# 1. Logiken und Strukturen

## 1.1. Mathematische Logik

**Definition 1.1**
Eine *relationale Signatur* ist ein Tupel $\tau = (R_1, .., R_n)$ aus Relationensymbolen. Jedes Symbol $R_i$ hat eine Stelligkeit $ar(R_i)$. In dieser Vorlesung betrachten wir nur 1- und 2-stellige Relationen. Eine $\tau$-Struktur ist ein Tupel $M = (U, R_1, .., R_n)$ wobei

- $U$ — das *Universum* — eine Menge ist, und
- für alle $i = 1, ..n$: $R_i \subseteq U^{ar(R_i)}$ eine Relation auf $U$ ist.

**Beispiel 1.1**
Sei $P$ ein Java-Programm mit den globalen `int`-Variablen $x_1, .., x_n$. $U = \mathbb{Z}^n$ die möglichen Belegungen auf dem Heap, $R_i = (x_i == 0)$.

**Beispiel 1.2**
Ein Fussgängerüberweg mit Ampel lässt sich als relationale Struktur modellieren. Universum $U = \{\texttt{rot}, \texttt{rotgelb}, \texttt{gruen}, \texttt{gelb}\} \times \{\texttt{rot}, \texttt{gruen}\}$, eine 2-stellige Relation

$$\begin{aligned} \mathit{Next} = \{&((\texttt{rot}, \texttt{rot}), (\texttt{rotgelb}, \texttt{rot})),\\ &((\texttt{rot}, \texttt{rot}), (\texttt{rot}, \texttt{gruen})),\\ &((\texttt{rotgelb}, \texttt{rot}), (\texttt{gruen}, \texttt{rot})),\\ &((\texttt{gruen}, \texttt{rot}), (\texttt{gelb}, \texttt{rot})),\\ &((\texttt{gelb}, \texttt{rot}), (\texttt{rot}, \texttt{rot})),\\ &((\texttt{rot}, \texttt{gruen}), (\texttt{rot}, \texttt{rot}))\} \end{aligned}$$

und eine 1-stellige Relation $\mathit{Splash} = \{(\texttt{gruen}, \texttt{gruen})\}$.

**Definition 1.2**
Wir definieren Logiken durch ihre Syntax und ihre Semantik. Die Syntax ist eine formale Sprache über einem Alphabet, welches normalerweise $\tau$ enthält. Ein Element $\varphi$ dieser Sprache wird *Formel* genannt.

Formeln können *Variablen* enthalten. Dabei unterscheiden wir erst- und zweitstufige. Erststufige Variablen stehen für Elemente des Universums, zweitstufige für Relationen. Eine *Interpretation* für eine Formel $\varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$, die möglicherweise freie Variablen enthält ($n \geq 1$, $m \geq 1$), ist eine $\tau$-Struktur zusammen mit einer (möglicherweise leeren) Menge von Elementen und Relationen $\mathcal{I} = M, a_1, \ldots, a_n, R_1, \ldots, R_m$.

Die *Semantik* ist eine Relation $\models$ zwischen Interpretationen und Formeln. Wir schreiben $\mathcal{I} \models \varphi(x_1, \ldots, x_n, X_1, \ldots, X_m)$ (sprich: $\mathcal{I}$ erfüllt $\varphi$) um auszudrücken, dass $\mathcal{I}$ ein *Modell* von $\varphi$ ist. Wir schreiben auch $[\![\varphi]\!] = \{\mathcal{I} \mid \mathcal{I} \models \varphi\}$ für die Semantik von $\varphi$.

Wir wollen nicht nur Algorithmen angeben, mit denen sich gewisse Probleme bzgl. einer Logik lösen lassen, sondern auch verstehen, wie schwer diese Probleme wirklich sind. Dazu hat die Komplexitätstheorie die Begriffe der *Härte* und der *Vollständigkeit* eingeführt, die im Falle der Komplexitätsklasse NP aus Info IV bekannt sind. Die meisten der hier angesprochenen Logiken haben jedoch wesentlich schwierigere Probleme. Für eine knappe Einführung in die für diese Vorlesung wichtigen Begriffe der Komplexitätstheorie siehe Anhang A.

An einer Logik $L$ interessieren uns insbesondere die folgenden Probleme:

1. *Erfüllbarkeit*: Gegeben eine Formel $\varphi \in L$, gibt es eine Interpretation $\mathcal{I}$, so dass $\mathcal{I} \models \varphi$ gilt? Wie schwierig ist es (obere und untere Komplexitätsschranken), das Erfüllbarkeitsproblem zu lösen?

   Erfüllbarkeit einer Formel belegt z.B. die Realisierbarkeit eines Vorhabens. Sind $A_1$,$A_2$ Aussagen, die ein Programm erfüllen soll, dann besagt die Erfüllbarkeit von $A_1$ *und* $A_2$, ob ein solches Programm überhaupt existieren kann.

2. *Allgemeingültigkeit*: Gegeben eine Formel $\varphi \in L$, sind alle möglichen Interpretationen $\mathcal{I}$ ein Modell von $\varphi$? In diesem Fall schreiben wir auch $\models \varphi$.

3. *Model Checking*: Gegeben eine Interpretation $\mathcal{I}$ und eine Formel $\varphi$, gilt $\mathcal{I} \models \varphi$? Wie schwierig ist es, das Model Checking Problem zu lösen?

   Model Checking ist neben Erfüllbarkeit das Hauptmittel zur Programmverifikation: Ein Programm wird in eine relationale Struktur abstrahiert, die zu verifizierende Eigenschaft als logische Formel formuliert. Da wir meistens zu einer gegebenen Eigenschaft ein korrekt arbeitendes Programm haben wollen, untersuchen wir neben der kombinierten Komplexität des Model Checking Problems (Eingabe = Struktur + Formel) vor allem auch die Datenkomplexität einer Logik (Eingabe = Struktur, Formel wird als fest angesehen).

   Manchmal ist es auch sinnvoll, die Ausdruckskomplexität einer Logik zu untersuchen (Eingabe = Formel, Struktur ist fest). Diese beschreibt in gewissem Sinne, wie schwierig es ist, zu einem gegebenen Programm die Eigenschaften zu finden, die es erfüllt.

   Offensichtlich gilt, dass die Daten- und Ausdruckskomplexität einer Logik höchstens so schwer ist wie die Komplexität ihres kombinierten Model Checking Problems. Dies gilt sowohl für untere wie auch obere Schranken.

   Wir unterscheiden darüberhinaus das *lokale* und das *globale* Model Checking Problem. Bei ersterem ist eine Interpretation $\mathcal{I}$ und eine Formel $\varphi$ gegeben, und es soll bestimmt werden, ob $\mathcal{I} \models \varphi$ gilt oder nicht. Bei letzterem ist eine Struktur $\mathcal{M}$ und eine Formel $\varphi$ gegeben, und es sollen alle Interpretationen $\mathcal{I} = \mathcal{M}, \ldots$ berechnet werden, für die $\mathcal{I} \models \varphi$ gilt. Offensichtlich ist das lokale Model Checking Problem höchstens so schwer wie das globale.

4. *Ausdrucksstärke* und *Prägnanz*: Welche Eigenschaften können in welcher Logik ausgedrückt werden? Kann eine Logik mehr als eine andere ausdrücken? Wenn eine

Eigenschaft in zwei verschiedenen Logiken ausdrückbar ist, wie lang ist jeweils die Formel?

5. *Modelleigenschaften*: Hat jede erfüllbare Formel $\varphi$ der Logik $L$ ein Modell $\mathcal{M}$ mit einer gewissen Eigenschaft? Insbesondere interessieren uns die *endliche Modelleigenschaft* (ist $|\mathcal{M}| < \infty$?), die stärkere *kleine Modelleigenschaft* (gibt es eine Funktion $f : L \to \mathbb{N}$, so dass jedes erfüllbare $\varphi \in L$ ein Modell $\mathcal{M}$ hat, so dass $|\mathcal{M}| \leq f(\varphi)$ gilt?), sowie die *Baummodelleigenschaft* (ist $\mathcal{M}$ ein Baum?).

6. *Vollständige Axiomatisierung*: Gibt es ein System aus Axiomen und Regeln, in dem genau diejenigen $\varphi$ herleitbar sind, für die $\models \varphi$ gilt?

**Definition 1.3**
Eine Logik $L$ heisst *unter Komplement angeschlossen*, wenn für jedes $\varphi \in L$ ein $\psi \in L$ exisitiert, so dass für alle Interpretationen $\mathcal{I}$ gilt: $\mathcal{I} \models \varphi$ gdw. $\mathcal{I} \not\models \psi$. Wir schreiben auch $\overline{\varphi}$ für das Komplement von $\varphi$.

Ist eine Logik unter Komplement abgeschlossen, dann sind (1) und (2) im Grunde dasselbe Problem.

**Lemma 1.1**
Ist $L$ unter Komplement abgeschlossen, dann gilt für alle $\varphi \in L$: $\varphi$ ist unerfüllbar gdw. $\models \overline{\varphi}$.

**Definition 1.4**
Seien $L$ und $L'$ zwei Logiken, die über derselben Klasse $\mathcal{K}$ von Strukturen interpretiert werden ($L = L'$ ist auch möglich). Seien $\varphi \in L$ und $\psi \in L'$. Dann heissen $\varphi$ und $\psi$ *äquivalent*, geschrieben $\varphi \equiv \psi$, falls für alle Interpretationen $\mathcal{I}$ gilt:

$$\mathcal{I} \models \varphi \quad \text{gdw.} \quad \mathcal{I} \models \psi$$

Eine Logik $L'$ *subsumiert* eine Logik $L$, wenn für alle $\varphi \in L$ es ein $\psi \in L'$ gibt, so dass $\varphi \equiv \psi$. In diesem Fall schreiben wir $L \leq L'$ und $L \equiv L'$, falls $L \leq L'$ und $L' \leq L$.

## 1.2. Die Schwächen der Aussagenlogik

Sei $\tau = \emptyset$. Aussagenlogische Formeln können über der Klasse, die nur die Struktur $\mathcal{B} = (\{0, 1\})$ enthält, interpretiert werden. Sei $\mathcal{V}$ eine höchstens abzählbar große Menge von 1-stufigen Variablen. Die Syntax der *Aussagenlogik* ist definiert durch die kontextfreie Grammatik

$$\varphi \ ::= \ x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi$$

wobei $x \in \mathcal{V}$.

Die Semantik einer aussagenlogischen Formel $\varphi(x_1, \ldots, x_n)$ kann leicht induktiv definiert werden. Eine Interpretation ist hier ein Tupel $\mathcal{B}, \bar{b}$, wobei $\bar{b} = b_1, \ldots, b_n$ und

$b_i \in \{0,1\}$ für alle $i = 1, \ldots, n$.

$$\begin{aligned} \mathcal{B}, \bar{b} \models x_i \quad &\text{gdw.} \quad b_i = 1 \\ \mathcal{B}, \bar{b} \models \varphi \vee \psi \quad &\text{gdw.} \quad \mathcal{B}, \bar{b} \models \varphi \text{ oder } \mathcal{B}, \bar{b} \models \psi \\ \mathcal{B}, \bar{b} \models \varphi \wedge \psi \quad &\text{gdw.} \quad \mathcal{B}, \bar{b} \models \varphi \text{ und } \mathcal{B}, \bar{b} \models \psi \\ \mathcal{B}, \bar{b} \models \neg\varphi \quad &\text{gdw.} \quad \mathcal{B}, \bar{b} \not\models \varphi \end{aligned}$$

Sei z.B. $\varphi = \neg x_1 \vee (x_2 \wedge x_3)$. Dann ist $[\![\varphi]\!] = \{(\mathcal{B}, \bar{b}) \mid b_1 \leq \min\{b_2, b_3\}\}$

Aussagenlogik hat allerdings Schwächen. So ist es z.B. nicht auf Anhieb klar, wie man die Aussage "die Ampel steht nicht auf (`gruen`, `gruen`)" aus Bsp. 1.2 so formalisieren kann. Das Problem ist offensichtlich, dass diese Aussage davon abhängt, in welchem Zustand man sich befindet, sprich, die Wahrheit dieser Aussage hängt von den Elementen des Universums ab.

Lösungsvorschlag: Betrachte aussagenlogische Variablen als monadisch 2-stufig, die wir zur Verdeutlichung jetzt mit Großbuchstaben notieren. D.h. die Interpretation eines $X$ ist eine Teilmenge des Universums. Füge ausserdem 1-stufige Variablen $x, y, ...$ hinzu. Syntax:

$$\varphi \quad ::= \quad X(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi$$

Diese Logik interpretieren wir nun über Strukturen $M$ beliebiger Grösse. Sei $b : \mathcal{V} \to U \cup 2^U$ wobei $U$ das Universum von $M$ ist. Dann definieren wir zusätzlich:

$$M, b \models X(x) \quad \text{gdw.} \quad b(x) \in b(X)$$

Jetzt können wir das Ampelbsp. folgendermaßen modellieren. $M_{Ampel} = (U, R_1, ..., R_n)$ wobei $U = \{\texttt{rot}, ...\} \times \{\texttt{rot}, ...\}$ wie oben und $R_i$ beliebige Teilmengen von $U$ sind, z.B. $R_{grgr} = \{(\texttt{gruen}, \texttt{gruen})\}$. Dann besagt die Formel $\neg R_{grgr}(x)$ — interpretiert in einem Element $s \in U$ — dass die Ampeln in diesem "Zustand" nicht beide grün sind.

Was wir jedoch wirklich ausdrücken möchten, ist: von einem bestimmten Anfangszustand — z.B. (`rot`, `rot`) — aus ist kein Zustand (Element des Universums) via der Relation *Next* erreichbar, der zu $R_{grgr}$ gehört.

Aussagen dieser Form lassen sich geeignet über speziellen, relationalen Strukturen, sogenannten *Transitionssystemen* interpretieren. Deren Elemente sehen wir an als Zustände, 2-stellige Relationen als Zustandsübergänge (Transitionen), und 1-stellige Relationen als atomare Aussagen (Propositionen), deren Wahrheitswert in verschiedenen Zuständen unterschiedlich sein kann.

## 1.3. Transitionssysteme und Läufe

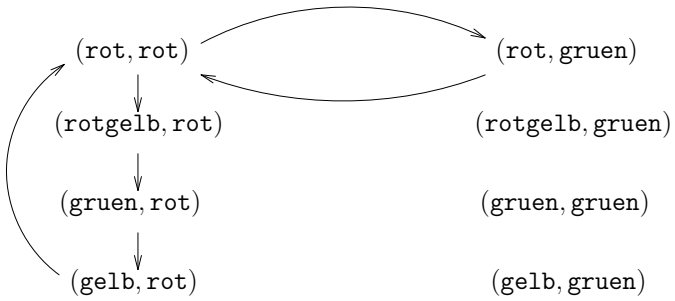
**Definition 1.5**
Sei $\Sigma$ eine endliche Menge von Aktionennamen, $\mathcal{P}$ eine endliche Menge von Propositionen. Ein Transitionssystem ist ein Tupel $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \lambda, s_0)$ wobei

- $\mathcal{S}$ eine Menge von Zuständen ist,

- $s_0 \in \mathcal{S}$ ein ausgezeichneter Anfangszustand ist,
- $\xrightarrow{a}$ für jedes $a \in \Sigma \subseteq \mathcal{S} \times \mathcal{S}$ ist,
- $\lambda : \mathcal{S} \to 2^{\mathcal{P}}$ eine Beschriftung der Zustände mit Propositionen ist.

Wir benutzen Infix-Notation $s \xrightarrow{a} t$. Ist es nicht relevant, einen ausgezeichneten Anfangszustand zu haben, dann präsentieren wir Transitionssysteme auch als entsprechende Tripel.

**Beispiel 1.3**
$\mathcal{T}_{Ampel} = (U, \{Next\}, \lambda, (\texttt{rot}, \texttt{rot}))$ ist z.B. ein Transitionssystem über einem einelementigen $\Sigma$. Transitionssysteme lassen sich leicht grafisch verdeutlichen, weil sie nichts anderes als beschriftete, gerichtete Graphen sind.




Wir bezeichnen allgemeine Transitionssysteme $\mathcal{T}$ auch als *kanten- und knotenbeschriftet*. Ist $|\Sigma| = 1$, dann heisst $\mathcal{T}$ nur knotenbeschriftet. In diesem Fall schreiben wir einfach $s \to t$.

**Definition 1.6**
Ein knotenbeschriftetes Transitionssystem $\mathcal{T} = (\mathcal{S}, \to, \lambda)$ heißt *total*, falls für alle $s \in \mathcal{S}$ es mindestens ein $t \in \mathcal{S}$ gibt mit $s \to t$.

**Definition 1.7**
Ein *Lauf* eines knotenbeschrifteten Transitionssystems $\mathcal{T} = (\mathcal{S}, \to, \lambda, s_0)$ ist eine maximale Sequenz $\pi = s_0, s_1, s_2, ...$, so dass für alle $i \in \mathbb{N} : s_i \to s_{i+1}$ gilt. Maximal bedeutet, dass $\pi$ nicht verlängert werden kann.

Bemerkung: Ist $\mathcal{T}$ total gdw. jeder Lauf unendlich lang ist.

Wir sehen Transitionssysteme als Abstraktionen von Programmen an. Diese Abstraktionen können nicht-deterministsich sein. Dies erhält man z.B. dadurch, dass man von konkreten Variablenbelegungen abstrahiert und nur den Kontrollfluss eines Programms betrachtet. Läufe sind dann Abstraktionen von deterministischen Abläufen eines Programms, und die Transitionsrelation modelliert das Fortschreiten eines Programms in diskreten Zeitabständen.

In dieser Vorlesung betrachten wir nur noch Logiken, die über Transitionssystemen (*branching time*) oder Läufen (*linear time*) interpretiert werden.

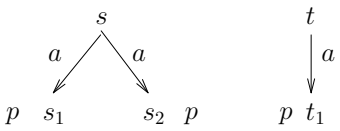
## 1.4. Äquivalenzrelationen auf Transitionssystemen

Eine Menge $\Phi$ von Formeln einer Logik $L$ induziert in natürlicher Weise einen Äquivalenzbegriff $\equiv_\Phi$ auf der Klasse der Strukturen, über der sie interpretiert wird:

$$\mathcal{I} \equiv_\Phi \mathcal{I}' \quad \text{gdw.} \quad \text{für alle } \varphi \in \Phi : \mathcal{I} \models \varphi \Leftrightarrow \mathcal{I}' \models \varphi$$

Dies ist bei der Suche nach geeigneten Logiken zu berücksichtigen. Denn je nach Verwendungszweck (bei temporalen Logiken also z.B. Programmverifikation) soll die Logik nicht in der Lage sein, bestimmte Strukturen zu unterscheiden. So ist z.B. Prädikatenlogik i.A. zu stark, wie folgendes Beispiel zeigt.

**Beispiel 1.4**
Die beiden Zustände $s$ und $t$ aus dem Transitionssystem $\mathcal{T}$




werden von der prädikatenlogischen Formel $\varphi(z) := \exists x.\exists y.x \neq y \wedge p(x) \wedge p(y) \wedge z \xrightarrow{a} x \wedge z \xrightarrow{a} y$ unterschieden. Es gilt: $\mathcal{T}, s \models \varphi(z)$ aber $\mathcal{T}, t \not\models \varphi(z)$. Als Programme "verhalten" sie sich jedoch gleich.

Im folgenden werden wir außer den bekannten Begriffen $=$ (Gleichheit) und $\simeq$ (Isomorphie) noch weitere Äquivalenzbegriffe auf Transitionssystemen definieren.

**Definition 1.8**
Sei $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \lambda)$. Eine *Bisimulation* ist eine Relation $B \subseteq \mathcal{S} \times \mathcal{S}$, so dass für alle $(s,t) \in B$ und alle $a \in \Sigma$ gilt:

1. $\lambda(s) = \lambda(t)$,
2. falls es ein $s'$ gibt mit $s \xrightarrow{a} s'$, dann gibt es auch ein $t'$ mit $t \xrightarrow{a} t'$ und $(s', t') \in B$,
3. falls es ein $t'$ gibt mit $t \xrightarrow{a} t'$, dann gibt es auch ein $s'$ mit $s \xrightarrow{a} s'$ und $(s', t') \in B$,

Gilt nur (1) und (2), dann heisst B *Simulation*.

Zwei Zustände $s, t \in \mathcal{S}$ heissen *bisimilar*, $s \sim t$, wenn es eine Bisimulation $B$ gibt mit $(s,t) \in B$.

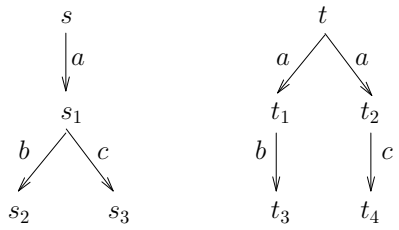
Der Zustand $t$ simuliert $s$, $s \lesssim t$, wenn es eine Simulation $B$ gibt mit $(s,t) \in B$.

Die Zustände $s$ und $t$ heissen *simulationsäquivalent*, $s \approx t$, wenn $s \lesssim t$ und $t \lesssim s$ gelten.

Auf totalen und knotenbeschrifteten Transitionssystemen betrachten wir auch noch *Traceäquivalenz*: $s \leq_{tr} t$, falls für alle Läufe $\pi = s_0, s_1, ...$ mit $s = s_0$ es einen Lauf $t_0, t_1, ...$ mit $t = t_0$ gibt, so dass fuer alle $i \in \mathbb{N}$ gilt: $\lambda(s_i) = \lambda(t_i)$. Zwei Zustände sind traceäquivalent, $s \equiv_{tr} t$, falls $s \leq_{tr} t$ und $t \leq_{tr} s$ gilt.

Alle Äquivalenzen lassen sich in einfacher Weise auf Paare von Zuständen verschiedener Transitionssysteme erweitern. In diesem Fall schreiben wir z.B. $\mathcal{T}, s \sim \mathcal{T}', s'$.

**Beispiel 1.5**
Wir betrachten das Standardbeispiel zweier Getränkeautomaten, die jeweils zwei Getränke zum Preis von 1 € anbieten. Diese werden durch die folgenden beiden Transitionssysteme modelliert. Aktion $a$ bedeutet Eingabe der 1-€-Münze, $b$ bedeutet Ausgabe eines Kaffees, $c$ bedeutet Ausgabe eines Tees.




Offensichtlich modellieren diese beiden Transitionssysteme unterschiedliches Verhalten. Im linken Fall kann man das Getränk nach Bezahlung wählen, im rechten Fall geschieht dies bereits vor der Bezahlung. Beachte, dass beide $s$ und $t$ traceäquivalent, jedoch nicht bisimilar sind.

**Definition 1.9**
Sei $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \lambda, s)$ ein Transitionssystem. Die *Baumabwicklung* von $\mathcal{T}$ bzgl. $s$ ist das Transitionssystem $\mathcal{R}_{\mathcal{T}}(s) = (\mathcal{S}^+, \{\xrightarrow{a} \mid a \in \Sigma\}, \lambda, s)$ mit

- $\mathcal{S}^+$ ist die Menge aller nicht-leeren Wörter über dem Alphabet $\mathcal{S}$,
- $s_0 \dots s_n \xrightarrow{a} s_0 \dots s_n s_{n+1}$ gdw. $s_n \xrightarrow{a} s_{n+1}$,
- $\lambda(s_0 \dots s_n) := \lambda(s_n)$.

**Lemma 1.2**
Für alle Transitionssysteme $\mathcal{T}$ und alle Zustände $s$ gilt: $\mathcal{T}, s \sim \mathcal{R}_{\mathcal{T}}(s), s$.

**Beweis** Übung. ∎

**Lemma 1.3**
Für alle Transitionssysteme gelten folgende Zusammenhänge zwischen Gleichheit, Isomorphie, Bisimilarität, Similarität, Simulationsäquivalenz und Traceäquivalenz:

(a) $= \subsetneq \simeq$,

(b) $\simeq \subsetneq \sim$,

(c) $\sim \subsetneq \approx$,

(d) $\approx \subsetneq \equiv_{tr}$,

(e) $\approx \subsetneq \lesssim$.

**Beweis** Teil (a) ist trivial. Bei den anderen Teilen müssen wir jeweils Inklusion und Striktheit zeigen.

**(b)** Sei $s \simeq t$. Konstruiere eine Relation $B$ folgendermaßen.

1. $(s,t) \in B$,
2. Für alle $(s',t') \in B$ und alle $a \in \Sigma$: falls es ein $s''$ gibt mit $s' \xrightarrow{a} s''$ dann existiert wegen Isomorphie ein $t''$ mit $t' \xrightarrow{a} t''$ und $s'' \simeq t''$. Füge $(s'',t'')$ zu $B$ hinzu.
3. Für alle $(s',t') \in B$ und alle $a \in \Sigma$: falls es ein $t''$ gibt mit $t' \xrightarrow{a} t''$ dann existiert wegen Isomorphie ein $s''$ mit $s' \xrightarrow{a} s''$ und $s'' \simeq t''$. Füge $(s'',t'')$ zu $B$ hinzu.

Wegen Isomorphie gilt auch für alle $(s',t') \in B$: $\lambda(s') = \lambda(t')$. Man sieht leicht, dass $B$ auch die Punkte (2) und (3) aus Def. 1.8 erfüllt.

Es bleibt zu zeigen, dass nicht jedes bisimilare Paar von Zuständen isomorph ist. Betrachte dazu Bsp. 1.4. Es gilt $s \sim t$ mit der Bisimulation $B = \{(s,t),(s_1,t_1),(s_2,t_2)\}$, aber offensichtlich nicht $s \simeq t$.

**(c)** Sei $s \sim t$. Also existiert eine Bisimulation $B$ mit $(s,t) \in B$. Diese erfüllt die Punkte (1)–(3) aus Def. 1.8. Da eine Simulation weniger erfüllen muss als eine Bisimulation ist jede Bisimulation auch eine Simulation. Also $s \lesssim t$. Ausserdem ist leicht zu sehen: $B$ ist Bisimulation gdw. $B' := \{(t,s) \mid (s,t) \in B\}$ eine Bisimulation ist. Also auch $t \lesssim s$ und somit $s \approx t$.

Betrachte jetzt die beiden Zustände $s$ und $t$ in dem folgenden Transitionssystem.


Es gilt $s \lesssim t$ mit der Simulation $\{(s,t),(s_1,t)\}$ und auch $t \lesssim s$ mit Simulation $\{(t,s),(t,s_1),(t_1,s_1)\}$ und somit $s \approx t$. Aber es gilt nicht $s \sim t$.

**(d)** Sei $s \approx t$, also existieren Simulationen $B, B'$ mit $(s,t) \in B$, $(t,s) \in B'$. Sein nun $s_0, s_1, ...$ ein Lauf mit $s_0 = s$. Da $(s,t) \in B$ existiert $t_0$ (nämlich $t$) mit $\lambda(s_0) = \lambda(t_0)$. Da $s_0 \to s_1$ existiert auch ein $t_1$ mit $t_0 \to t_1$ und $(s_1,t_1) \in B$. Also auch $\lambda(s_1) = \lambda(t_1)$. Dies lässt sich ad infinitum zu einem Lauf $t_0, t_1, ...$ fortsetzen. Ausserdem lässt sich zu jedem solchen Lauf mithilfe von $B'$ auch wieder ein Lauf ausgehend aus $s$ konstruieren.

Dass Traceäquivalenz echt schwächer ist als Simulationsäquivalenz wird in Bsp. 1.5 gezeigt.

**(e)** Der Inklusionsteil ist trivial. Zur Striktheit betrachte wiederum Bsp. 1.5. Es gilt: $s \lesssim t$, aber $t \not\lesssim s$ und somit $s \not\approx t$. ∎

Aus diesem Lemma und den obigen Beispielen ziehen wir den Schluss, dass Bisimilarität eine erstrebenswerte Äquivalenz ist. Isomorphie ist zu stark, denn sie unterscheidet Programme, die sich gleich "verhalten". Dagegen ist Simulationsäquivalenz zu schwach,

denn sie kann gewisse Programme nicht unterscheiden, die sich nicht gleich verhalten: Zustände $s$ und $t$ aus Teil (c) des obigen Beweises abstrahieren das Verhalten der beiden Programmfragmente

$$P \ ; \ \texttt{REPEAT} \ \{ \ P \ \} \qquad\qquad \texttt{WHILE} \ b \ \{ \ P \ \} \ ; \ P$$

## 1.5. Bisimulationsspiele

**Definition 1.10**
Set $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, \lambda, s_0)$ ein Transitionssystem und $s, t \in \mathcal{S}$. Das Bisimulationsspiel $\mathcal{G}_{\mathcal{T}}(s,t)$ wird gespielt zwischen Spielern I und II auf dem Spielbrett $\mathcal{S} \times \mathcal{S}$. Spieler I möchte zeigen, dass $s \not\sim t$ gilt, Spieler II möchte zeigen, dass $s \sim t$ gilt.

Jede Partie von $\mathcal{G}_{\mathcal{T}}(s,t)$ startet in der Konfiguration $(s,t)$. Eine Runde läuft wie folgt ab. Sei $(s',t')$ die aktuelle Konfiguration.

1. Spieler I wählt entweder (i) eine Transition $s' \xrightarrow{a} s''$ für ein beliebiges $a$ oder (ii) eine Transition $t' \xrightarrow{a} t''$.
2. Spieler II antwortet im Fall (i) mit einer Transition $t' \xrightarrow{a} t''$, und im Fall (ii) mit einer Transition $s' \xrightarrow{a} s''$.
3. Die nächste aktuelle Konfiguration ist $(s'', t'')$.

Eine Partie endet, wenn

1. die aktuelle Konfiguration $(s',t')$ ist und $\lambda(s') \neq \lambda(t')$ gilt, oder
2. Spieler II nicht mit einer entsprechenden Transition antworten kann.

Spieler II gewinnt jede Partie, die unendlich lang ist. Spieler I gewinnt jede endliche Partie.

**Satz 1.1**
Spieler II hat eine Gewinnstrategie für das Spiel $\mathcal{G}_{\mathcal{T}}(s,t)$ gdw. $s \sim t$.

**Beweis** ($\Leftarrow$) Angenommen $s \sim t$, d.h. es gibt eine Bisimulation $B$ mit $(s,t) \in B$. Für Spieler I gilt jetzt folgende Invariante für jede Partie: Ist die aktuelle Position $(s',t')$, dann gilt $\lambda(s') = \lambda(t')$, und

- wenn er eine Transition $s' \xrightarrow{a} s''$ wählt, dann existiert auch eine Transition $t' \xrightarrow{a} t''$, so dass $(s'', t'') \in B$.
- wenn er eine Transition $t' \xrightarrow{a} t''$ wählt, dann existiert auch eine Transition $s' \xrightarrow{a} s''$, so dass $(s'', t'') \in B$.

Dies liefert eine Strategie für Spieler II. Sie wählt einfach die entsprechende Transition, die nach Voraussetzung immer existiert. Da sie immer existiert, ist jede Partie, in der sich Spieler II an diese Strategie hält, eine Gewinnpartie für sie. Daher ist diese Strategie eine Gewinnstrategie.

($\Rightarrow$) Angenommen, Spieler II hat eine Gewinnstrategie $\zeta$ für das Spiel $\mathcal{G}_{\mathcal{T}}(s,t)$. Wir benutzen diese, um eine Bisimulation $B$ induktiv zu konstruieren. Dazu definieren wir Mengen $B_i \subseteq \mathcal{S} \times \mathcal{S}$, so dass für alle $i \in \mathbb{N}$ und alle $(s',t') \in B_i$ gilt: (*) Spieler II hat eine Gewinnstrategie für das Spiel $\mathcal{G}_{\mathcal{T}}(s',t')$.

Setze $B_0 := \{(s,t)\}$. Offensichtlich gilt (*) für alle $(s',t') \in B_0$. Sei nun $B_i$ für ein beliebiges $i \in \mathbb{N}$ bereits konstruiert. Nach Voraussetzung hat Spieler II eine Gewinnstrategie für das Spiel $\mathcal{G}_{\mathcal{T}}(s',t')$ für alle $(s',t') \in B_i$. Betrachte nun die erste Runde in solch einem Spiel ausgehend von $(s',t')$. Offensichtlich gilt $\lambda(s') = \lambda(t')$, denn sonst gäbe es keine erste Runde.

Angenommen Spieler I wählt eine Transition $s' \xrightarrow{a} s''$. Dann muss Spieler II darauf mit einem Zug $t' \xrightarrow{a} t''$ reagieren können. Dasselbe gilt, falls Spieler I zuerst eine Transition $t' \xrightarrow{a} t''$ wählt. Setze jetzt

$$B_{i+1} := \bigcup_{(s',t') \in B_i} \{(s'',t'') \mid \text{Spieler II reagiert mit } t' \xrightarrow{a} t'' \text{ auf } s' \xrightarrow{a} s'' \text{ oder mit } s' \xrightarrow{a} s'' \text{ auf } t' \xrightarrow{a} t''\}$$

Da für jedes $(s'',t'') \in B_{i+1}$ gilt, dass es ein $(s',t') \in B_i$ gibt, so dass Spieler II auf einen Zug von Spieler I in $\mathcal{G}_{\mathcal{T}}(s',t')$ reagiert und zu $(s'',t'')$ gelangt, hat Spieler II auch eine Gewinnstrategie für das Spiel $\mathcal{G}_{\mathcal{T}}(s'',t'')$.

Setze nun $B := \bigcup_{i \in \mathbb{N}} B_i$. Es bleibt zu zeigen, dass $B$ eine Bisimulation ist. Sei also $(s',t') \in B$. Dann gibt es ein $i \in \mathbb{N}$ so dass $(s',t') \in B_i$. Wie oben gezeigt, gilt $\lambda(s') = \lambda(t')$. Falls es nun ein $a \in \Sigma$ und ein $s'' \in \mathcal{S}$ gibt mit $s' \xrightarrow{a} s''$, dann gibt es auch eine Transition $t' \xrightarrow{a} t''$ und $(s'',t'') \in B_{i+1}$. Genauso umgekehrt. Da $B_{i+1} \subseteq B$, gilt also auch in beiden Fällen $(s'',t'') \in B$. Somit ist $B$ eine Bisimulation, und es gilt $(s,t) \in B$. Also $s \sim t$. ∎

Satz 1.1 liefert ein mächtiges Werkzeug, um Nichtbisimilarität zwischen $s$ und $t$ zu zeigen. Anstatt die universelle Aussage, dass alle Relationen entweder keine Bisimulation sind oder $(s,t)$ nicht enthalten, zu beweisen, reicht es aus, eine Gewinnstrategie für Spieler I in dem Spiel $\mathcal{G}_{\mathcal{T}}(s,t)$ anzugeben. Denn offensichtlich kann Spieler II keine Gewinnstrategie haben, wenn Spieler I eine hat. Aus Satz 1.1 folgt dann, dass $s \not\sim t$ gilt. Eine Gewinnstrategie für Spieler I anzugeben, ist jedoch das Beweisen einer existentiellen Aussage.

**Beispiel 1.6**
Betrachte noch einmal das Transitionssystem $\mathcal{T}$ und die beiden Zustände $s$ und $t$ aus Bsp. 1.5. Es ist leicht, eine Gewinnstrategie für Spieler I in dem Spiel $\mathcal{G}_{\mathcal{T}}(s,t)$ anzugeben. Wähle zuerst die Transition $s \xrightarrow{a} s_1$. Darauf muss Spieler II entweder mit $t \xrightarrow{a} t_1$ oder $t \xrightarrow{a} t_2$ antworten. Im ersten Fall wähle dann die Transition $s_1 \xrightarrow{c} s_3$, im zweiten Fall wähle dann die Transition $s \xrightarrow{b} s_2$. In beiden Fällen kann Spieler II nicht mehr antworten und verliert.