Process Algebra - Lectures

LaTeX commands

Lecture 0 - What is process algebra?

Small introduction

Website

Classic view of a computer program is a program that transforms an input form an output. Program P is a partial function: $[[P]] : States \to States$, which always terminate.

What about vending machines? Operating systems? $\implies$ **Reactive systems:** Systems that compute something by reacting to stimuli in the environment.

Goals of this coarse

- How to develop (design) a system that works
- How to analyse (verify) the design.

Concurrency theory: Active field in CS that studies formalisms for modelling/analysing systems.

Process Algebra: Branch of concurrency theory.

1. Number of **atomic processes** $\longleftarrow$ Simplest behaviour
2. Define new **composition operators** $\longleftarrow$ More complex behaviours.

Consider the following example:

$$(x := 1 || x := 2) \cdot x := x + 2 \cdot (x := x - 1 || x := x + 5)$$

- We don't know which parallel process will execute first, so there will be no unique output.
- The lowest output is 7, the largest output is 8.

Contents of the course

- Various notions of composition.
    - This is ... . Examples: sequential composition / parallel composition.
- Expressiveness
- Various notions of behavioural equivalence.
- Axiom systems + quality.
- Abstraction.

Lecture 1 - Process Algebra

Algebra of Natural Numbers

Consider set $\mathbb{N}$ together with three operations:

- $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
- $\times : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
- $succ(n) = n + 1$

Semantics

This is the Semantics of $(\mathbb{N}, +, \times, succ, 0)$:

- $n + 0 = n$
- $m + succ(n) = succ(n + m)$

These are some properties of the sets of operations.

Syntax / Logic

Signature: collection of symbols with an *arity*. Generally denoted with Σ. Signature + variables determine

Equation: Formula of the form $t = u$, where t and u are terms. **Equational theory:** Pair (Σ, E) consisting of a signature Σ and set of Σ-equations E. These equations in E are the **axioms** of the equational theory.

Let $T = (\Sigma, E)$. We write $T \vdash t = u$ if $\exists$ derivation of $t = u$ using the following rules of equational logic:

- $\frac{}{t = u} \leftarrow$ **Axiom**, if $t = u$ is an equation in E.
- $\frac{}{t = t} \leftarrow$ **Reflection**
- $\frac{t = u}{u = t} \leftarrow$ **Symmetry**
- $\frac{t = u \quad u = v}{t = v} \leftarrow$ **Transitivity**
- **Substitution rule**
- **Cont**

In other words, you can completely formalize these derivations such that we can proof equations from a bunch of axioms.

We fine an interpretation ι of the function symbols as functions t

Process

Behaviour is the execution of actions / events.

Transition-system space $(S, L, \rightarrow, \downarrow)$ consists of:

- Set S of *states*.
- Set L of *labels*.
- Transition relation: $\rightarrow \subseteq S \times L \times S$
- Set $\downarrow \subseteq S$ of *terminating* / *final* states.

Example:

- Bunch of states, like a finite automata. You can do some labels and eventually terminate.

Reachability

Reachability relation $\rightarrow^* \subseteq S \times L^* \times S$ is defined as:

1. $s \rightarrow^* s$ for all $s \in S$.
2.

Basically means whether we can reach state t from state s.

Transition system with root denotes all states that is reachable from state r.

Connection with Automata

A transition system is **regular** iff S and L are finite. Regular transition system = finite automaton

A word $\sigma \in L^*$ is a complex execution / run of a transition

Language: words that are recognized by the finite automaton. (Ways of going input $\rightarrow$ output.)

Lecture 2

Some exercise stuff

To proof:

$$T \vdash e(s^m(0), s^n(0)) = s^{m^n}(0)$$

Proof: Induction on n

If $n = 0$, then:

$$\begin{aligned} T_2 \vdash e(s^m(0), s^n(0)) &\equiv e(s^m(0), 0) \\ &= s(0) && [PA5] \\ &\equiv s^{m^n}(0) \end{aligned}$$

Let $n \geq 0$ and suppose $T_2 \vdash e(s^m(0), s^n(0)) = s^{m^n}(0)$. (Induction Hypothesis)

Then:

$$\begin{aligned} T_2 \vdash e(s^m(0), s^{n+1}(0)) &\equiv e(s^m(0), s(s^n(0)) \\ &= m(e(s^m(0), s^n(0)), s^m(0)) && [PA6] \\ &\overset{IH}{=} m(s^{m^n}(0), s^m(0)) \\ &= s^{m^n m}(0) && [\text{By } (2.2.3)] \\ &\equiv s^{m^{n+1}}(0) \end{aligned}$$

Lecture part

Binary relation R on the set of states S of a transition-system space is a **bisimulation relation** iff $\forall s, t \in S$ where $s \; R \; t$:

1. If $s \xrightarrow{a} s'$ for some $a \in L$ and $s' \in S$, then $\exists t' \in S$ such that $t \xrightarrow{a} t'$ and $s' \; R \; t'$
2.

$a \leftrightarrows b$

Lecture 3

Missed the first part

Second part

Lecture 4

Logic	Semantics
Formalising reasoning about things	Mathematical representation of 'real world' phenomenon.
Signature $\Sigma = (\mathbf{a}, \mathbf{m}, \mathbf{s}, \mathbf{0}))$	$(\mathbf{N}, +, \times, succ, 0) \leftarrow$ functions on natural numbers

Logic	Semantics
Term $\mathbf{a}(\mathbf{0}, \mathbf{s}(\mathbf{0}))$	
Axioms $\mathbf{a}(x, \mathbf{0}) = x$	

Now we can do a similar thing for processes:

Logic	Semantics
Signature (0, a._, +)	Operational semantics (Operations)
Terms (a.0+b.(c.0+d.0))	Transition system space
	Bisimilarity
	Also consists of an algebra.
	Equivalence classes

If $p \leftrightarrows q$, then $a.b \leftrightarrows a.q$

We link this via the interpretation, that links symbols and operations:

- $0 \mapsto \mathbf{0}$
- $+ \mapsto +$
- $a. \mapsto \mathbf{a}.$

Example of interpretation:

$$\begin{aligned} \iota_\alpha(a.0 + b.c.0) &= \iota_\alpha(a.0)\iota(+) + \iota(b.c.0) \\ &= \mathbf{a}.\iota(0)\iota(+) + \iota(b.c.0) \end{aligned}$$

How to proof that an axiom is complete?

Consider the axiom $x + y = y + x$.

To show that it is valid / soundness in $\mathbb{P}(MPT(A))/_{\leftrightarrows}$ we should prove that $\iota_\alpha(x + y) = \iota_\alpha(y + x)$. We can reduce it to if $[p + q]_{\leftrightarrows}$ is equal to $[q + p]_{\leftrightarrows}$, thus we should prove that $p + q$ is bisimilar to $q + p$, which is done in the previous lemma.

How to proof that an axiom is ground-complete?

An algebra $\mathbb{A}$ is *ground-complete* for the algebra ... if $\mathbb{P}(MPT(A)) \models p = q \implies MPT(A) \vdash p = q$.

After some notational stuff, we have to show that $p \leftrightarrows q \implies p = q$. Then again, how would you proof that $p = q$? We should apply the axioms.

So given that two terms are bisimilar, we show equality via the four axioms of $MPT(A)$.

Initial algebra $\mathbb{I}(\Sigma_1, E_1)$ is all the terms but with their equivalence classes, i.e. $[\mathbf{0}]_=$. What we're basically proving with ground-completeness and soundness is that this class is isomorph with the algebra, which basically means that there is a one-on-one mapping between these two classes.

Part 2

Proof idea for ground completion:

1. Assume that $q = a_1 . q_1 + \cdots + a_k . q_k$, involving A1, A2, A3 and A6.

How to proof for all closed MPT(A)-terms that $p \leftrightarrows p + (q_1 + q_2) \implies p \leftrightarrows p + q_1 \wedge p \leftrightarrows p + q_2$.

> *Proof*
>
> *We assume that* $p \leftrightarrows p + (q_1 + q_2)$
>
> *Let R be a bisimulation relation such that* $p \; R \; [p + (q_1 + q_2)]$.
>
> *Define* $R' = R \cup \{(s, s + t_1) | \exists_{t_2} \, s \; R \; [s + (t_1 + t_2)]\} \cup \{(s, s) | s \in \mathcal{C}(MPT(A)\}$
>
> *Then*

Part 3

Extension to MSP: TSP - Sequential composition.

- We've completely ignored termination behaviour. We want to add a constant for successful termination.

Consider $a.1 \cdot b.1$ first executes a and then execute action b. Consider $a.0 \cdot b.1$ first executes an a but then stops because it's in a deadlock. Consider $(a.1 + b.1) \cdot (a.1 + b.1)$, we can first execute a or b, then we subsequently execute again and we can execute a or b again. Consider $(a.1 + 1) \cdot b.1$, then we can do an a step and successfully terminate, then a b step, or it can directly do a b step.

Marc says: "🤷‍♂️🤷‍♂️🤷‍♂️🤷‍♂️🔳". Joost did not reply to this nonsense.

Lecture 5

Exercise 4.2.1

Derive A2' from $MPT(A)$.

$$\begin{aligned} MPT(A) \vdash (x + y) + z &= x + (y + z) && [A2] \\ &= (y + z) + x && [A1] \end{aligned}$$

But we should also do it the other way around, deriving $A1$ from $A2'$ and $A3$.

$$\begin{aligned} MPT(A) \vdash x + y &= (x + x) + y && [A3] \\ &= (x + y) + x && [A2'] \\ &= (y + x) + x && [A2'] \\ &= ((y + y) + x) + x && [A3] \\ &= ((y + x) + y) + x && [A2'] \\ &= (y + x) + (y + x) && [A2'] \\ &= y + x \end{aligned}$$

We can also simplify the proof by applying the axioms the other direction.

Basic Sequential Processes

The theory of $BSP(A)$ is the same as $MSP(A)$ but with th added constant 1 for **successful termination**.

We can finally use our termination predicate $\downarrow$, yay! :D

SOS meta-theory - Determining that bisimilarity is a congruence to these TSP.

- A collection of *operational rules* is in **path format** iff every rule in R holds:
  - The target of the premise (transitions above the line) only contains a single variable.
  - Source of the conclusion (transition below the line) should either be:
    - Single variables
    - A function symbol applied to variables.
  - The variables in the source of the conclusion + target of premise and see whether one of these variables occurs more than once in either of these positions.

If the operational rules are all in the path format, then *bisimilarity* is a congruence for that process calculus.

For all closed $BSP(A)$-terms, $BSP(A) \vdash p = 1 \iff p \leftrightarrows q$.

Sequential Processes

We extend $BSP(A)$ with a binary operator $\cdot$ for sequential composition:

- Process $p \cdot q$ executes p first and, upon successful termination of p, then executes q.
- $(a.1 + b.1) \cdot (a.1 + b.1)$ first executes either an a or a b and again either an a or a b.

We have some extra operational semantics.

$$a.1 \xrightarrow{a} 1 \quad a.1 \cdot b.1 \xrightarrow{a} 1 \cdot b.1$$

Now we still have $1 \cdot b.1$ over. We can use this rule to indicate that:

$$1 \cdot b.1 \xrightarrow{b} 1$$

Remember that $\downarrow$ is the least set satisfying the rules above. If term $p \in \downarrow$. So apparently if this is true, it has to be so according to the roles, as $\downarrow$ is the *least* set that adheres to the rules, this means that there *has to be* a proof that p has to terminate.

Why can $a.1$ not be terminating? We don't have any possibility in our operational semantics that $a.1 \downarrow$ does not hold. By this reasoning we can conclude that $a.1 \not\downarrow$.

Then my questions are:

1. Can we express in $TSP(A)$ more behaviour than in $BSP(A)$?
2. Does the operational semantics for $TSP(A)$ change the behaviour of $BSP(A)$-terms?
   - In other words, is it *operationally conservative extension*? Do these three more rules influence behaviour of the old terms, such as $a.1 + b.0$?
     1. No, since the sources of the conclusions of all operational rules not stemming from $BSP(A)$ terms are not $BSP(A)$-terms.
     2. No, the operational rules stemming from $BSP(A)$ are *source-dependent*, i.e. we cannot introduce new arbitrary terms, such as $TSP(A)$ terms.
     3. Therefore, according to Theorem 3.2.19, $TSP(A)$ is an *operationally conservative extension*, i.e. $p \leftrightarrows q$ in $TSP(A) \iff p \leftrightarrows q$ in $BSP(A)$.

We want to create a strategy to eliminate our new behaviour by removing every term with sequential composition to without sequential composition.

Elimination

For every closed *BSP(A)*-term p it holds that for every closed $BSP(A)$-term there exists a closed $BSP(A)$-term r such that $p \cdot q = r$.

> *If* $p \equiv 0$, *then by A7,* $p \cdot q = 0$, *which is a* $BSP(A)$*-term.*
>
> *If* $p \equiv 1$, *then by A9,* $p \cdot q = q$, *which is a* $BSP(A)$*-term.*
>
> *Suppose that* $p \equiv a.p'$ *and sassume that for every BSP(A)-term there exists a BSP(A)-term r' such that* $p' \cdot q = r$ *(IH)*
>
> *Then, by A10 and IH,* $p \cdot q = a.(p' \cdot q) = a.r'$, *which is a* $BSP(A)$*-term.*
>
> *Suppose that* $p \equiv p_1 + p_2$, *and suppose ...*
>
> *Now for every arbitrary* $TSP(A)$*-term* p *there exists a closed* $BSP(A)$*-term q such that* $p = q$.

Lecture 6 - Start of recursion

Exercise 6.2.6

$$\begin{aligned} x + x &= 1 \cdot x + 1 \cdot x && A9 \\ &= (1 + 1) \cdot x && A4 \\ &= 1 \cdot x && 1 + 1 = 1 \\ &= x && A9 \end{aligned}$$

Model: Axioms are valid for the algebra. Check validity for algebra? Do interpretation, and in case of a bisimilarity model check if the terms are bisimilar. Need soundness for this.

Otherwise just use the axioms if you want to stay within the theory.

Another tip: derive all functionality of transitions via the TDS, then we can draw the transition system.

Recursion

Suppose we have a transition system with a loop. Can we express this behaviour in $BSP(A)$? No, as we want to have a finite transition system.

General method for expressing transition systems:

1. Label states with *process names*, also known as **recursion variables**.
2. Associate behaviour of every recursion variable by means of an equation.
    - For every recursion variable, specify its transition and termination behaviour.
    - Example: $X = a.Y + c.Z$, $Y = b.X + 1$, and $Z = 0$.

Formally: Let Σ be a signature and let V_R be a set of *recursion variables*. A **recursive equation** over Σ and V_R is an equation of the form of:

$$X = t$$

With $X \in V_R$ and t a term over Σ and V_R. Recursive equation $X = t$ *defines* X.

Recursive specification: Set of *recursive equations* over Σ and V_R such that there is *exactly* one equation defining X for each $X \in V_R$.

Simplified representation of operational semantics

So, how do we specify an operation over these new equations?

Let E be a recursive specification over Σ and V_R including a defining equation $\forall X \in V_R : X = T_X$:

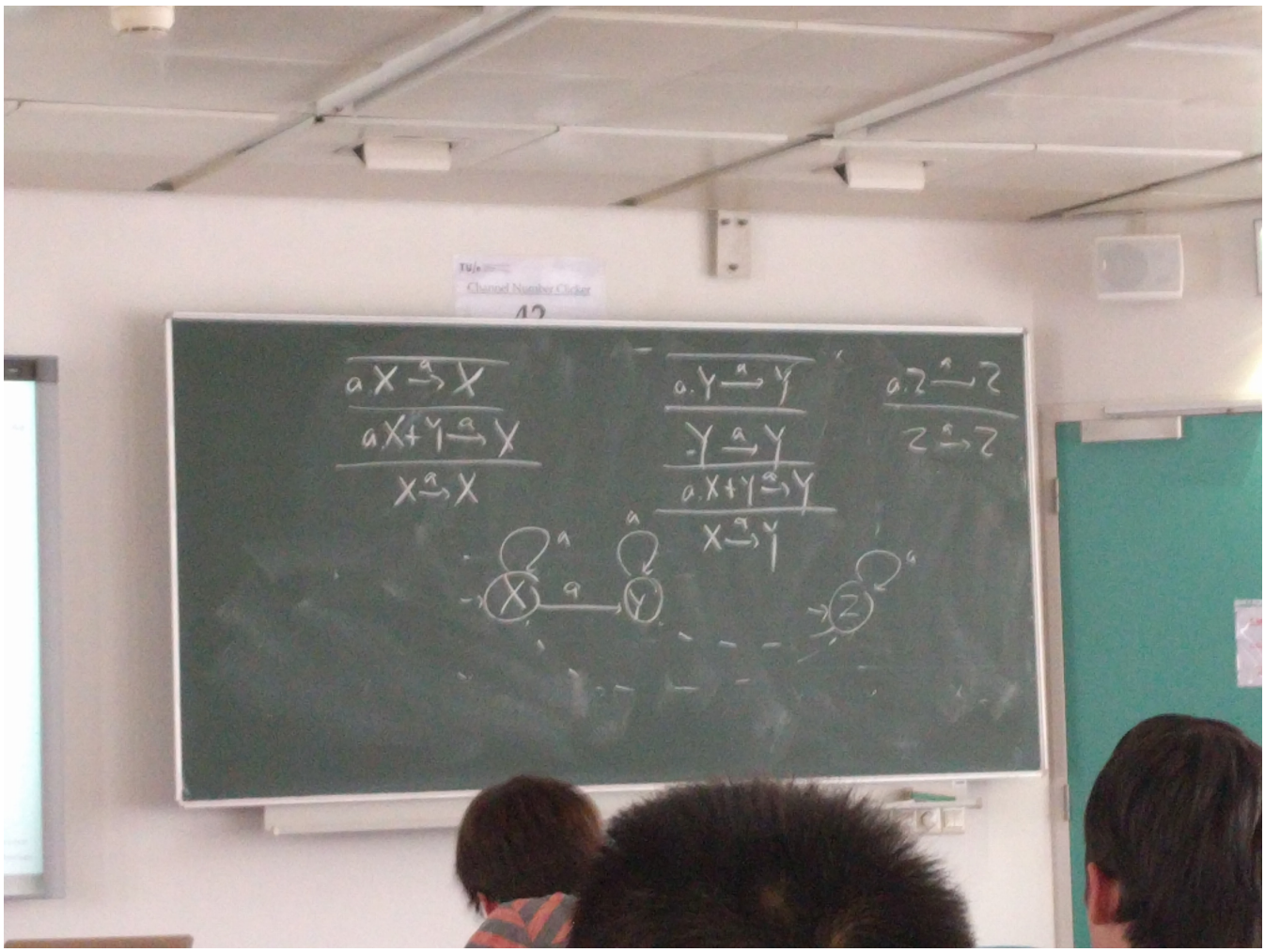
$$\frac{t_X \xrightarrow{a} t_X}{X \xrightarrow{a} t_X} \qquad \frac{t_X \downarrow}{X \downarrow}$$

Recursion - Bisimilarity

Prove that $X \leftrightarrows Z$, given that:

$$X = a.X + Y$$
$$Y = a.Y$$
$$Z = a.Z$$

Proof, see this picture:



In the end, this is our relation: $R = \{(X, Z), (Y, Z)\}$

Term model (Simplified)

Let E be a recursive specification over $BSP(A)$ and V_R.

Term algebra for $BSP(A) + E$ is the algebra:

$$\mathbb{P}(BSP(A) + E) = (\mathcal{C}(BSP(A) + E), +, (a.)_{a \in A}, 0, 1, (X)_{X \in V_R})$$

Theorem - Soundness of recursion

The equational theory BSP(A)+E is a sound axiomatisation of $\mathbb{P}(BSP(A) + E)_{/\leftrightarrows}$.

Furthermore, bisimilarity is a congruence on $\mathbb{P}(BSP(A) + E)$ as it is in path format.

Equivalence of recursion variables. Consider the recursive specification:

$$\left\{ \begin{array}{l} X = a.X, \\ Y = a.a.Y \end{array} \right\}$$

There are bisimilar as we can create a relation R. However, we cannot proof this as we cannot get rid of the X or Y from the equation of the X and Y respectively.

Conclusion, we need additional methods to reason about the equivalence of recursion variables. (There won't be a full-fledged ground-complete axiomisation of $\mathbb{P}(BSP(A) + E)_{\setminus\leftrightarrows}$, but we'll get pretty close.)

Lecture 7 - Recursion

Solutions that are interpretations

Let E be a recursive specification over signature Σ and set of variables V_R. Furthermore, let $\mathbb{A}$ be a Σ-algebra and ι the associated interpretation.

Solution: Let κ be an extension of ι with interpretations of the recursive variables in V_R such that $\mathbb{A}, \kappa \models X = t_X$ for every equation in E. Then we call $\kappa(X)$ a solution of X in E.

> *Reminder, ι is an interpretation of the terms of the theory:*
>
> $+ \mapsto +\ a. \mapsto \mathbf{a.}$
>
> *This changes what these variables do, as the left sides follow the axioms of our theory, whereas the right part works with the equivalence classes:* $\mathbf{a.}[p]_{/\leftrightarrows} = [a.p]_{/\leftrightarrows}$.
>
> *Then κ would be an extension of ι, adding more interpretations to the already existing interpretations, where an example would be:*
>
> $\kappa : X \mapsto [a.1]_{/\leftrightarrows}$
>
> *So how would we prove this?*
>
> *We know that* $\mathbb{P}(BSP(A))_{/\leftrightarrows}, \kappa \models X = a.1$. *Interpreting the lefthandside gives:* $\kappa(X) = [a.1]_{/\leftrightarrows}$
> *Interpreting the righthandside results in:* $\kappa(a.1) = \iota(a.1) = [a.1]_{/\leftrightarrows}$

What about recursion? Consider the recursive specification $E_2 = \{X = a.X\}$. Then $X \mapsto [a.X]_{/\leftrightarrows}$ is an **invalid** mapping, as we'll have troubles with the interpretation of $\kappa(X) = \mathbf{a}.\kappa(X)$. In fact, there is no solution in $\mathbb{P}(BSP(A))_{/\leftrightarrows}$.

There is, however, a solution in $\mathbb{P}(BSP(A) + E_2)_{/\leftrightarrows}$, namely the interpretation of $\kappa : X \mapsto [X]_{/\leftrightarrows}$. Then we get a valid solution for the interpretation:

$$\begin{aligned}[X]_{/\leftrightarrows} &= \\ \kappa(X) &\stackrel{?}{=} \kappa(a.X) \\ &= \mathbf{a}.\kappa(X) = \mathbf{a}.[X]_{/\leftrightarrows} = [a.X]_{/\leftrightarrows}\end{aligned}$$

Now we just have to show that X and $a.X$ are equivalence, as then also their bisimilarity classes modulo bisimilarity are equivalent. We just have to create a bisimulation relation R and show that these are equivalent.

Exercise: shows that $\kappa : X \mapsto [a.X]_{/\leftrightarrows}$ is also a valid interpretation.

What about $E_3 = \{X = X\}$? Well, we can just put any interpretation for ι, as the left and right side will always be interpreted the same. Thus, this recursive specification has *many solutions*.

In fact, later on we will check whether an interpretation is unique to the model.

Equivalence of recursive values

Consider this recursive specification again:

$$\left\{\begin{array}{l} X = a.X, \\ Y = a.a.Y \end{array}\right\}$$

We can argue that every solution of X is a solution for Y too.

> *Proof*
>
> *This is what we want to proof:* $\mathbb{A}, \kappa \models X = a.X \implies \mathbb{A}, k \models Y = a.a.Y$
>
> *From the lefthandside:* $\kappa(X) = \kappa(a.X) = \mathbf{a}.\kappa(X)$. *Thus,* $\kappa(X) = \mathbf{a}.\mathbf{a}.\kappa(X)$. *[By transitivity] From the righthandside:* $\kappa(Y) = \kappa(a.a.Y) = \mathbf{a}.\mathbf{a}.\kappa(Y)$
>
> *Thus, if we define* $\kappa(Y) := \kappa(X)$, *then this* $\kappa(X)$ *is a solution for* Y.

Fun fact: this holds for *any* algebra.

However, the other way around will not work.

> *Proof*
>
> *We will give a model of the theory in which we have a solution for* Y *but not for* X. *In bisimilarity it holds, so we should think of a different kind of model.*
>
> *Let* $\mathbf{A} = \{0, 1\}$. *Let* 0 *and* 1 *be constants of this model. Define* $a.$ *as follows:* $a.1 = 0$ *and* $a.0 = 1$. *Define* $+$ *as follows:* $0 + 0 = 0$, *whereas* $1 + 0 = 1$, $0 + 1 = 1$, *and* $1 + 1 = 1$. *Note that all axioms in* $BSP(A)$ *still holds.*

> *Now, let* $\kappa(Y) = 1$ *by definition. Then the interpretation of* $\kappa(Y) = \kappa(a.a.Y)$ *results in the same answer with the interpretation. However, it does not hold for* $\kappa(X) = \kappa(a.X)$*, since the l.h.s. equals 1 and the r.h.s. equals 0.*
>
> *Furthermore, for* $\kappa(Y) = 0$ *it doesn't work as well. Therefore, we do not have a solution for* X*.*

Still, this result is not really desired, as we want to denote that these equations' interpretation are the same if we consider them as processes in modulo bisimilarity.

Suppose we have a more general method that excludes some models, such that X an Y have both a *unique solution*, then we can conclude that X and Y denote the same solution. Thus, this means that these denote the same process, which is what we want.

Guarded

Our first goal is to exclude behaviour that are trivial and result in many solutions in any nontrivial model, such as $E = E$. This is called **guardedness**:

An occurrence of a recursion variable X in a closed term s is **guarded** if it occurs in the scope of a prefix.

- $a.X \leftarrow X$ is guarded
- $Y + b.X \leftarrow X$ is guarded, Y is not guarded.

A *term* s is **completely guarded** if all its terms all guarded. A *recursive specification* is **completely guarded** if all the terms of its equations are guarded. A *recursive specification* is **guarded** if there exists a *completely guarded* recursive specification F with $V_R(E) = V_R(F)$ and $BSP(A) + E \vdash X = t$ for all $X = t \in F$.

The last requirement is for some recursive specification that in itself is not completely guarded, but has the same solution as a completely guarded specification.

Exercise 5.5.2: 1. is guarded, 2. is not guarded.

New recursive specification

All this results in a new theory called **Recursive Specification Principle (RSP)**:

Σ-algebra $\mathbb{A}$ satisfies RSP if every guarded recursive specification E and some set V_R of variables has *at most* one solution.

Lecture 8 - Recursion II

Recursive Specification Principle

An example

Consider the following recursive specification:

$$\begin{aligned} X &= a.X + b.X \\ Y &= a.Y + b.Z \\ Z &= a.Z + b.Y \end{aligned}$$

How do we prove *using RSP* that $X = Y$?

> ***Proof***

Consider the following terms, which represents a solution to the equation:

$$t_1 \equiv X$$
$$t_2 \equiv Y$$
$$t_3 \equiv Z$$

To show that these terms are a solution, we should show that the equations should still hold:

$$t_1 \stackrel{?}{=} a.t_1 + b.t_1$$
$$t_2 \stackrel{?}{=} a.t_2 + b.t_3$$
$$t_3 \stackrel{?}{=} a.t_3 + b.t_2$$

We can use the equations of the solution to substitute t_1, t_2, and t_3 with X, Y, and Z respectively. This results in the following equations:

$$t_1 \equiv X = a.X + b.X = a.t_1 + b.t_1$$
$$t_2 \equiv Y = a.Y + b.Z = a.t_2 + b.t_3$$
$$t_3 \equiv Z = a.Z + b.Y = a.t_3 + b.t_2$$

Therefore, this is a valid solution.

Now, consider the following solutions:

$$t_1 \equiv X$$
$$t_2 \equiv Y$$
$$t_3 \equiv Z$$

Now we should check that the solution still holds.

$$u_1 \stackrel{?}{=} a.u_1 + b.u_1$$
$$u_2 \stackrel{?}{=} a.u_2 + b.u_3$$
$$u_3 \stackrel{?}{=} a.u_3 + b.u_2$$

We know that $u_2 \equiv X$. By the first equation, we know that:

$$\begin{aligned} u_2 \equiv X &= a.X + b.X \\ &= a.u_2 + b.u_3 \end{aligned}$$

Satisfying the second equation. We can proof something similar.

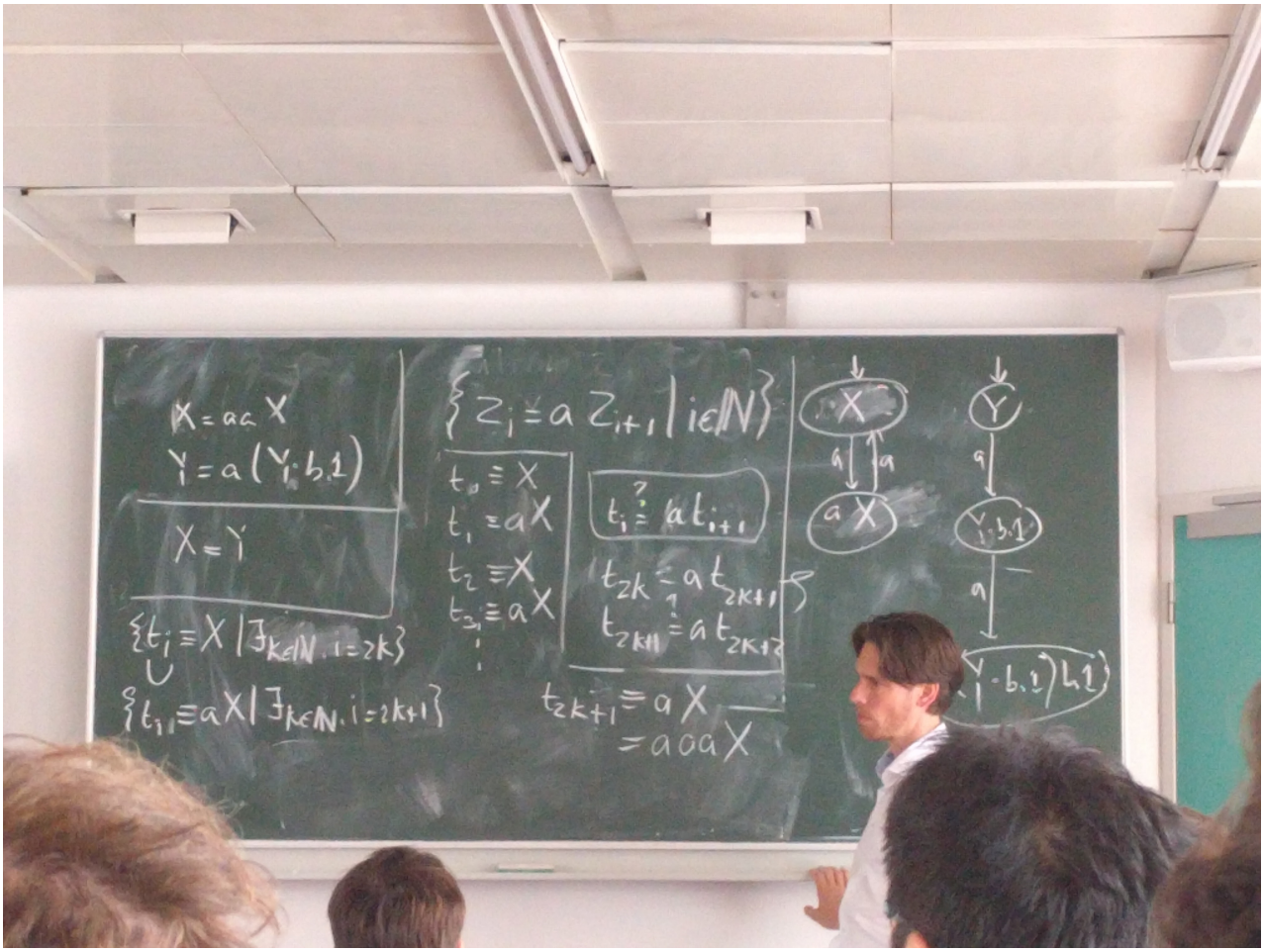
By RSP we know that there exists at most 1 solution for the recursive specification. Therefore:
$t_1, t_2, t_3 = u_1, u_2, u_3$ and thus $t_1 = u_1$, $t_2 = u_2$, and $t_3 = u_3$. Because we know that $t_2 = u_2$, we know that $Y = X$ which is what we wanted to show.

Another example

Another example, consider the following two recursive specifications:

$$E = \{X = a.a.X\}$$
$$F = \{Y = a.(Y \cdot b.1)\}$$

Can we proof that $\mu X.\, E = \mu Y.\, F$? Yes, we can!

> ***Proof***
>
> *We proof this by defining an infinitely guarded specification $\{Z_i = aZ_{i+1} \mid i \in \mathbb{N}\}$. We will find two different solutions that hold for this specification; one containing X and the other one containing Y. Finally, as RSP can only have one solution at most, we will conclude that $X = Y$.*
>
> 
>
>
> *(We could also have defined the terms as follows: $t_0 = X$ and $t_{i+1} = a.\, t_i$.*
>
> *Furthermore, we define the terms u as: $u_0 = Y$ and $u_{i+1} = u_i + b.1$. We proof that this ($u_i \stackrel{?}{=} a.\, u_{i+1}$ is a valid solution by solution by induction:*
>
> - ***Base case:*** $u_0 \equiv Y = a(Y \cdot b.1) \equiv a.\, u_1$
> - ***Inductive case:*** *Suppose* $u_i = a.\, u_{i+1}$ *(Induction Hypothesis).*
>
> *$u_{i+1} =*
>
> *By RSP, we know that there exists at most one solution. Thus these solutions must be the same, thus $t_0 = u_0, t_1 = u_1, \ldots$. Since $t_0 = u_0$, we know that $X \equiv t_0 = u_0 \equiv Y$.*

Term model

New recursive specification

How do we talk about *multiple recursive specifications* at once, and how do we reason about this? Well, for this purpose, we are going to extend $BSP(A)$ such that we add all different recursive specifications possible, such as E and F. Because then we can talk about equality with different recursive specifications.

For this purpose, let us define *Rec* as the collection of **all** recursive specifications. Furthermore, let us call our new model as $BSP_{\mathbf{rec}}(A)$. Since we have multiple recursive specifications, it might be that constants are defined in different ways, such as $E = \{X = aaX\}$ and $F = \{X = bX\}$. So which X are we talking about? For this purpose, we will add the following notation:

$$\mu X.\, E$$

...

...

Recursive definition principle

Consider Σ-algebra $\mathbb{A}$. We state that $\mathbb{A}$ satisfies **Recursive Definition Principle** if every recursive specification + some variable set V_R has *at least* one solution.

Furthermore, we can show that $\mathbb{P}(BSP_{\mathbf{rec}}(A))_{/\leftrightarrows}$ satisfies RSP. This is denoted as:

$$\mathbb{P}(BSP_{\mathbf{rec}}(A))_{/\leftrightarrows} \models RSP$$

The proof of this, however, will be postponed.

Infinite processes

Regular behaviour: An equivalence class of transition systems modulo $\leftrightarrows$ containing at least one regular transition system. So if we can proof that at least one transition system is a regular transition system (i.e. if you draw it, it's a finite drawing) in this class of modulo $\leftrightarrows$, then the behaviour of all transition systems in this class of modulo $\leftrightarrows$ will be regular. (Example: see this example, where X is regular, Y is not regular but its behaviour is still regular as $X = Y$.)

How to show that the behaviour of a transition system is not bisimilar to a regular transition system? Well, show that there are infinitely many states that are not bisimilar to eachother. (See Lemma in lecture page 3/18.)

Lecture 9 - Definability and Expressiveness

Stack

Suppose we have a LIFO queue, aka a Stack. Let $D = \{d_1, \ldots, d_n\}$ be a finite set of data, and D^* is the set of all finite sequences of elements of D.

Initially, the stack can push any arbitrary elements of d or it can terminate. Finally, let d be the last pushed on the stack:

$$S_{d\sigma} = pop(d).\, S\sigma + \sum_{e \in D} push(e).\, S_{ed\sigma}$$

We can proof that the behaviour of the stack is not regular. First, here is a sketch of the behaviour of the stack.

How to proof not regular?

There is a uniquely way of following a pop sequence going from one state to the empty state. Therefore, any two states given in the transition system, called S_σ and $S_{\sigma'}$, then it follows that there is a unique sequence of $pop(d)$-transitions to S_ϵ, the only state with the termination option. Therefore S_σ and $S_{\sigma'}$ are bisimilar only if $\sigma = \sigma'$.

Since there are infinitely many reachable states σ, there thus exists an infinite amount of bisimilar states and therefore is not regular. □

Another proof is proof by contradiction.

Definable

A process is **definable** iff it is the unique solution of a *guarded* recursive specification over the signature of T.

A process is **finitely definable** iff it is the unique solution of a *finite guarded specification* over the signature of T. This means that you can define it with finitely many equations.

Theorem - Finitely definable and regularity

A behaviour is *finitely definable* in `BSP(A)` if and only if it is *regular*.

Sketch of proof: for the regular $\Rightarrow$ finitely definable, we can just name all processes with recursion variable and by construction these will be guarded and finite, since regular denotes finite.

Intermezzo

Suppose $s_i \leftrightarrows s_j$ and $i < j$ and consider the *least* i with this property. This means that there exists a bisimulation relation R such that $s_i \ R \ s_j$. There are two possibilities for this: either s_i is s_1 or $i > 1$. If $i = 1$ then s_i does not have a b step, whereas s_j has a b-step to s_{j-1}., contradicting that $s_i \ R \ s_j$. If $i > 1$, then $s_i \xrightarrow{b} s_{i-1}$. Hence, since, $s_i \ R \ s_j$, then there should exist s' such that $s_j \to s'$ and $s_{i-1} \ R \ s'$. Note that $s' = s_{j-1}$. Therefore $s_{i-1} \ R \ s_{j-1}$. However, we chose i to be the least *i* with this property, but since $i - 1 < i$, this results in a contradiction.

Back to stack again

This means that the stack is not *finitely definable* in $BSP(A)$. However, in $TSP(A)$ we can write the stack down in a single line using sequential decomposition.

Lecture 10 - Expressiveness

Expressible up to bisimilarity

A transition system is **expressible up to bisimilarity** in a process theory if it is bisimilar to the transition system associated with a closed T(A)-term. Regular transition system is expressible up to bisimilarity in $RSP_{\mathtt{rec}}(A)$.

- Why this? We want to state something about some unguarded recursive specifications.

> *Example*
>
> *Consider the following recursive specification:*
>
> $$\{X_n = a^n.1 + X_{n+1} \mid n \in \mathbb{N}\}$$
>
> *Then we have the following transitions:*
>
> - $X_0 = 1 + X_1 \qquad X_1 = a.1 + X_2 \qquad X_2 = a.a.1 + X_3$.
>
> *So when we're in the process X_0, we can actually execute transitions when we execute the process X_1, even though it isn't guarded. Similarly, for X_2 and X_3.*
>
> *Notice that when we draw this process, it's not regular. Therefore we cannot finitely define it in BSP. However, we can specify it in BSP with an infinite recursive specification with unguarded terms!*

> *Assignment 6.6.7*
>
> *Find two non-bisimilar transition systems that are both solutions of the unguarded recursive equation $X = X \cdot a.1 + X \cdot b.1$.*
>
> *Consider the solution $X \mapsto 1$. Then it should hold that the transition system 1 should be bisimilar to $1 \cdot a.1 + 1 \cdot b.1$, quod non.*
>
> *Consider the solution $X \mapsto 0$, Then it should hold that 0 and $0 \cdot a.1 + 0 \cdot b.1$, which holds. Similarly, consider the solution $X \mapsto a.b.0$. Then $a.b.0$ is also bisimilar to $(a.b.0) \cdot a.1 + (a.b.0) \cdot b.1$*
>
> *Finally, another transition system would be $\mu Y.\{Y = a.Y\}$, since we always have to do a steps and can never reach $a.1$ or $b.1$.*
>
> *However, notice that there **never** is a Transition System that we can fully compute or draw. The "solution" is just: $\rightarrow \circ$.*

So... how do we give a recursive specification E over $BSP(A)$ of an infinite transition system? We're going to define recursion variables as follows, for each state to each state:

$$\begin{aligned} X_{0,0} &= X_{0,1} \\ X_{0,1} &= a.X_{1.0} + X_{0,2} \\ X_{0,2} &= a.X_{2,0} + X_{0,3} \\ &\cdots \end{aligned}$$

There are the infinite sequence of equations of state s_0. How about the rest of the states?

$$\begin{aligned} X_{1,0} &= 1 \\ X_{2,0} &= X_{2,1} \qquad X_{2,1} = b.X_{1,0} \\ X_{3,0} &= X_{3,1} \qquad X_{3,1} = b.X_{2,0} \end{aligned}$$

In general, the components are ordered like this: $X_{\text{state,trans}}$. In fact, we can do this for every *countable* transition system.

Theorem

Every countable *transition system* is expressible up to bisimilarity in $BSP_{\text{rec}}(A)$.

Notice the difference between this and the definable theorem.

Definability and expressiveness in TSP(A)

We know that TSP(A) is as expressive as BSP(A), in both theories precisely all countable transition systems can be specified.

However, in TSP(A) it's possible to finitely define infinite-state behaviours (which are not regular).

> *Exercise 6.6.8*
>
> *Consider the recursive specification* $\{X = X \cdot a.1 + a.1\}$.
>
> *One solution would be* $\{X_n = a^{n+1} + X_{n+1} \mid i \in \mathbb{N}\}$*, but this is the same solution, we want a different solution.*
>
> *How about the following recursive specification, is this a solution:*
>
> $$F = \{Y = a.Z + U, \qquad Z = a.Z, \qquad U = U \cdot a.1 + a.1\}$$
>
> *How do we check if* $\mu Y.F$ *is a solution? We draw the transition system of* $\mu Y.F$ *and we draw the transition system of it as a solution of* X*, namely* $\mu Y.F \cdot a.1 + a.1$*. If they are bisimilar.*
>
> *Therefore, such an unguarded equation has more than 1 solution.*

Note: solutions are about *validity* and interpretation and confirm they' re the same in the model, i.e. bisimilarity, not that two equations are axiomatic the same.

Basic Communicating Processes

Why did we again do the processes? Well, it's easy in sequential algorithm to show that a process is correct, but for a parallel program it's much harder.

Goal: Extend BSP(A) with a binary operator $\|$ for **parallel composition**. Processes $p \| q$ executes p and q in parallel.

Consider the following process terms:

- $a.b.1 \| c.1$
  - We can either execute an a step or a c step.

- $a.b.1 \parallel c.1 \xrightarrow{a} b.1 \parallel c.1 \xrightarrow{c} (b.1 \parallel 1)\downarrow$

Assumption: In our theory, we can execute either the left side or the right side, but not both sides at the same time. Therefore, we assume that each action is *atomic*.

Operational rules

We can do either a step of the left component or the step of the right component.

Synchronisation and Communication

Extra idea: we're going to extend the semantics of parallel composition with synchronisation. This happens when the executions of two processes running in parallel are *interleaved*.

This is when components may **synchronise** on certain actions using a *communication function*.
Communication Function: partial function $\gamma : A \times A \rightharpoonup A$ satisfying two conditions:

- $\gamma(a,b) = \gamma(b,a) \qquad \forall a,b \in A$ (Commutativity)
- $\gamma(\gamma(a,b),c) = \gamma(a,\gamma(b,c)) \qquad \forall a,b \in A$ (Associativity)

We can, e.g. assume that there are actions $c?k$, $c!k$ and $c!?k$, and that $\gamma(c?k, c!k) = \gamma(c!k, c?k) = c!?k$.

This results in a new operational rule:

$$\frac{x \xrightarrow{a} x' \qquad y \xrightarrow{b} y' \qquad \gamma(a,b) = c}{x \parallel y \xrightarrow{c} x' \parallel y'}$$

Encapsulation

However, what if we *only* want to perform this communication action and not each action separate. That's where we have the **encapsulation**, $\partial_H(x)$, causing that all actions in H are not allowed to execute. Very useful if you have an action a, an action b and $\gamma(a,b) = c$, but you only want to let them execute at the same time. Then we can state that $H = \{a,b\}$.

Example

> *Consider processes*
>
> $A = runA.\,give.1$ *and* $B = take.\,runB.1$.
>
> *We want to say that there is communication between the* $give$ *and* $take$ *actions.*
> $\gamma(give, take) = \gamma(take, give) = pass$ *(and undefined for the rest). And let* $H = \{give, take\}$.
>
> *We want to compute* $A \parallel B$ *and* $\partial_H(A \parallel B)$.

Lecture 11 - Parallelism

Basic Communication Process - BCP

Summary

So now we have the new *parallel composition* operator $\parallel$. Furthermore, we have introduced a communication function γ where only if $\gamma(a,b) = c$ then we can execute the command.

Furthermore, we can enforce that a send or receive at the same time with the *encapsulation*, called $\delta_H(x)$, where H typically consists of the sends and receives. It filters out transitions you don't want to happen individually.

Axioms

Theorem

There does not exist a direct finite ground-complete axiomatisation of the algebra associated with $BSP(A)$ extended with $\|$.

The problem with proving this is that there does not exist a distribution over $+$, so we cannot prove it by reducing it to e.g. $BSP(A)$.

For the ground-complete axiomatisation, **auxiliary operators** have been added:

- **Left-merge:** $p \underline{\|} q$ executes p and q in parallel, but the first execution step must come to p.
- **Communication-merge:** $p \mid q$ executes p and q in parallel, but the first execution stup must be a *sync step* from p and q.

We can then add fairly straightforward axioms for the *left-merge*, as it only adds one new rule, as there is no termination behaviour here.

Similarly, we can have axioms for the *communication-merge*.

Finally, we axiomitize encapsulation. Some axioms in particular are:

- $\delta_H(a.x) = 0$ if $a \in H$
- $\delta_H(a.x) = a.x$ if $a \notin H$.

There are also other axioms which are there for convenience, containing $|, \|, +, (a._-), \delta_H$.

Results

Furthermore, we know the following characteristics from BCP:

1. Bisimilarity is a **congruence** on its algebra.
2. $BCP(A, \gamma)$ is **sound** for the algebra modulo bisimilarity.
3. **Elimination:** for every closed $BSP(A, \gamma)$-term p there exists a closed $BCP(A)$-term q such that $BCP(A, \gamma) \vdash p = q$.
4. $BCP(A, \gamma)$ is **ground-complete** for the algebra modulo bisimilarity.

Finally, we know the following theorem of an n-fold parallel composition.

No communication

If we have no communication at all, $\gamma = \emptyset$, then we may add the following **Free Merge Axiom**:

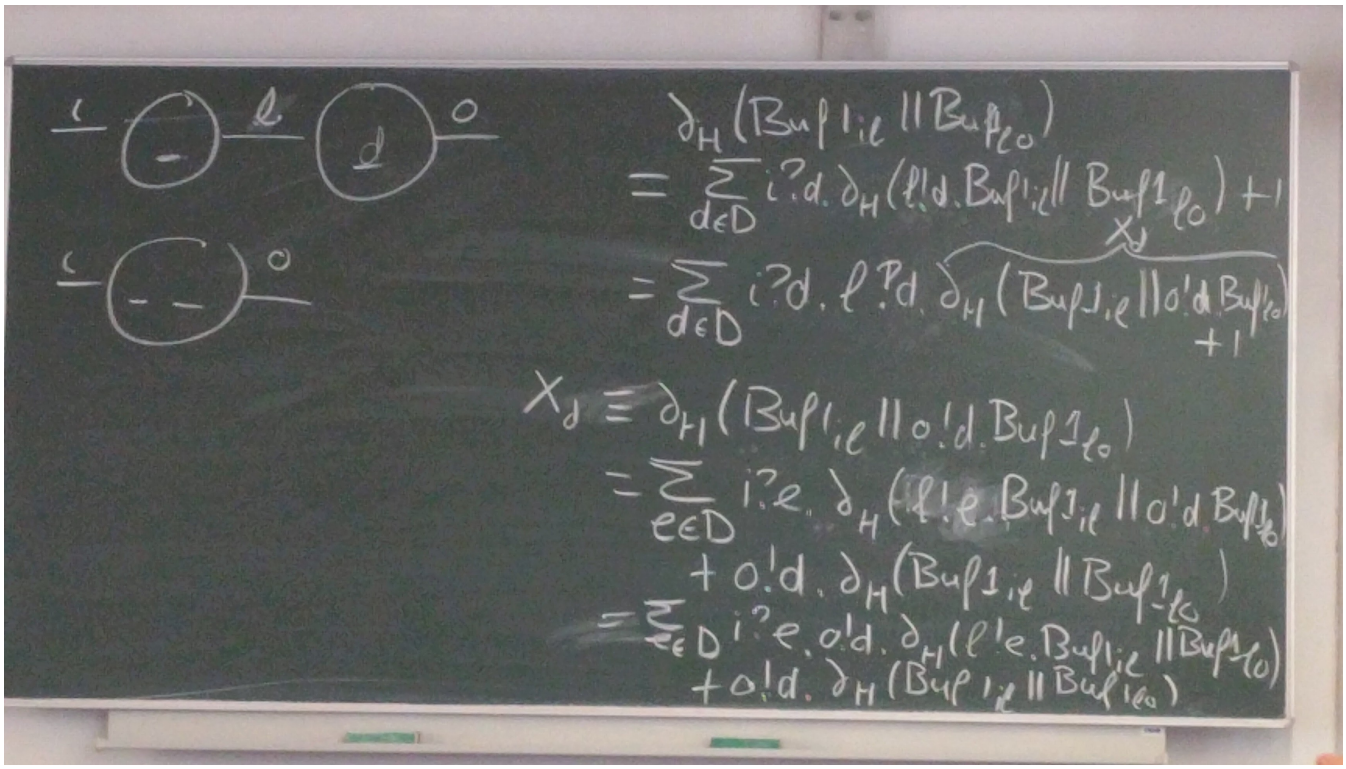
$$x \mid y + 1 = 1$$

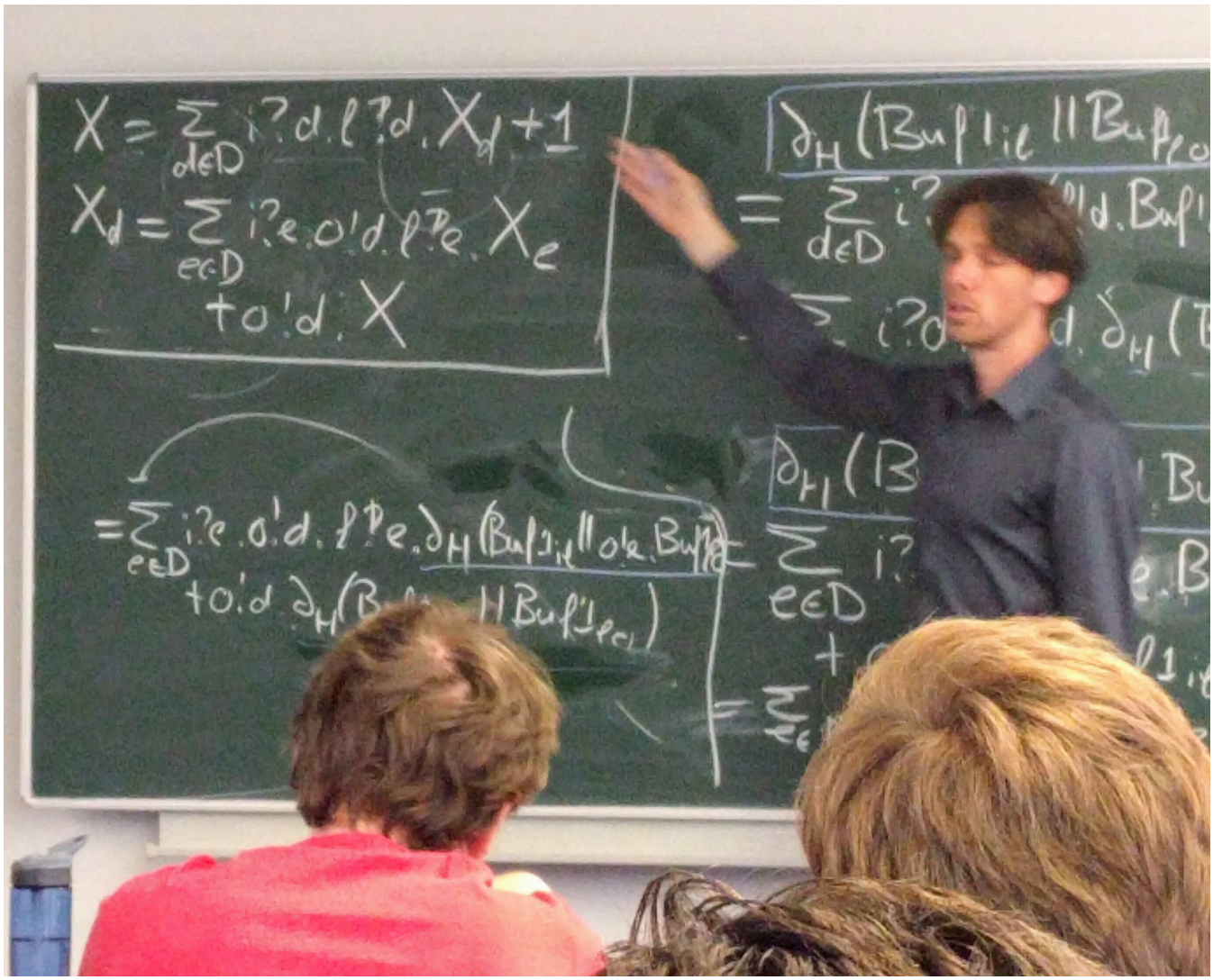
Buffer example

We can create a **one-place buffer** that can accept a single data element $d \in D$ and then has to output it before accepting another one.

Similarly, we can create a **two-place buffer** where for each data element we add an extra equation. But we can also build a two-place buffer by placing two one-place buffer in parallel.

A kind of intuitive proof that these are the same is as follows:





We know have a specification of the process of our one-bit buffer. When we compare this to the two-bit specification, we cannot conclude that they're the same or bisimilar, as there are intermediate communication actions. Can we *hide* the internal communication and then show that they are identical? Can we *abstract* from the communication?

Skip operator

We'll introduce an extra operator **skip** $\epsilon_I(x)$, where $I \subseteq A$ is a subset of axioms. These also have some corresponding rules. The idea is:

- Epsilon to successful termination is 1, $\epsilon_I(0) = 0$.
- Epsilon skips the action - $\epsilon_I(a.x) = \epsilon_I(x)$ if $a \in I$.
- Epsilon doesn't skips the action - $\epsilon_I(a.x) = a.\epsilon_I(x)$ if $a \notin I$.

With this help we can proof equality of $\epsilon_I(\delta_H(Buf_{il} \parallel Buf_{lo})) = Buf_2$.

Abstraction

A new way of abstracting

With the ϵ_I operator we can *abstract* from internal activities. Is it always suitable? Nope. Consider:

$$\epsilon_{\{b\}}(a.1 + b.0) = a.1 + 0 = a.1$$

This results in removing the deadlock that was inside the process. You cannot see the $b.\_$ happen, but it can happen.

That's why we'll do abstraction differently with $\tau_I(x)$. This operator follows slightly different rules, as when we abstract we change the action $a \in I$ to the action τ.

Furthermore, we need to do something extras, as we will have some troubles with equality when we have a τ behaviour: $\tau.\tau.a \neq \tau.a$ or what about $\tau.b + a \stackrel{?}{=} b + a$? This is done by the so-called **Branching bisimilarity**.

We denote the reflexive-transitive closure of $\xrightarrow{\tau}$, also known as $\xrightarrow{\tau}{}^*$ by $\twoheadrightarrow$. If $s \xrightarrow{a} t$ || ($a = \tau$ && $s = t$), then we write $s \xrightarrow{(a)} t$.

Here is the definition of branching bisimulation $\leftrightarroweq_b$:

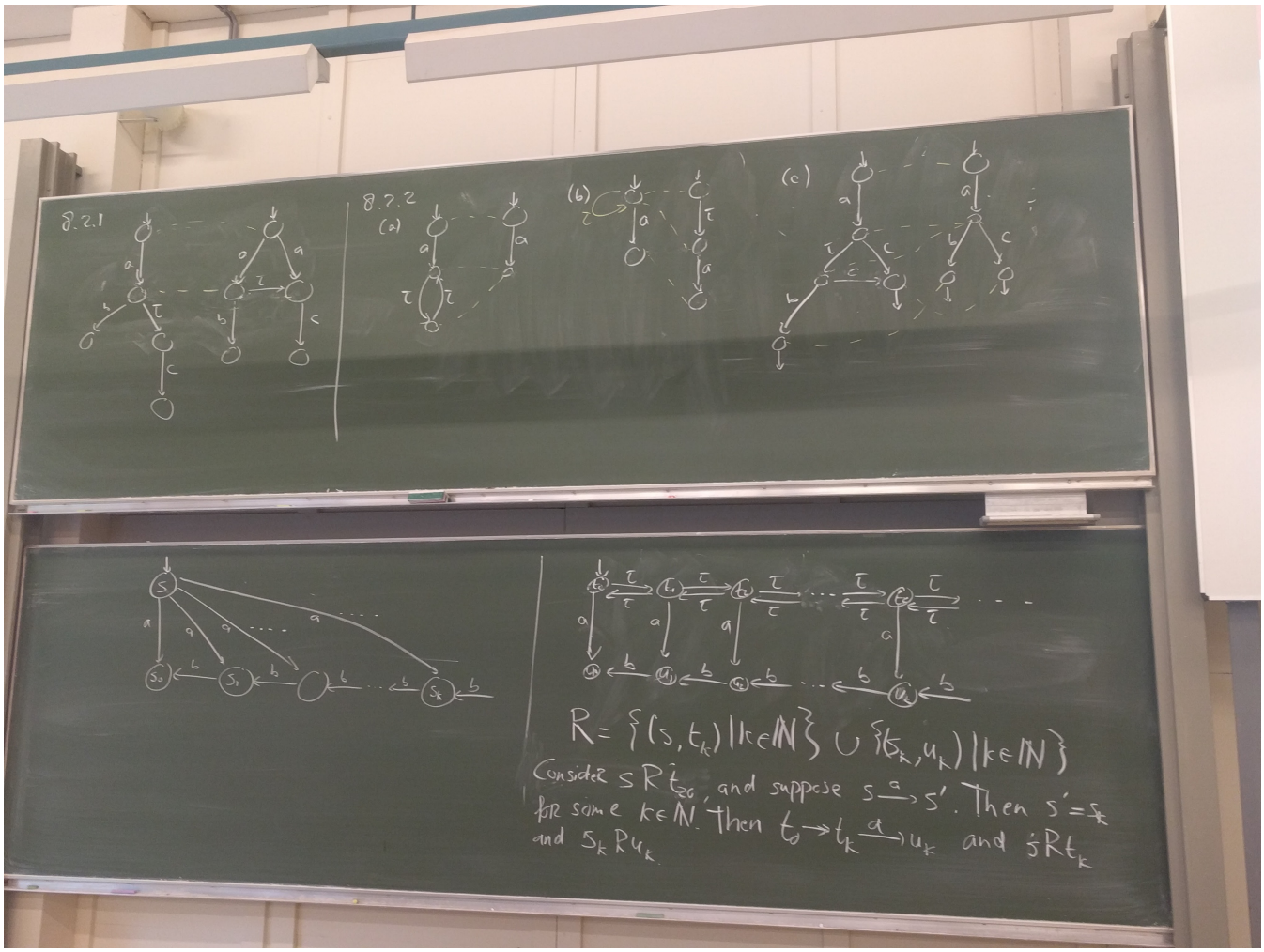
1. If $s \xrightarrow{a} s'$ for some $a \in L$ and $s' \in S$,
   Then $\exists t', t'' \in S$ such that $t \twoheadrightarrow t'' \xrightarrow{(a)} t'$, where $s \; R \; t''$ and $s' \; R \; t'$.
2. Vice versa.
3. For termination: if s wants to terminate, then there exists a t' such that $t \twoheadrightarrow t'$, $t' \downarrow$ and $s \; R \; t'$
4. Vice versa.

An advantage of using *branching bisimilarity* is that in a way that keeps the choice points, as well as being a strong notion of bisimilarity.

Branching bisimilarity is an *equivalence* that is *compatible* with the operations of the algebra. However, we do not have a *congruence*, i.e. if we have multiple terms that are branching bisimilar, then their $+$ might not be bisimilar. Recall: congruence on an algebra is an equivalence that is compatible with the operations of the algebra.

Note: if you have two transition systems which are branching bisimilar, and one of them does not contain any taus, then if you remove the taus from the other transition system then it should result in a bisimilarity relation.

Here's an example of the cases:



Lecture 12 -

Branching Bisimilarity examples

8.2.1 is not branching bisimilar, nor rooted branching bisimilar. 8.2.2a is branching bisimilar and rooted branching bisimilar. 8.2.2b is branching bisimilar as well as rooting bisimilar. 8.2.2c is branching bisimilar. Notice that the τ step does *not* make a choice. It only increases the options. If it would decrease the options by making a choice then it would not be bisimilar.

8.3.2:

$$
\begin{aligned}
BSP_\tau(A) \vdash a.\tau.x &= a.x \\
&= a(\tau x + 0) && [A6] \\
&= a(\tau(x+0)+0) && [A6] \\
&= a(\tau(0+x)+0) && [A1] \\
&= a(0+x) && [B] \\
&= a.x && [A1, A6]
\end{aligned}
$$

8.3.3:

$$\begin{aligned} BSP_\tau(A) \vdash a.(\tau.x + x) &= a(\tau(x+0)+x) && [A6] \\ &= a(x+0) && [B] \\ &= a.x && [A6] \end{aligned}$$

8.3.1:

$$\begin{aligned} BSP_\tau(A) \vdash a.\tau(\tau.b.1 + \tau.\tau.b.1) &= a.\tau(\tau.\tau.b.1 + \tau.b.1) && [A1] \\ &= a.\tau(\tau.b.1) && [8.3.3] \\ &= a.\tau.b.1 && [8.3.2] \\ &= a.\tau.(\tau.b.1 + b.1) && [8.3.3] \end{aligned}$$

Rooting branching bisimilar

Still, we want branching bisimilarity to be a congruence. How do we fix this? Well, with **rooted branching bisimulation**.

A branching bisimulation R satisfies the **root condition** such that $s \; R \; t$ and:

1. If $s \xrightarrow{a} s'$ for some $a \in L$ and $s' \in S$ (Notice that $\tau \in L$.)
   Then $\exists t'$ such that $t \xrightarrow{a} t'$ and $s' \; R \; t'$.
2. Vice versa.
3. $s \downarrow \implies t \downarrow$
4. Vice versa.

We denote this as $s \leftrightarrows_{rb} t$. We just add one more axiom to $BSP(A)$ to get a sound + ground-complete axiomatisation of $\mathbb{P}(BSP_\tau(A))_{\backslash \leftrightarrows_{rb}}$.

$$a.(\tau.(x+y)+x) = a(x.y)$$

Notice that this a can also be a τ itself!

TCP

Finally, we can have a new theory called $TCP_\tau(a, \gamma)$, combining everything.

(With waaaay too many axioms, like 30-40 or so.)

Fixing tau problems

Problems with tau encapsulation

So now's the question, can we encapsulate τ? No, this results in a problem:

$$\partial_{\{\tau\}}(a.\tau.1) = a.\partial_{\{\tau\}}(\tau.1) = a.0$$

And by exercise 8.3.2 we know that:

$$\partial_{\{\tau\}}(a.\tau.1) = \partial_{\{\tau\}}(a.1) = a.1$$

Problem with tau recursion

If τ is an action,t hen the following recursive specification should be guarded:

$$\{X = \tau.X\}$$

However, $X \mapsto \tau.1$ is a solution and so is $X \mapsto \tau.0$. Thus we have two distinct solutions, not satisfying RSP anymore. Therefore, we cannot allow τ to be a guard.

Furthermore, we cannot abstract with $\tau()$ on the right hand side of the equation.

Theorem

RDP and RSP are valid in $\mathbb{P}(TCP_{\tau,rec}(A,\gamma))$

Lecture 13 - Proving RSP

Recursion

We have a set of actions A, a theory $T(A)$ (such as $BSP(A)$). Then we can have a recursive specification *Rec* over $T(A)$. $T(A)_{rec}$ is an extension of $T(A)$ with for every rec. spec. E over $T(A)$ and every $(X = t_X) \in E$.

1. Constant symbol $\mu X.E$
2. An axiom $\mu X.E = \mu t_X.E$
   (Where the latter is a shorthand notation of distributing μ over all the terms.)

For example: a recursive specification over $BSP(A)$ is $X = a.X + 1$.

Also, $\leftrightarrows$ is a congruence over $BSP(A)$.

Recursive Definition Principle

Recursive Definition Principle (RDP): Let Σ be a signature with an algebra $\mathbb{A}$. It satisfies RDP if every recursive specification has at least one solution.

Theorem

$\mathbb{P}(BSP_{rec}(A)) \models RDP$

Proof: Let E be a rec spec. Define κ as the extension of ι such that, for every recursion variable X in E:

$$\kappa(X) = [\mu X.E]_{\leftrightarrows}$$

Then we have to show that $\kappa(X) = \kappa(t_X)$, which can be done by showing that these classes are equal modulo bisimilarity which can be done with the TDS.

Projection

Define the unary projection operator π_n, where $n \in \mathbb{N}$. $\pi_n(p)$ executes the behaviour of p up to depth n. (Stopping is equivalent to a deadlock, $\mathbf{0}$.)

Question, does this hold? $\pi_{n+1}(p) \leftrightarrows \pi_{n+1}(q) \implies \pi_n(p) \leftrightarrows \pi_n(q)$.

> ***Proof***
>
> *Try 1*

> *Base case:* $\pi_1(p) \leftrightarrows \pi_1(q) \implies \pi_0(p) \leftrightarrows \pi_0(q)$ *Trivial, since* $\pi_0(p) \leftrightarrows 0 \leftrightarrows \pi_0(q)$.
>
> *To prove:* $\pi_{n+1}(p) \leftrightarrows \pi_{n+1}(q) \implies \pi_n(p) \leftrightarrows \pi_n(q)$
>
> *We know that there exists a bisimulation relation* R *such that* $\pi_{n+1}(p)\ R\ \pi_{n+1}(q)$.
>
> *Then, let us define* R'. *Well, we're then in a bit of a problem, therefore, let us use co-induction.*
>
> *Try 2*
>
> *We prove that* $R = \{(\pi_n(p), \pi_n(q) \mid p, q \in \mathcal{C}(BSP + PR)_{rec}), n \in \mathbb{N}, \pi_{n+1}(p) \leftrightarrows \pi_{n_1}(q)\}$.
>
> 1. *If* $\pi_n(p) \xrightarrow{a} r$, *then* $n > 0$ *and* $r \equiv \pi_{n-1}(p')$ *for some* p' *such that* $p \xrightarrow{a} p'$.
>   *Hence, since* $\pi_{n+1} \leftrightarrows \pi_{n+1}(q)$ *and* $\pi_{n+1}(p) \xrightarrow{a} \pi_n(p')$, *there exists* q' *such that* $q \xrightarrow{a} q'$ *and* $\pi_{n+1}(q) \xrightarrow{a} \pi_n(q')$.
>   *Because of the definition of bisimilarity, we know that* $\pi_n(p') \leftrightarrows \pi_n(q')$. *Because of the definition of* R, *we know that* $\pi_{n-1}(p)\ R\ \pi_{n-1}(q)$ *as we just apply the definition for* $n := n - 1$.
> 2. *Vice versa*
> 3. *If* $\pi_n(p) \downarrow$ *then* $p \downarrow$, *so* $\pi_{n+1}(p) \downarrow$ *therefore* $\pi_{n+1}(q) \downarrow$, *thus* $\pi_n(q) \downarrow$.
> 4. *Vice versa.*

There are also some axioms with projections, but these are relatively forward. The only one that is interesting is: $\pi_0(1) = 1$.

Approximation Induction Principle (AIP)

Suppose that we have algebra $\mathbb{A}$ and projection operators $\pi_n(n \in \mathbb{N})$. This algebra satisfies AIP if, for any terms s and t, $\mathbb{A} \models \pi_n(s) = \pi_n(t)$ for all $n \in \mathbb{N}$, then $\mathbb{A} \models s = t$.

Basically, if all finite projections of s and t are equal to each other for every possible n, then these terms are equal to one another.

However, this doesn't hold in our term model $\mathbb{P}((BSP + PR)_{rec}(A))_{/\leftrightarrows}$, because of lack of bisimilarity! See the following example.s

> *"But... There is a big but[t]" - Bas Luttik, 2017*

Consider the recursive specification:

$$\{X_n = a^n.0 + X_{n+1} \mid n \in \mathbb{N}\} \cup \{Y = a.Y\}$$

Then $X_0 = \sum_{i=0}^{n} a^i.0 + X_{n+1}$ for all $n \in \mathbb{N}$ Therefore, by applying $A3$, $A1$ and $A2$, we can show that $X_0 = X_0 + a^n.0$. Then

$$\pi_n(X_0 + Y) = \pi_n(X_0) + \pi_n(Y) = \pi_n(X_0) + a^n.0 = \pi_n(X_0) + \pi_n(a^n.0) = \pi_n(X_0 + a^n.0) = \pi_n(X_0)$$

(Although we need an induction for the second-to-final step.)

However, when we draw the transition systems of $X_0 + Y$ and X_0, then these are not bisimilar!

Restriction to AIP

We denote that a term s is **finitely branching** if $\forall s'$ reachable from s the set $\{s'' \mid \exists a \in A : s' \xrightarrow{a} s''\}$ is finite.

When we restrict AIP to only terms that are finitely branching, then we call it AIP^-. We know that AIP- is valid in our term algebra.

Head normal form

The set of **head normal forms** for $T(A)$ A head normal form can be written as a sum of prefixes. From a head normal form, we know that it's finitely branching.

Furthermore, we can prove that in our term model, if a term s is guarded, then it's in head normal form and therefore is finitely branching.

Recursive Specification Principle

Reminder: Let Σ be a signature, then its algebra $\mathbb{A}$ satisfies the RSP principle if every *guarded* recursive specification E has at most one solution.

Theorem - Projection Theorem

Suppose we have s and t satisfying the solution X in E it holds that $\pi_n(s) = \pi_n(t)$ for all $n \in \mathbb{N}$.

Corollary: $\mathbb{P}((BSP + PR)_{rec}(a))_{/\leftrightarrows}$ satisfies RSP.

Proof: We can show that since $\pi_n(s) = \pi_n(t)$, we know that $s \leftrightarrows t$ by AIP^-.

Bags and Queues

Bags

Consider the behaviour of a bag. This is a multiset over which every element, of say D, can be any n number of times in the bag.

A bag is not regular (aka infinitely many distinct non-bisimilar states), hence not finitely definable in $BSP(A)$.

However, this is finitely definable (with 1 line) in $BCP(A, \emptyset)$, aka with parallelism, but not finitely definable in $TSP(A)$.

TCP vs BCP

Question: Is every finitely definable $TSP(A)$ behaviour also finitely definable over $BCP(A, \gamma)$.
Conjecture: No!

Queue

With both sequential composition and parallel composition together, we can define a FIFO queue with finitely many recursive specification rules.

However, this is not finitely definable in $TCP(A)$ and $BCP(A, \gamma)$.

Lecture 14 - Random Exercises

Exercise 4.3.5

Goal: Proof Lemma 4.3.9

Lemma 4.3.9: If $(p + q) + r \leftrightarrows r$, then $p + r \leftrightarrows r$ and $q + r \leftrightarrows r$.

> *Proof*
>
> *Let R be a bis. rel. such that $(p+q)+r\ R\ r$*
>
> *Define $R' = \{(s+t,t) \mid (s+q)+t\ R\ t, \forall s,t \in \mathcal{C}(\mathrm{MPT}(A))\} \cup R \cup \{(t,t) \mid \mathcal{C}(MPT(A))\}$*
>
> *Then, if $s+t \xrightarrow{a} u$, then either $s \xrightarrow{a} u$ (case 1) or $t \xrightarrow{a} u$ (case 2).*
>
> ***Case 1:** If $s \xrightarrow{a} u$, then $(s+q)+t \xrightarrow{a} u$, so $t \xrightarrow{a} v$ such that $u\ R\ v$, and hence $u\ R'\ v$. **Case 2:** If $t \xrightarrow{a} u$, then note that $t\ R\ t'$.*
>
> *Similarly we can create a relation such that $q+r \leftrightarrows r$, as we know that $(p+q)+r \leftrightarrows (q+p)+r$.*

Lecture 15 - More random exercises

Exercise 5.5.6: Consider the recursive specification $\{X = a.X + b.X\}$. Determine $\pi_n(X)$ for all $n \in \mathbb{N}$.
Solution:

$$\begin{aligned} \pi_0(X) &= \pi_0(a.X + b.X) && [Rec] \\ &= \pi_0(a.X) + \pi_0(b.X) && [PR5] \\ &= 0 + 0 && [PR] \\ &= 0 && [A6] \end{aligned}$$

$$\pi_1(X) = \ldots = a.0 + b.0$$

What about:

$$\pi_n(X) \stackrel{?}{=} a.\pi_{n-1}(X) + b.\pi_{n-1}(X)$$

However, we want to get rid of π! There are two ways to do so:

$$\pi_n(X) = (a.1 + b.1)^n$$

Or perhaps even inductively: Define $p_0, p_1, p_2, \ldots \in \mathcal{C}(BSP(A))$ with induction on n as follows:

$$\begin{aligned} p_0 &\equiv 0 \\ p_{n+1} &= a.p_n + b.p_n \end{aligned}$$

We can actually confirm that the former equals p_n. We can proof this with induction.

> *Note: we can use the projection to proof that two equations are equal, using AIP^-, by:*
>
> - *Proving that all the projections for $n \in \mathbb{N}$ are equal..*
> - *One of the two equations is in Head Normal Form or has a guarded specification / is finitely branching.*

Exercise 6.6.1: Prove, using AIP^-, that any solution of $X = a.X \cdot b.1$ is also a solution of $X = a.X$.

Solution: To prove: $\pi_n(X) = \pi_n(Y)$ for all $n \in \mathbb{N}$. Proof by induction on $n \in \mathbb{N}$. If $n = 0$, then:

$$\begin{aligned}\pi_n(X) = \pi_0(X) &= \pi(a.\,X \cdot b.1)\\ &= 0\\ &= \pi_0(a.\,Y)\\ &= \pi_0(Y) = \pi_n(Y)\end{aligned}$$

Welp, induction step is not going that well.

That's why we want to do something different: $\pi_n(X \cdot p) = \pi_n(Y), \forall n \in \mathbb{N}, \forall p \in \mathcal{C}(TSP(A))$.

Since we prove for all $\pi_n(X)$, we know that $\pi_n(X \cdot 1) = \pi_n(Y)$.

Exercise 6.4.2: Prove by structural induction that for all closed $TSP(A)$-terms p and q and $n \geq 0$,

$$(TSP + PR)(A) \vdash \pi_n(p \cdot q) = \pi_n(\pi_n(p) \cdot \pi_n(q))$$

Solution:

Note that we know that $\pi_n(\pi_n(p)) = \pi_{\min(m,n)}(p)$. Furthermore, notice by the elimination theorem for $TSP(A)$ there exists for every $p \in \mathcal{C}(TSP(A))$ there exists a $p' \in \mathcal{C}(BSP(A))$ such that $TSP(A) \vdash p = p'$. Therefore, by the elimination theory of $TSP(A)$, if we proof it by structural induction for all closed $BSP(A)$ terms, then it should also hold for $TSP(A)$.

Base case: We do induction on the structure of p. Furthermore, we know that $n = 0$.

- $p \equiv 0 : \pi_n(p \cdot q) = \pi_n(0) = \pi_n(0 \cdot \pi_n(q)) = \pi_n(\pi_n(0) \cdot \pi_n(q))$
- $p \equiv 1 : \pi_n(p \cdot q) = \pi_0(q) = \pi_n(\pi_n(q)) = \pi_n(1 \cdot \pi_n(q)) = \pi_n(\pi_n(1) \cdot \pi_n(q))$
- $p \equiv a.p'$:
  $\pi_n(p \cdot q) \equiv \pi_n(a.p' \cdot q) = \pi_n(a.(p' \cdot q)) = 0 = \pi_n(0) = \pi_n(0 \cdot \pi_0(q) = \pi_0(\pi_(a.p') \cdot \pi_n(q) = \pi_n(\pi_n(p) \cdot \pi_n(q))$
- For this, we need to formulate for appropriate IH.
  $p = p_1 + p_2$:
  $\pi(p \cdot q) \equiv \pi_n((p_1 + p_2) \cdot q) = \pi_0(p_1 \cdot q) + \pi_0(p_2 \cdot q) \overset{IH}{=} \pi_0(\pi_0(p_1) \cdot \pi_0(q)) + \pi_0(\pi_0(p_2) \cdot \pi_0(q))$
  $= \pi(\pi(p_1) \cdot \pi_0(q) + \pi_0(p_2) \cdot \pi_0(q)) = \pi_0((\pi_0(p_1) + \pi_0(p_1)) \cdot \pi_0(q)) = \pi_n(\pi_n(p) \cdot \pi_n(q))$

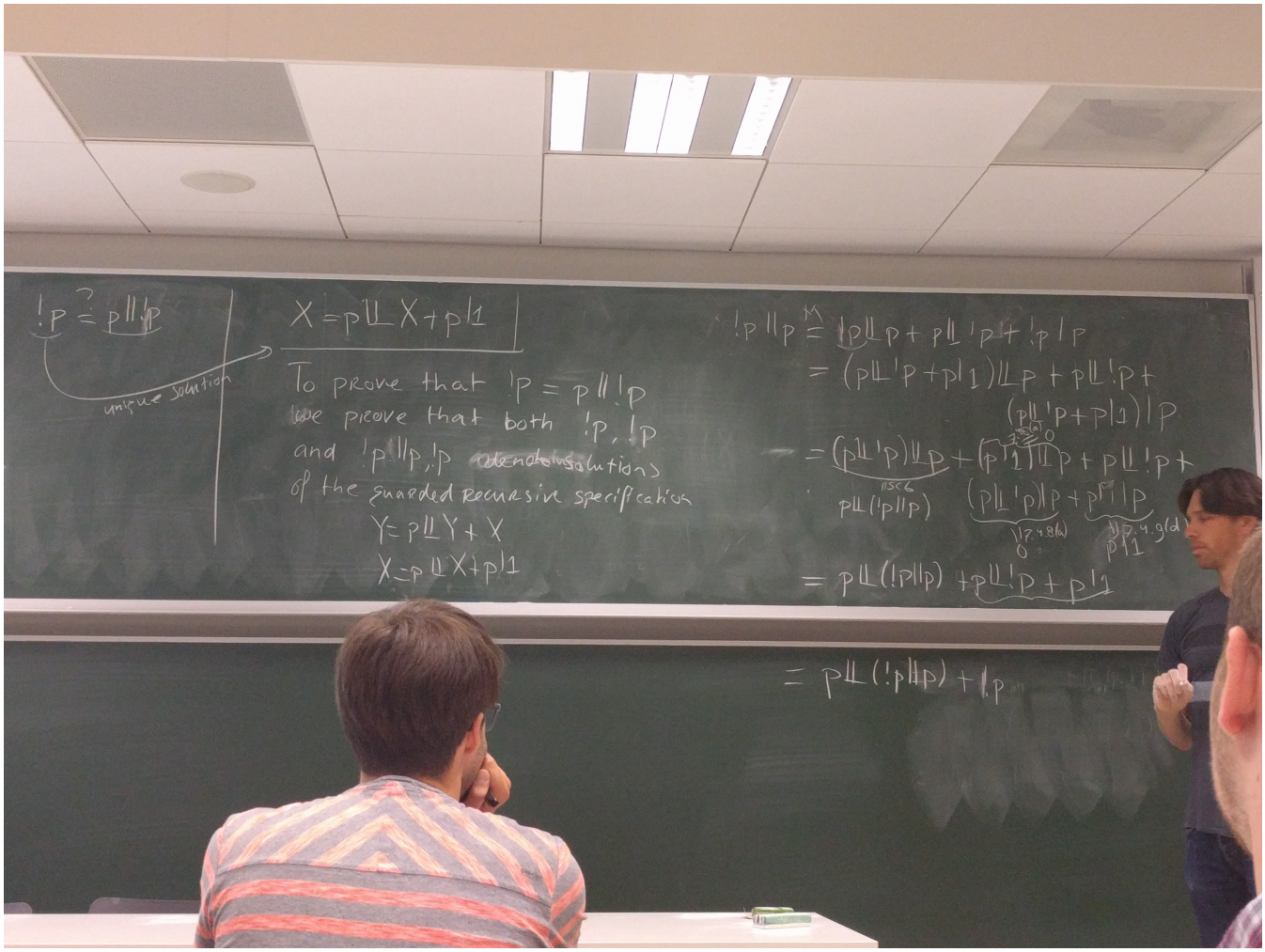
Induction step: Let $n \in \mathbb{N}$. We want to proof it holds for $n + 1$.

Exercise 7.6.9: The process $!p$ denotes the *replication* of p, denoting the process term $p \parallel p \parallel p \parallel \cdots$. This cannot be defined by $X = p \parallel X$, since X is unguarded and has infinitely many solutions. However, the equation $X = p \between X + p \mid 1$ *can* be defined for the intended purpose.

We want to prove that $!p = p \parallel\ !p$. To do so, we prove that both $!p, !p$ and $!p \parallel p, !p$ denotes solutions to the guarded recursive specification

$$\begin{aligned}Y &= p \between Y + X\\ X &= p \between X + p \mid 1\end{aligned}$$

Here's a proof:



Lecture 16 - Yet even more exercises

Exam questions

Exam June 2016, question 2:

1. We want to prove that $\sim_T$ is a congruence. To do so, we need to prove two things: that $\sim_T$ is an equivalence relation and that $\sim_T$ is *compatible* with all operations of $MPT(A)$.
    1. Why is it an equivalence?
    (We want it to hold it for all $MPT(A)$ terms, so we declare an arbitrary one and show it holds.)
    Reflexivity: Let $p \in \mathcal{C}(MPT(A))$, then clearly $tr(p) = tr(p)$, thus $p \sim_T p$.
    Symmetry: Similarly to reflexivity.
    Transitivity: Similar to Symmetry.
    2. Why is it compatible with $MPT(A)$.
    Case $a._$: Let $p, q \in \mathcal{C}(MPT(A))$ and suppose that $p \sim_T q$. We need to prove that $a.p \sim_T a.q$.
    Note that $tr(a.p) = \{a.\sigma \mid \sigma \in tr(p)\} \cup \{\epsilon\} = \{a.\sigma \mid \sigma \in tr(q)\} \cup \{\epsilon\} = tr(a.q)$.
    Thus, we have that $a.p \sim_T a.q$
    (If we want to answer the question more formally, then we would have to look at the operational semantics, aka TDS, and notice that $a.p$ could only do an a step to p, therefore when looking at the definition of trace, then ...)
    Case $\_ + \_$: Is similar to $a._$.

2. Note that $a.(b.0 + c.0) \sim_T a.b.0 + a.c.0$. For $tr(a.(b.0 + c.0)) = \{\epsilon, a, ab, ac\} = tr(a.b.0 + a.c.0)$. However, $a.(b.0 + c.0) \not\leftrightarrows a.b.0 + a.c.0$, as when we try to draw the transition system and relate them, then we get some trouble with the relation.
3. Suppose $MPT(A)$ would be a ground-complete axiomatisation for $\mathbb{P}(MPT(A))_{\sim_T}$, then for all $p, q \in \mathcal{C}(MPT(A))$ such that $\mathbb{P}(MPT(A))_{/\sim_T} \models p \sim_T q$ we would have that $MPT(A) \vdash p = q$.

   Now consider the equation $a.(b.0 + c.0) = a.b.0 + a.c.0$. This equation is valid in $\mathbb{P}(MPT(A))_{/\sim_T}$, thus $MPT(A) \vdash a.(b.0 + c.0) = a.b.0 + a.c.0$.

   Since $MPT(A)$ is sound for the algebra $\mathbb{P}(MPT(A))_{/\leftrightarrows}$, it would follow that $a.(b.0 + c.0) \leftrightarrows a.b.0 + a.c.0$, quod non (see (b)). We have derived a contradiction from the assumption that $MPT(A)$ is a ground-complete axiomatisation for $\mathbb{P}(MPT(A))_{/\sim_T}$, thus it is not.

Exam June 2016, question 2:

1. $$\begin{aligned} X &\equiv \delta_H(\mu.B.E \parallel \mu C.E) \\ &= \sum_{d \in D} i?d.\underbrace{\delta_H(\mu B_d.E \parallel \mu C.E)}_{X_d} \equiv \sum_{d \in D} i?d.X_d \end{aligned}$$

   Let us now derive X_d:

   $$\begin{aligned} X_d &\equiv \delta_H(\mu B_d.E \parallel \mu C.E) \\ &= l!? \perp.\underbrace{\delta_H(\mu B_d.E \parallel \mu C.E)}_{X_d} + l!?d.\underbrace{\delta_h(\mu B_d.E \parallel o!d.\mu C.E)}_{Y_d} = l!? \perp.X_d + l!?d.Y_d \end{aligned}$$

   Now, let's derive Y_d:

   $$\begin{aligned} Y_d &= \delta_h(\mu B_d.E \parallel o!d.\mu C.E) \\ &= \sum_{e \in D} i?e.\underbrace{\delta_H(\mu B_e.E \parallel o!d.\mu C.E)}_{Z_{de}} + o!d.\underbrace{\delta_H(\mu.B.E \parallel \mu C.E)}_{X} = \sum_{e \in D} i?e.Z_{de} + o!d.X \end{aligned}$$

   Let us now continue with Z_{de}:

   $$\begin{aligned} Z_{de} &= \delta_H(\mu B_e.E \parallel p!d.\mu C.E) \\ &= o!d.\underbrace{\delta_H(\mu B_e.E \parallel \mu C.E)}_{X_e} = o!d.X_e \end{aligned}$$

   Thus, we have a recursive specification that we can write down.

   Look at the terms at the start of each derivation of the equation. We can call them p, p_d, q_d and r_{de} respectively. In the above derivation we have established that $p, p_d(d \in D), q_d(d \in d)$, and $r_{de}(d.e \in D)$ denotes a solution for F. Since $X, X_d(d \in D), Y_d$, and Z_{de} are also a solution of this recursive specification, it follows by RSP that these solutions must be the same, as there can only be one solution. Therefore we have that $p = X$.
2. The idea here is to draw the transition system of $\tau_I(\mu X.E)$, since we proved in the first exercise that $\mu X.E$ is the same as $\delta_H(\mu B.E \parallel \mu C.E)$. Then we draw the transition system of $\mu B.G$ and give a relation R such that these are rooting branching bisimilar.

Question 1 - Exam 2015:

1. 1. Prove the first equation using only axioms.

2. Prove via structural induction. **Tip:** Look at elimination theorem in the book. It's the same as BCP_τ.

2. 1. Use proof trees for this.
   2. We've done this one before in the lectures.
   3. Identify an infinite subset of terms and argue that these states are not pairwise bisimilar. Joost: 🚲