathbb{N}, y \in \mathbb{N}, x + y > 1, y > x \vdash x > 0 \vee y > 1$

## The propositional language: rules for negation

$$\frac{}{\perp \vdash Q} \text{ CNTR}$$

$$\frac{}{P, \neg P \vdash Q} \text{ NOT-L}$$

*If we reach to a contradiction in the hypotheses, anything can be proven (principle of explosion). Note: not everyone accepts this – more on that later.*

$$\frac{H, \neg P \vdash \neg Q \qquad H, \neg P \vdash Q}{H \vdash P} \text{ NOT-R}$$

*Reductio ad absurdum: assume the negation of what we want to prove and reach a contradiction. Similarly with $H \vdash \neg P$.*

$P \wedge \neg P \equiv \perp$ (Falsehood) $\qquad P \vee \neg P \equiv \top$ (Truth) $\qquad \top = \neg\perp$

## The propositional language: rules for implication

$$\frac{H \vdash P \qquad H, Q \vdash R}{H, P \Rightarrow Q \vdash R} \text{ IMP-L}$$

*If we want to use $P \Rightarrow Q$, we show that $P$ is deducible from $H$ and that, assuming $Q$, we can infer $R$.*

$$\frac{H, P \vdash Q}{H \vdash P \Rightarrow Q} \text{ IMP-R}$$

*We move the LHS $P$ to the hypotheses. Note that since $P \Rightarrow Q$ is $\neg P \vee Q$, we are applying the NEG rule in disguise.*

$x \in \mathbb{N}, y \in \mathbb{N}, x + y > k \vdash x = k \Rightarrow y > 0$

## Additional rules

Equality axiom

$$\frac{}{\vdash E = E}\ \text{EQL}$$

First Peano axiom

$$\frac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}$$

Equality propagation

$$\frac{H(F), E = F \vdash P(F)}{H(E), E = F \vdash P(E)}\ \text{EQL-LR}$$

Second Peano axiom

$$\frac{}{n \in \mathbb{N} \vdash n + 1 \in \mathbb{N}}\ \text{P1}$$

**Forthcoming proofs and propositional rules**

The following proofs feature variables. Strictly speaking, they are not propositional. We will however not use quantifiers, so we will treat formulas as propositions when applying the previous rules.
We will assume the existence of simple, well-known arithmetic rules.

## Invariant preservation proofs

$E_{Progress}$ / $I_2$ / INV

Progress I2 invariant proof (18)

P0
Arith
MON
EQ-LR
Arith*
Simp-M-Minus
Arith-M-M-R
OR-L
Arith

$$\frac{}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{Progress}$ / $I_2$ / INV

Progress I2 invariant proof (19)

P0
Arith
MON
EQ-LR
Arith*
Simp-M-Minus
Arith-M-M-R
OR-L

$$\frac{\dfrac{}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{Progress}$ / $I_2$ / INV

Progress I2 invariant proof (20)

P0
Arith
MON
EQ-LR
Arith*
Simp-M-Minus
Arith-M-M-R

$$\frac{\dfrac{\dfrac{}{c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{OR-L}}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\mathrm{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (21)

P0
Arith
MON
EQ-LR
$c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

Arith*
Simp-M-Minus
Arith-M-M-R

OR-L
$c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
Arith
$c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
MON
$b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\mathrm{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (22)

P0
Arith
MON
$c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}$
EQ-LR
$c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

Arith*
Simp-M-Minus
Arith-M-M-R

OR-L
$c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
Arith
$c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
MON
$b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\mathrm{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (23)

P0
Arith
$\vdash c - c \in \mathbb{N}$
MON
$c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}$
EQ-LR
$c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

Arith*
Simp-M-Minus
Arith-M-M-R

OR-L
$c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
Arith
$c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
MON
$b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\mathrm{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (24)

P0
$\vdash 0 \in \mathbb{N}$
Arith
$\vdash c - c \in \mathbb{N}$
MON
$c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}$
EQ-LR
$c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

Arith*
Simp-M-Minus
Arith-M-M-R

OR-L
$c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
Arith
$c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$
MON
$b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (25)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}}{\vdash c - c \in \mathbb{N}}\ \text{Arith}}{c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}}\ \text{MON}}{c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{EQ-LR}
\qquad
\dfrac{\dfrac{\dfrac{}{\phantom{x}}\ \text{Arith}^*}{\phantom{x}}\ \text{Simp-M-Minus}}{c \in \mathbb{N}, r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith-M-M-R}$$

$$\dfrac{\dfrac{c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}\ \ (\text{OR-L})}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (26)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}}{\vdash c - c \in \mathbb{N}}\ \text{Arith}}{c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}}\ \text{MON}}{c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{EQ-LR}
\qquad
\dfrac{\dfrac{\dfrac{}{c \in \mathbb{N}, r - c > c - c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}^*}{\phantom{x}}\ \text{Simp-M-Minus}}{c \in \mathbb{N}, r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith-M-M-R}$$

$$\dfrac{\dfrac{c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}\ \ (\text{OR-L})}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_2$ / INV

Progress I2 invariant proof (27)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}}{\vdash c - c \in \mathbb{N}}\ \text{Arith}}{c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}}\ \text{MON}}{c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{EQ-LR}
\qquad
\dfrac{\dfrac{\dfrac{}{c \in \mathbb{N}, r - c > 0, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}^*}{c \in \mathbb{N}, r - c > c - c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Simp-M-Minus}}{c \in \mathbb{N}, r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith-M-M-R}$$

$$\dfrac{\dfrac{c \in \mathbb{N}, r = c \lor r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}\ \ (\text{OR-L})}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\ \text{MON}$$

$I_2$: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_3$ / INV

Progress I3 invariant proof (28)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\phantom{x}}\ \text{HYP}}{\phantom{x}}\ \text{Arith-M-Pl-Dist}}{\phantom{x}}\ \text{Arith-M-Pl-Dist}}{\phantom{x}}\ \text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a + 1) \times c + (r - c)}\ \text{MON}$$

$I_3$: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_3$ / INV

Progress I3 invariant proof (29)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ }\ \text{HYP}}{\ }\ \text{Arith-M-Pl-Dist}}{\ }\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{MON}$$

$I_3$: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_3$ / INV

Progress I3 invariant proof (30)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ }\ \text{HYP}}{\ }\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + r - c}\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{MON}$$

$I_3$: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_3$ / INV

Progress I3 invariant proof (31)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ }\ \text{HYP}}{b = a \times c + r \vdash b = a \times c + c + r - c}\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + r - c}\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{MON}$$

$I_3$: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

$E_{\text{Progress}}$ / $I_3$ / INV

Progress I3 invariant proof (32)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b = a \times c + r \vdash b = a \times c + r}\ \text{HYP}}{b = a \times c + r \vdash b = a \times c + c + r - c}\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + r - c}\ \text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a+1) \times c + (r - c)}\ \text{MON}$$

$I_3$: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

## Invariant preservation proofs

Proofs for `Finish`

- $E_{Finish}/I_1/INV$
- $E_{Finish}/I_2/INV$
- $E_{Finish}/I_3/INV$

are trivial (Finish does not change anything)

Correctness: when Finish is executed, $I_3 \wedge G_{\mathtt{Finish}} \Rightarrow a = \left\lfloor \frac{b}{c} \right\rfloor$ (with the definition given for integer division).

## Inductive and non-inductive invariants

- We want to prove

$$A(c) \vdash I_j(E_{\text{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

- $I_j$: *inductive invariant* (base case + inductive case)

## Inductive and non-inductive invariants

- We want to prove

$$A(c) \vdash I_j(E_{\text{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

- $I_j$: *inductive invariant* (base case + inductive case)
- Invariants can be true but non-inductive if they cannot be proved from program

```
Event INIT        Event Loop
  a:  x := 1        a:  x := 2*x - 1
end               end
```

- $x \geq 0$ looks like an invariant.
  Prove it is preserved.

## Inductive and non-inductive invariants

- We want to prove

$$A(c) \vdash I_j(E_{\text{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

- $I_j$: *inductive invariant* (base case + inductive case)
- Invariants can be true but non-inductive if they cannot be proved from program

```
Event INIT        Event Loop
  a:  x := 1        a:  x := 2*x - 1
end               end
```

- $x \geq 0$ looks like an invariant.
  Prove it is preserved.
- It is not inductive (`Loop`: $x \geq 0 \vdash 2 * x - 1 \geq 0$?)

## Inductive and non-inductive invariants

- We want to prove

$$A(c) \vdash I_j(E_{\mathrm{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

- $I_j$: *inductive invariant* (base case + inductive case)
- Invariants can be true but non-inductive if they cannot be proved from program

```
Event INIT          Event Loop
  a:  x := 1          a:  x := 2*x - 1
end                 end
```

- $x \geq 0$ looks like an invariant. Prove it is preserved.
- It is not inductive (Loop: $x \geq 0 \vdash 2 * x - 1 \geq 0$?)
- $x > 0$ is inductive (Prove it!)

## Inductive and non-inductive invariants

- We want to prove

$$A(c) \vdash I_j(E_{\mathrm{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c) \vdash I_j(E_i(v,c),c)$$

- $I_j$: *inductive invariant* (base case + inductive case)
- Invariants can be true but non-inductive if they cannot be proved from program

```
Event INIT          Event Loop
  a:  x := 1          a:  x := 2*x - 1
end                 end
```

- $x \geq 0$ looks like an invariant. Prove it is preserved.
- It is not inductive (Loop: $x \geq 0 \vdash 2 * x - 1 \geq 0$?)
- $x > 0$ is inductive (Prove it!)

- $x > 0$ is stronger than $x \geq 0$ (if $A \Rightarrow B$, $A$ stronger than $B$.)
- Stronger invariants are preferred.

## Proof by contradiction: why?

$$\frac{}{\bot \vdash P}\ \text{CNTR}$$

## Proof by contradiction: why?

$$\frac{}{\bot \vdash P}\ \text{CNTR}$$

- Common sense:
  if we are in an impossible situation,
  just do not bother.

## Proof by contradiction: why?

$$\frac{}{\bot \vdash P}\ \text{CNTR}$$

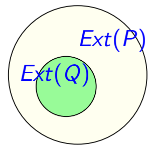
- Common sense:
  if we are in an impossible situation,
  just do not bother.
- Proof-based:
  - Let's assume $Q$ and $\neg Q$.
  - Then $\neg Q$.
  - Then $\neg Q \vee P \equiv Q \Rightarrow P$.
  - But since $Q \wedge (Q \Rightarrow P)$, then $P$.


## Proof by contradiction: why?

$$\frac{}{\bot \vdash P}\ \text{CNTR}$$

- Common sense:
  if we are in an impossible situation,
  just do not bother.
- Proof-based:
  - Let's assume $Q$ and $\neg Q$.
  - Then $\neg Q$.
  - Then $\neg Q \vee P \equiv Q \Rightarrow P$.
  - But since $Q \wedge (Q \Rightarrow P)$, then $P$.


- Model-based:
  - If $Q \Rightarrow P$, then $Q \vdash P$.
  - Extension: $Ext(P) = \{x | P(x)\}$ (id. $Q$).
  - $Q \Rightarrow P$ iff $Ext(Q) \subseteq Ext(P)$. Why???




  - If $Q \equiv R \wedge \neg R$, $Ext(Q) = \varnothing$.
  - $\varnothing \subseteq S$, for any $S$.
  - Therefore, $Ext(R \wedge \neg R) \subseteq Ext(P)$ for any $P$.
  - Thus, $R \wedge \neg R \Rightarrow P$ and then $\bot \vdash P$.


- Mordechai Ben-Ari.
  *Mathematical Logic for Computer Science, 3rd Edition*.
  Springer, 2012.
- Michael Huth and Mark Ryan.
  *Logic in Computer Science: Modelling and Reasoning About Systems*.
  Cambridge University Press, New York, NY, USA, 2004.
- Original Author Unclear.
  Lecture 10: Gentzen Systems to Refinement Logic.
  Available at https://www.cs.cornell.edu/courses/cs4860/2009sp/lec-10.pdf, last acccessed on Feb 20, 2021.
- Original Author Unclear.
  Lecture 11: Refinement Logic.
  Available at https://www.cs.cornell.edu/courses/cs4860/2009sp/lec-11.pdf, last acccessed on Feb 20, 2021.
- Original Author Unclear.
  Lecture 9: From Analytic Tableaux to Gentzen Systems.
  Available at https://www.cs.cornell.edu/courses/cs4860/2009sp/lec-09.pdf, last acccessed on Feb 20, 2021.
- Lawrence C. Paulson.
  Logic and Proof.
  Lecture notes, U. of Cambridge.