Software Product Lines

Concepts, Analysis and Implementation

Verifikation von Produktlinien

Dr. Malte Lochau
Malte.Lochau@es.tu-darmstadt.de

Inhalt

I. Einführung

- Motivation und Grundlagen
- Feature-orientierte Produktlinien

II. Produktlinien-Engineering

- Feature-Modelle und Produktkonfiguration
- Variabilitätsmodellierung im Lösungsraum
- Programmierparadigmen für Produktlinien

III. Produktlinien-Analyse

- Feature-Interaktion
- Testen von Produktlinien
- Verifikation von Produktlinien

IV. Fallbeispiele und aktuelle Forschungsthemen

- Grundlagen Model-Checking
- Family-based SPL Model-Checking
- Incremental SPL Model-Checking

Beispiel: BCS

(Informelle) Anforderungen

[…] Wenn das Fenster die höchste/tiefste Position erreicht, muss der Lauf stoppen

Transitionssysteme: Abstrakte Repräsentation des Systemverhaltens mit formaler Semantik

Modell

Verifikation

$\models AG\ \neg(pw\_isUp \wedge pw\_mv\_up)$

CTL: Temporallogik zur Spezifikation von Korrektheitseigenschaften

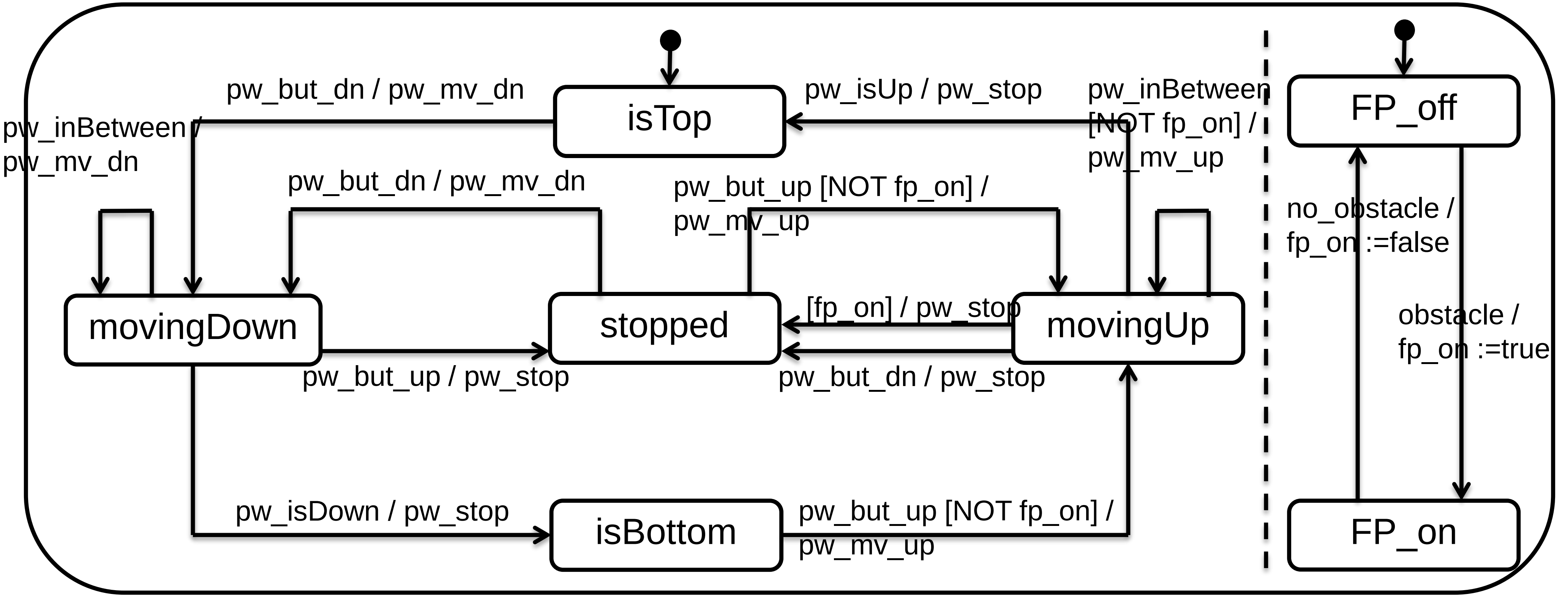
Testen

$(pw\_but\_up,\ pw\_isUp,\ pw\_but\_Up) \Rightarrow$ Implementierung $\Rightarrow (pw\_mv\_up,\ pw\_stop)$

(Testfall = experimentelle Testeingaben + erwartete Testausgaben)

Verifikation von Modelleigenschaften

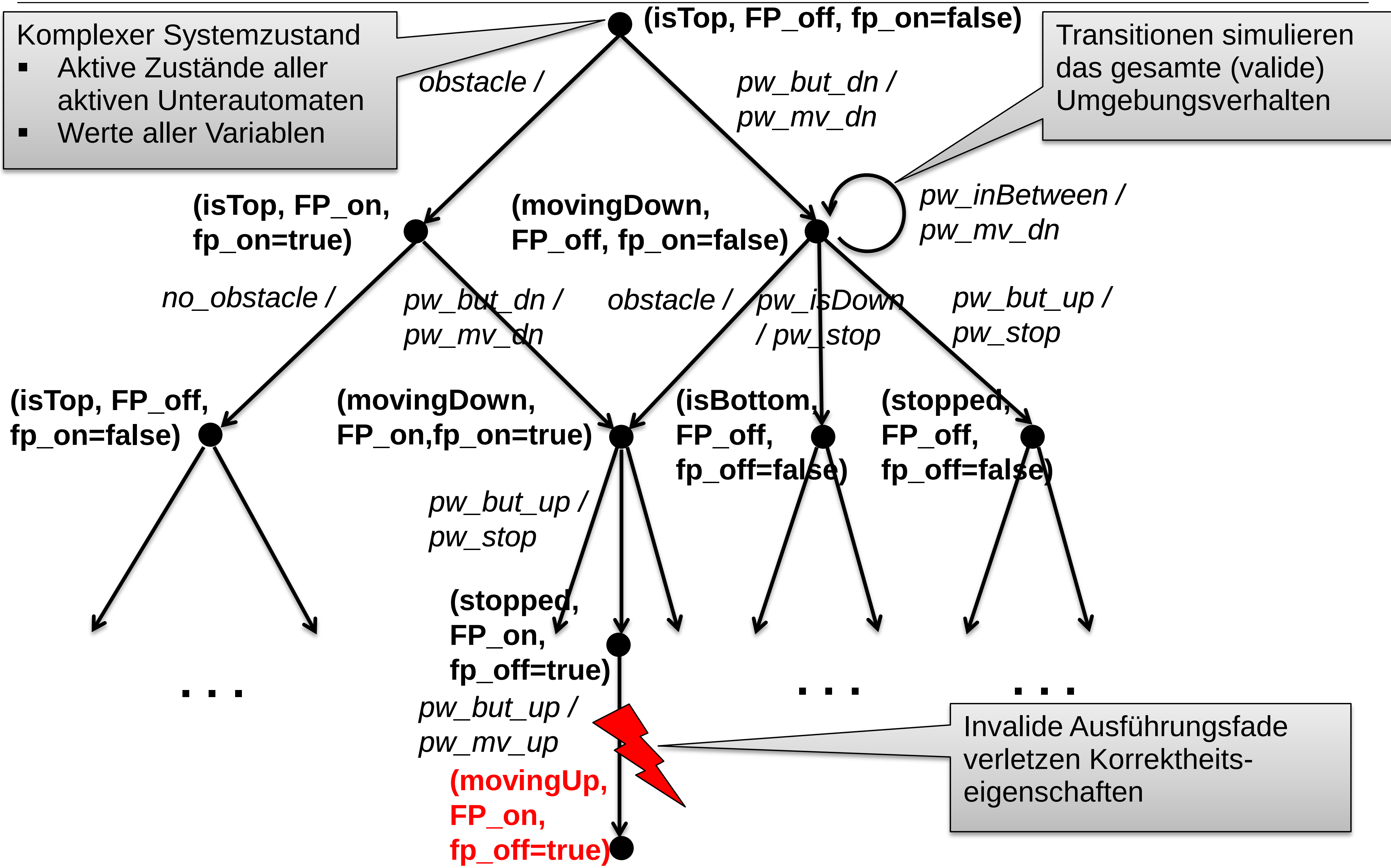
p = { PW, AutPw, FP }



- …
- […] Wenn ein Hindernis während des Hochlaufs erkannt wird, soll der Hochlauf stoppen
- […] Nachdem ein Hindernis entfernt wurde, soll der Fensterlauf fortgesetzt werden können
- …

Eigenschaften auf allen möglichen Abläufen erfüllt?

Transitionssysteme als allgemeines semantisches Modell



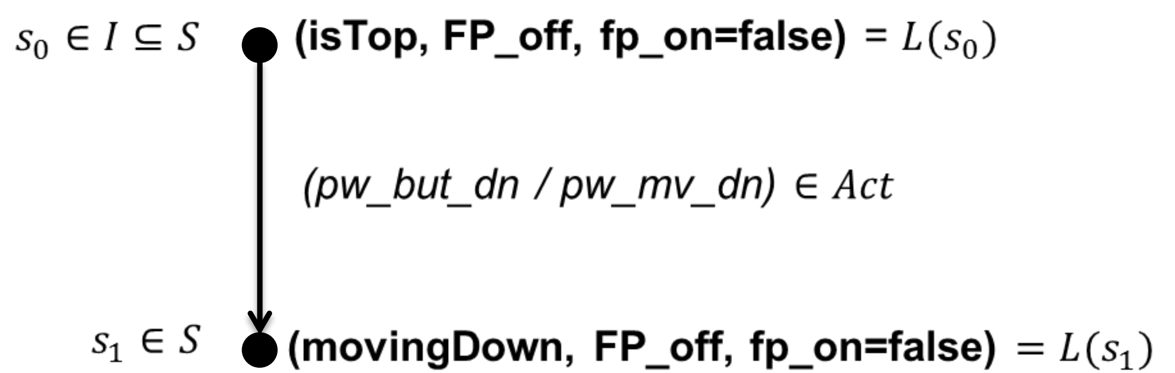
Transitionssysteme – Abstrakte Syntax

Ein **Transitionssystem** ist ein Tupel $ts = (S, Act, \to, I, P, L)$ mit

- einer abzählbaren Menge S (Zustände)
- einer abzählbaren Menge Act (Aktionen)
- einer Relation $\to \subseteq S \times act \times S$ (beschriftete Transitionen)
- einer endlichen Menge $I \subseteq S$ (Initialzustände)
- einer abzählbare Menge P (Zustandsprädikate)
- einer Funktion $L : S \to 2^P$ (Zustandsbeschriftung)

Notation:

- $\boldsymbol{s} \xrightarrow{a} \boldsymbol{s}'$ falls $(s, a, s') \in \to$



Transitionssysteme – Semantik

- Eine **Ausführung** (**Verhalten**) eines Transitionssystems ts ist eine Sequenz

$$\sigma = s_0 a_1 s_1 a_2 s_2 \ldots$$

mit $s_0 \in I$ und $s_i \overset{a_{i+1}}{\rightarrow} s_{i+1}$.

- Ein **Pfad** π einer Ausführung σ eines Transitionsystems ts ist die Einschränkung von σ auf die Sequenz der Zustände:

$$\pi = s_0\ s_1\ s_2.$$

- Die **Semantik** $[\![ts]\!]_{TS} \subseteq S^*$ eines Transitionsystems ts ist gegeben durch die Menge der Pfade von ts.

Spezifikation von Korrektheitseigenschaften

- Korrektheitseigenschaften müssen durch gesamte Menge aller Pfade des Transitionssystems erfüllt sein
- Korrektheitseigenschaften beziehen sich auf die (Abfolge von) Zustandsprädikaten von allen über Pfade erreichbaren Zustände

- **Sicherheitseigenschaften (Safety):** Eigenschaften, die für alle erreichbaren Zustände gelten müssen.

> […] Wenn ein Hindernis während des Hochlaufs erkannt wird, soll der Hochlauf stoppen

- **Lebendigkeitseigenschaften (Liveness):** Eigenschaften, die für Abfolgen von Zuständen gelten müssen.

> […] Nachdem ein Hindernis entfernt wurde, soll der Fensterlauf fortgesetzt werden können

Sicherheits- und Korrektheitseigenschaften können durch Temporallogiken **spezifiziert** und auf einem Transitionssystem **formal verifiziert** werden.

Temporallogiken

- Mit Temporallogiken können Aussagen über Abläufe über die Zeit spezifiziert und automatisiert auf Modellen beweisen werden
(Modell ~ Transitionssystem, Ablauf ~ Pfad)

- Diskreter Zeitbegriff (Zeitpunkt ~ Zustand)

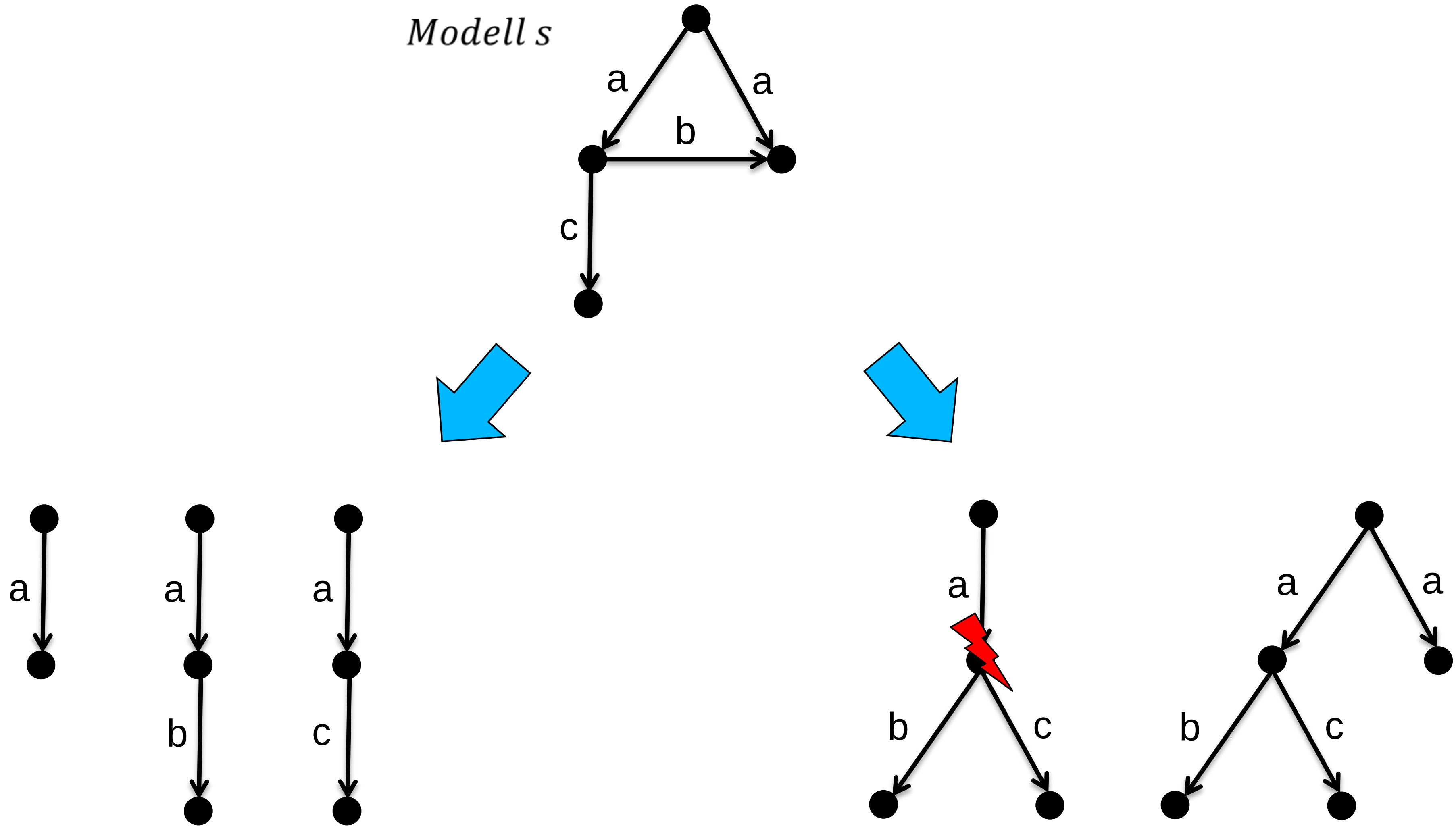
- Dynamischer Wahrheitsbegriff: In verschiedenen Zuständen eines Modells können Aussagen unterschiedliche Wahrheitswerte annehmen

- Linear Time Logic (LTL) vs. Computation Tree Logic (CTL)

- Model-Checking: Automatisierte Prüfung, ob eine temporallogische Aussage ϕ (Korrektheitseigenschaft) durch ein Modell s (Systemspezifikation) erfüllt ist

  Notation: $s \vDash \phi$

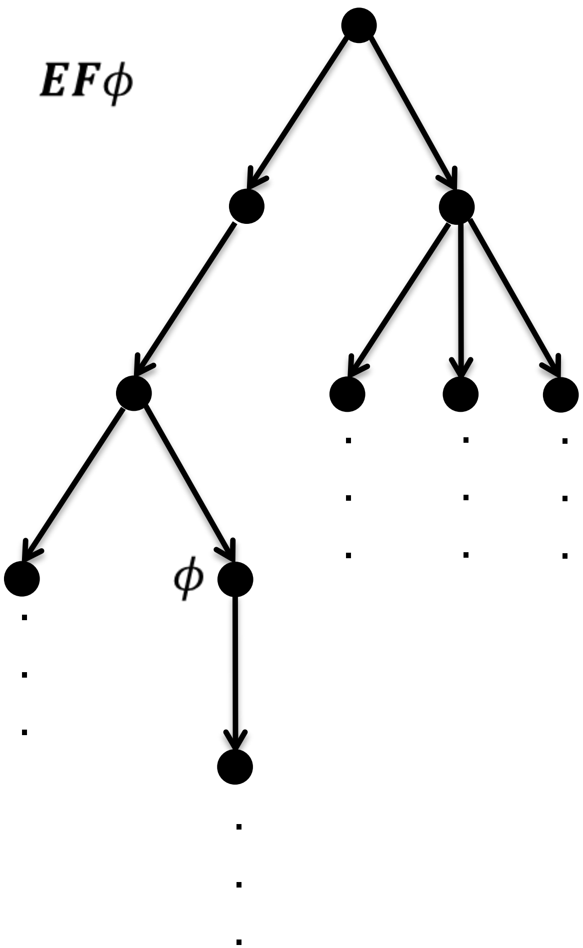
Lineare Zeit vs. Verzweigte Zeit



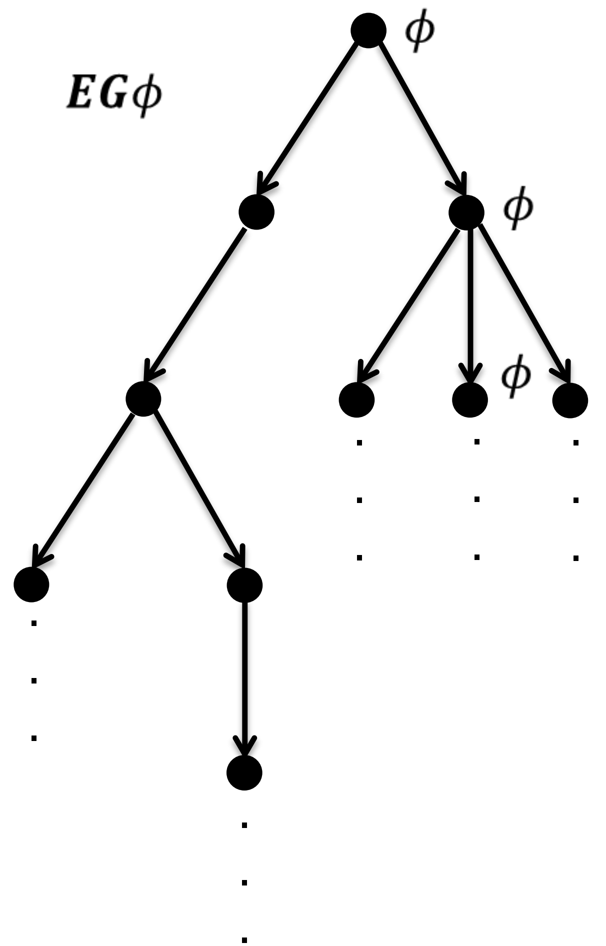
Linear Time
(Ausführungsmenge)

Branching Time
(Ausführungsbaum)

Beispiele: Aussagen (1/2)

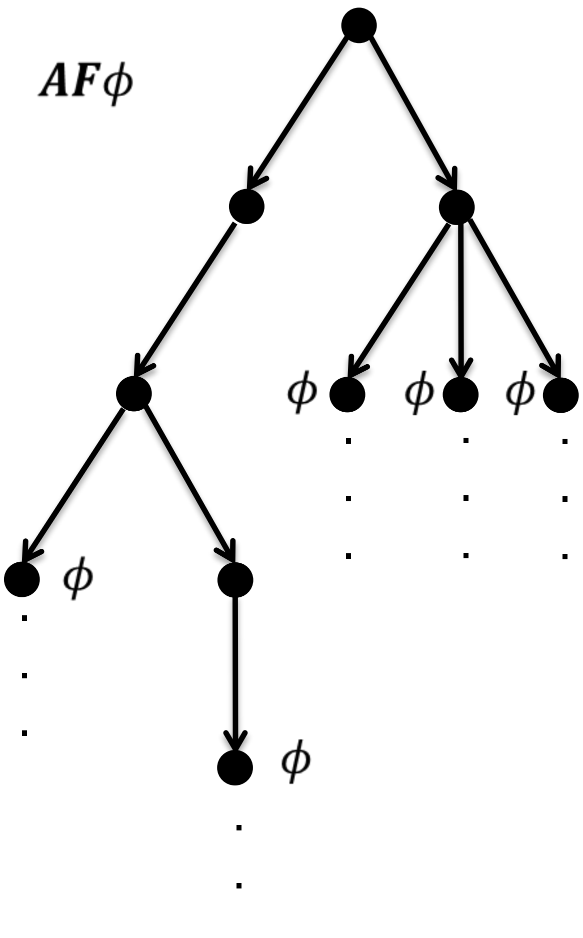


„Auf irgendeinem Pfad gilt irgendwann mal ϕ"

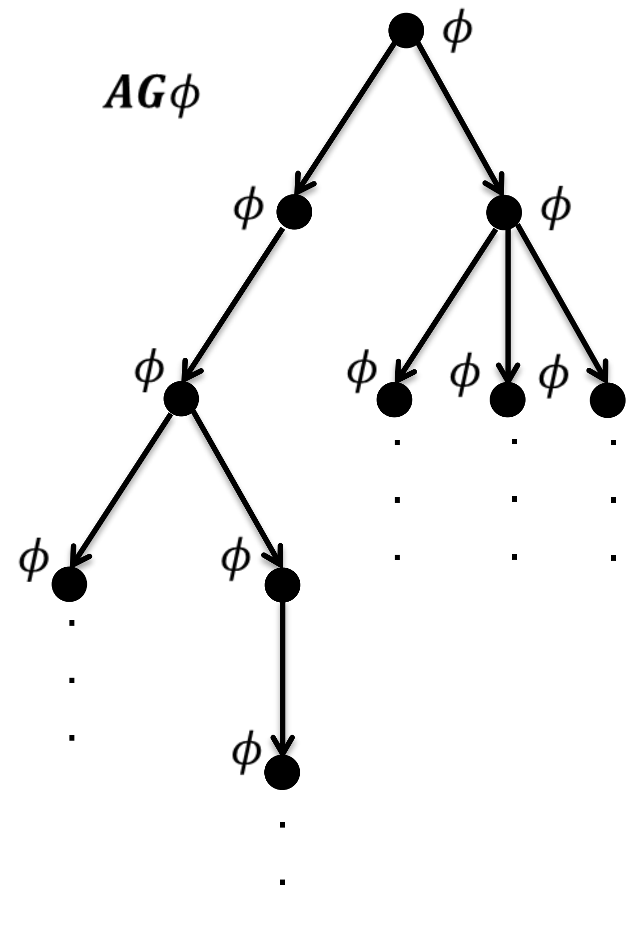


„Auf irgendeinem Pfad gilt immer ϕ"

Beispiele: Aussagen (2/2)



„Auf allen Pfaden gilt irgendwann mal ϕ"



„Auf allen Pfaden gilt immer ϕ"

Temporallogik CTL – Syntax (1/2)

Atomare Zustandsformeln:

- $\top, \bot$ Konstanten *wahr, falsch*
- $p \in P$ Zustandsprädikate

Aussagenlogische Verknüpfungen:

- $\wedge, \vee, \neg$

Temporale Verknüpfungen (Pfad-Quantoren):

- $\boldsymbol{A}$ Always (entlang aller Pfade)
- $\boldsymbol{E}$ Exists (entlang eines Pfades)
- $\boldsymbol{X}$ NeXt (im nächsten Zustand)
- $\boldsymbol{F}$ Future (in einem zukünftigen Zustand)
- $\boldsymbol{G}$ Globally (in allen zukünftigen Zuständen)
- $\boldsymbol{U}$ Until (bis)

Temporallogik CTL – Syntax (2/2)

CTL-Formeln:

$$\phi \quad ::= \quad \top \mid \bot \mid p \mid$$

$$(\neg \phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid$$

$$\boldsymbol{AX}\phi \mid \boldsymbol{EX}\phi \mid \boldsymbol{AG}\phi \mid \boldsymbol{EG}\phi \mid \boldsymbol{AF}\phi \mid \boldsymbol{EF}\phi \mid$$

$$\boldsymbol{A}[\phi\ \boldsymbol{U}\ \phi] \mid \boldsymbol{E}[\phi\ \boldsymbol{U}\ \phi]$$

[…] Wenn ein Hindernis während des Hochlaufs erkannt wird, soll der Hochlauf stoppen

$$\boldsymbol{AG}(\neg(FP\_on \wedge \boldsymbol{EX}\ movingUp))$$

[…] Nachdem ein Hindernis entfernt wurde, soll der Fensterlauf fortgesetzt werden können

$$\boldsymbol{AG}(FP\_off \Rightarrow \boldsymbol{AF}\ movingUp))$$

Temporallogik CTL – Semantik

Die **Auswertung** einer CTL-Formel ϕ auf einem Transitionssystem ts erfolgt durch die Betrachtung der Zustände $s \in S$ durch folgende Regeln:

- $s \vDash \top$ und $s \nvDash \bot$ für alle $s \in S$
- $s \vDash p$ genau dann, wenn $p \in \mathrm{L}(s)$
- $s \vDash \neg\phi$ genau dann, wenn $s \nvDash \phi$
- $s \vDash \phi_1 \wedge \phi_2$ genau dann, wenn $s \vDash \phi_1$ und $s \vDash \phi_2$ ($\vee$ analog)
- $s \vDash \boldsymbol{AX}\phi$ genau dann, wenn $s' \vDash \phi$ für alle $s' \in S$ mit $s \xrightarrow{a} s'$ gilt ($\boldsymbol{EX}$: es gibt ein s')
- $s \vDash \boldsymbol{AG}\phi$ genau dann, wenn alle von s aus über einen Pfad erreichbaren Zustände ϕ erfüllen ($\boldsymbol{EG}$: es gibt einen Pfad, auf dem jeder Zustand ϕ erfüllt)
- $s \vDash \boldsymbol{AF}\phi$ genau dann, wenn auf allen von s ausgehenden Pfaden ein Zustand erreichbar ist, der ϕ erfüllt ($\boldsymbol{EF}$: es gibt einen Pfad, auf dem ein Zustand ϕ erfüllt)
- $s \vDash \boldsymbol{A}[\phi_1 \boldsymbol{U} \phi_2]$ auf allen Pfaden beginnend mit s gilt ϕ_1 solange, bis in einem Zustand ϕ_2 gilt ($\boldsymbol{EU}$: es gibt einen Pfad …)

Eigenschaften von CTL-Formeln

De-Morgan-Regeln:

$$\neg \boldsymbol{AF}\phi \Leftrightarrow \boldsymbol{EG}\neg\phi$$
$$\neg \boldsymbol{EF}\phi \Leftrightarrow \boldsymbol{AG}\neg\phi$$
$$\neg \boldsymbol{AX}\phi \Leftrightarrow \boldsymbol{EX}\neg\phi$$

Fixpunkt-Charakterisierung (*):

$$\boldsymbol{AG}\phi \Leftrightarrow \phi \wedge \boldsymbol{AX}\ \boldsymbol{AG}\phi$$
$$\boldsymbol{EG}\phi \Leftrightarrow \phi \wedge \boldsymbol{EX}\ \boldsymbol{EG}\phi$$
$$\boldsymbol{AF}\phi \Leftrightarrow \phi \vee \boldsymbol{AX}\ \boldsymbol{AF}\phi$$
$$\boldsymbol{EF}\phi \Leftrightarrow \phi \vee \boldsymbol{EX}\ \boldsymbol{EF}\phi$$

(*) Grundlage für die Implementierung eines (rekursiven) CTL-Model-Checkers für Transitionssysteme mit endlicher Zustandsmenge

Häufige Muster für CTL-Spezifikationen

$\boldsymbol{AG}(request \Rightarrow \boldsymbol{AF}\, reply)$

Liveness/Fairness: „Das System liefert auf jede neue Anfrage (*request*) immer wieder irgendwann eine Antwort (*reply*)"

$\boldsymbol{AG}(\boldsymbol{AF}\, ready)$

Progress: „Das System wird immer wieder irgendwann bereit (*ready*) sein"

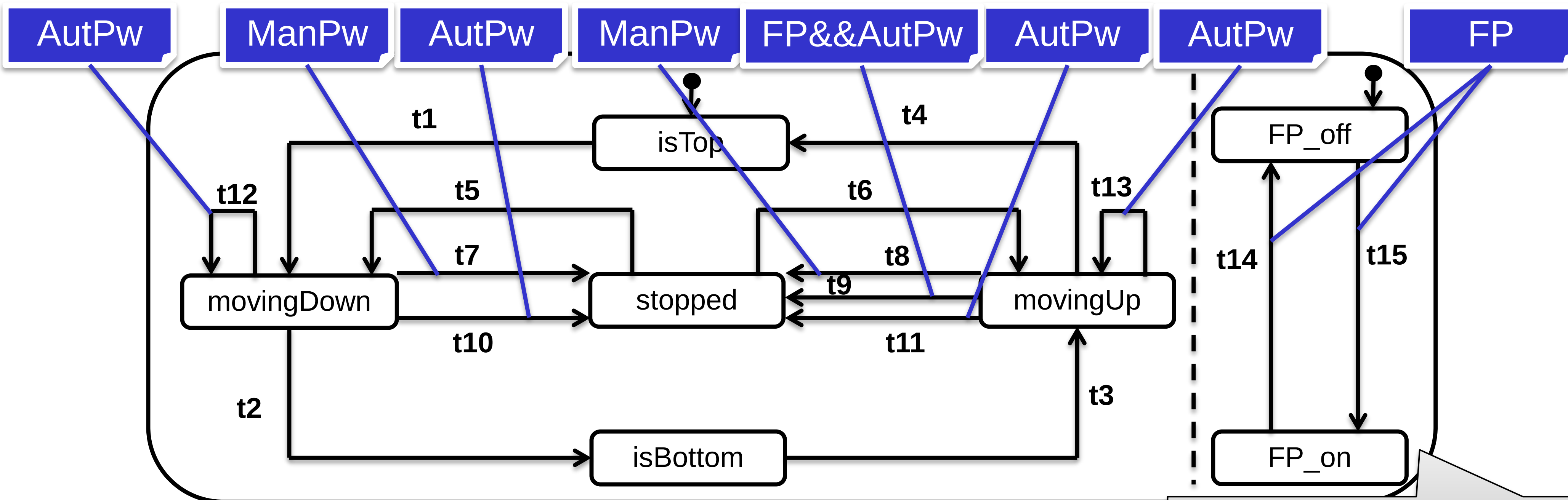
$\boldsymbol{AF}(\boldsymbol{AG}\, stop)$

Termination: „Das System wird immer irgendwann endgültig stoppen (*stop*)"

$\boldsymbol{AF}(\boldsymbol{EF}\, start)$

Resetable: „Das System lässt sich immer in den Startzustand (*start*) zurückversetzen"

Verifikation von SPL Eigenschaften



Konfigurations-spezifische Abläufe

[…] Wenn ein Hindernis während des Hochlaufs erkannt wird, soll der Hochlauf stoppen

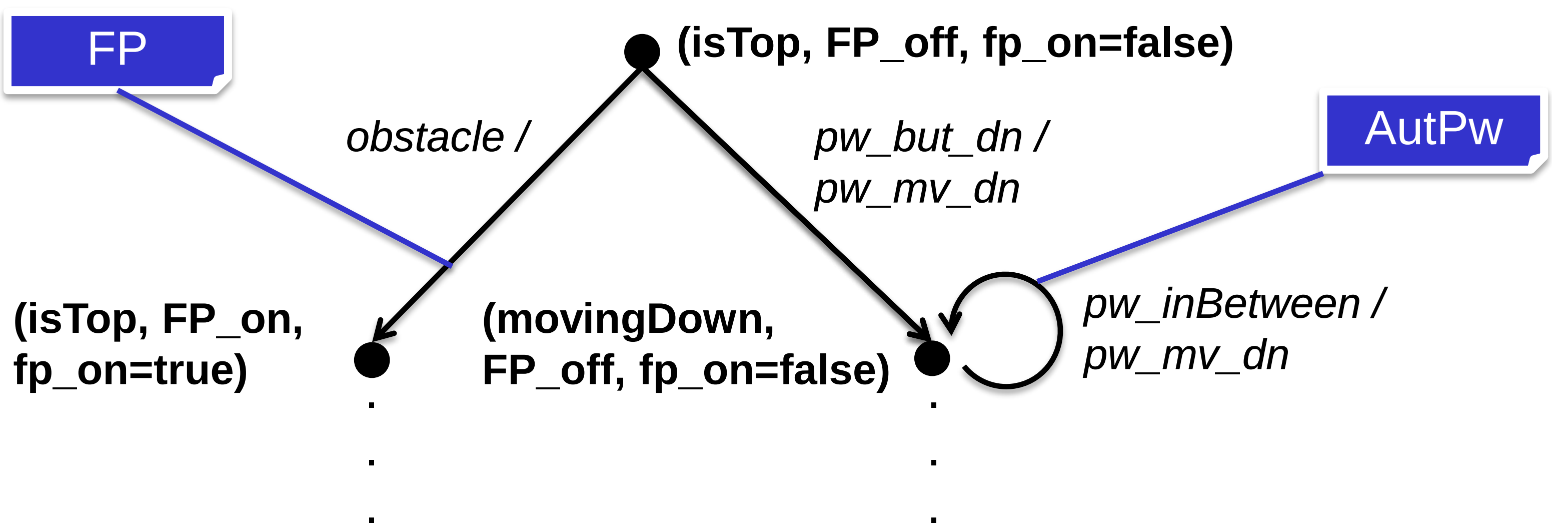
$$AG(\neg(FP\_on \wedge EX\ movingUp))$$

Konfigurations-spezifische Eigenschaften

[…] Nachdem ein Hindernis entfernt wurde, soll der Fensterlauf fortgesetzt werden können

$$AG(FP\_off \Rightarrow AF\ movingUp))$$

Featured Transition Systems (FTS)



FTS = Transitionssystem mit doppelt beschrifteten Transitionen:

- Aktion / Reaktion wie in Transitionssystemen
- Presence Condition

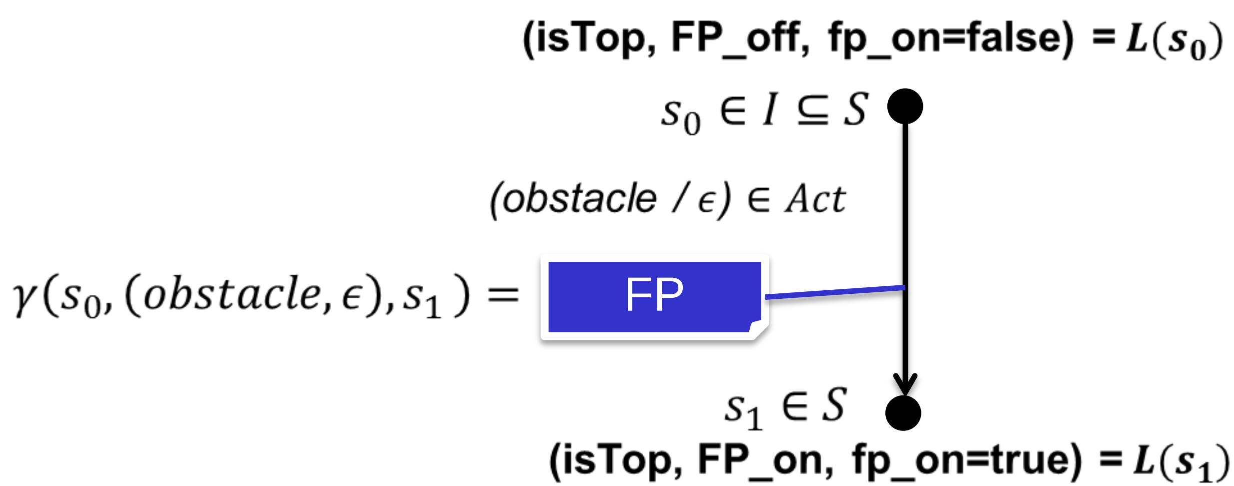
FTS – Syntax

Ein **Feature-Transitionssystem** ist ein Tupel $fts = (S, Act, \rightarrow, I, P, L, \widetilde{FM}, \gamma)$ mit

- einer abzählbaren Menge S (Zustände)
- einer abzählbaren Menge Act (Aktionen)
- einer Relation $\rightarrow \subseteq S \times act \times S$ (beschriftete Transitionen)
- einer endlichen Menge $I \subseteq S$ (Initialzustände)
- einer abzählbare Menge P $\longrightarrow$ (Zustandsprädikate)
- einer Funktion $L : S \rightarrow 2^P$ (Zustandsbeschriftung)
- einer Funktion $\widetilde{FM}: (F \rightarrow \mathcal{B}) \rightarrow \mathcal{B}$ (Feature-Modell)
- einer Funktion $\gamma: (\rightarrow) \rightarrow ((F \rightarrow \mathcal{B}) \rightarrow \mathcal{B})$ (Feature-Annotationen)

Notation:

- $s \xrightarrow{a/\varphi} s'$ falls $\gamma(s, a, s') = \varphi$



FTS – Semantik

- Die **Projektion** eines Feature-Transitionssystems $fts = (S, Act, \rightarrow, I, P, L, \widetilde{FM}, \gamma)$ auf eine Produktkonfiguration $p \in [FM]$ ist ein Transitionssystem $ts_p = (S, Act, \rightarrow_p, I, P, L)$ mit

$$\rightarrow_p = \{ (s, a, s') \in \rightarrow \mid p \vdash \gamma(s, a, s') \}$$

- Die **Semantik einer Produktvariante** mit Konfiguration $p \in [FM]$ ist definiert durch die Semantik der Projektion ts_p

$$[\![fts, p]\!]_{FTS} = [\![ts_p]\!]_{TS}$$

- Die **Semantik einer Produktlinie** ist definiert durch die Semantik der Produktvarianten

$$[\![fts]\!]_{FTS} = \bigcup_{p \in [\widetilde{FM}]} [\![ts_p]\!]_{TS}$$

fCTL

$$
\begin{array}{lll}
\phi & ::= & \top \mid \bot \mid p \mid \\
 & & (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid \\
 & & [\chi]\boldsymbol{AX}\phi \mid [\chi]\boldsymbol{EX}\phi \mid [\chi]\boldsymbol{AG}\phi \mid [\chi]\boldsymbol{EG}\phi \mid [\chi]\boldsymbol{AF}\phi \mid [\chi]\boldsymbol{EF}\phi \mid \\
 & & [\chi]\ \boldsymbol{A}[\phi\ \boldsymbol{U}\ \phi] \mid [\chi]\ \boldsymbol{E}[\phi\ \boldsymbol{U}\ \phi]
\end{array}
$$

- *Product Quantifier* $\chi: (F \rightarrow \mathcal{B}) \rightarrow \mathcal{B}$ als Guard für CTL-Formeln (*)

> […] Wenn ein Hindernis während des Hochlaufs erkannt wird, soll der Hochlauf stoppen

$\Rightarrow$ $[FP]\mathbf{AG}(\neg FP\_on \wedge \boldsymbol{EX} moving\_up)$

(*) Auswertungssemantik vgl. Classen et al. 2011

Family-based SPL Model-Checking

Sei FTS s eine Produktlinienspezifikation und $[\chi]\phi$ eine fCTL-Formel (O.B.d.A. sei der Product Quantifier χ „ganz außen“).

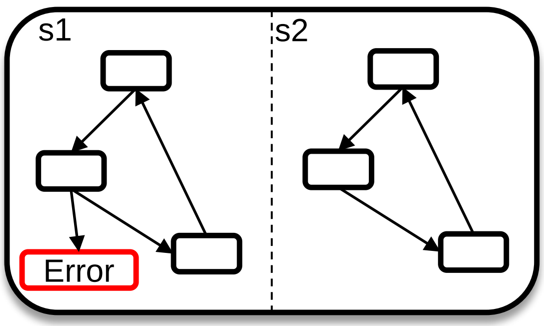
Dann gilt:

$$fts \vDash [\chi]\phi \qquad \Leftrightarrow \qquad \forall p \in [FM]\colon p \vdash \chi \Rightarrow ts_p \vDash \phi$$

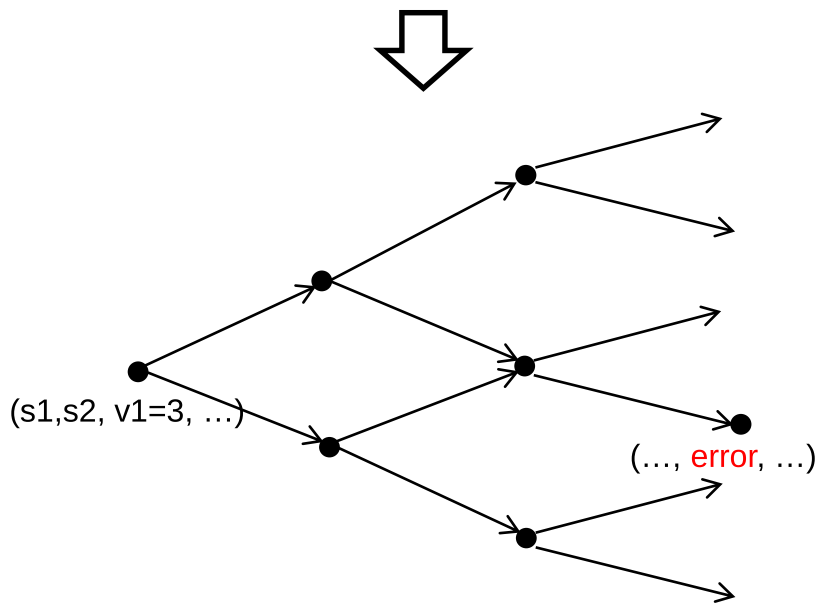
Andere semantische SPL Modelle

- Featured Petri Nets
- Featured Timed Automata
- Featured Weighted Automata
- PL-CCS
- …

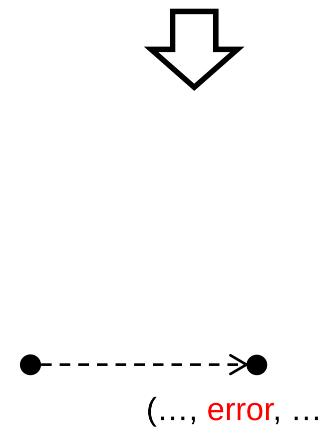
Incremental SPL Model-Checking



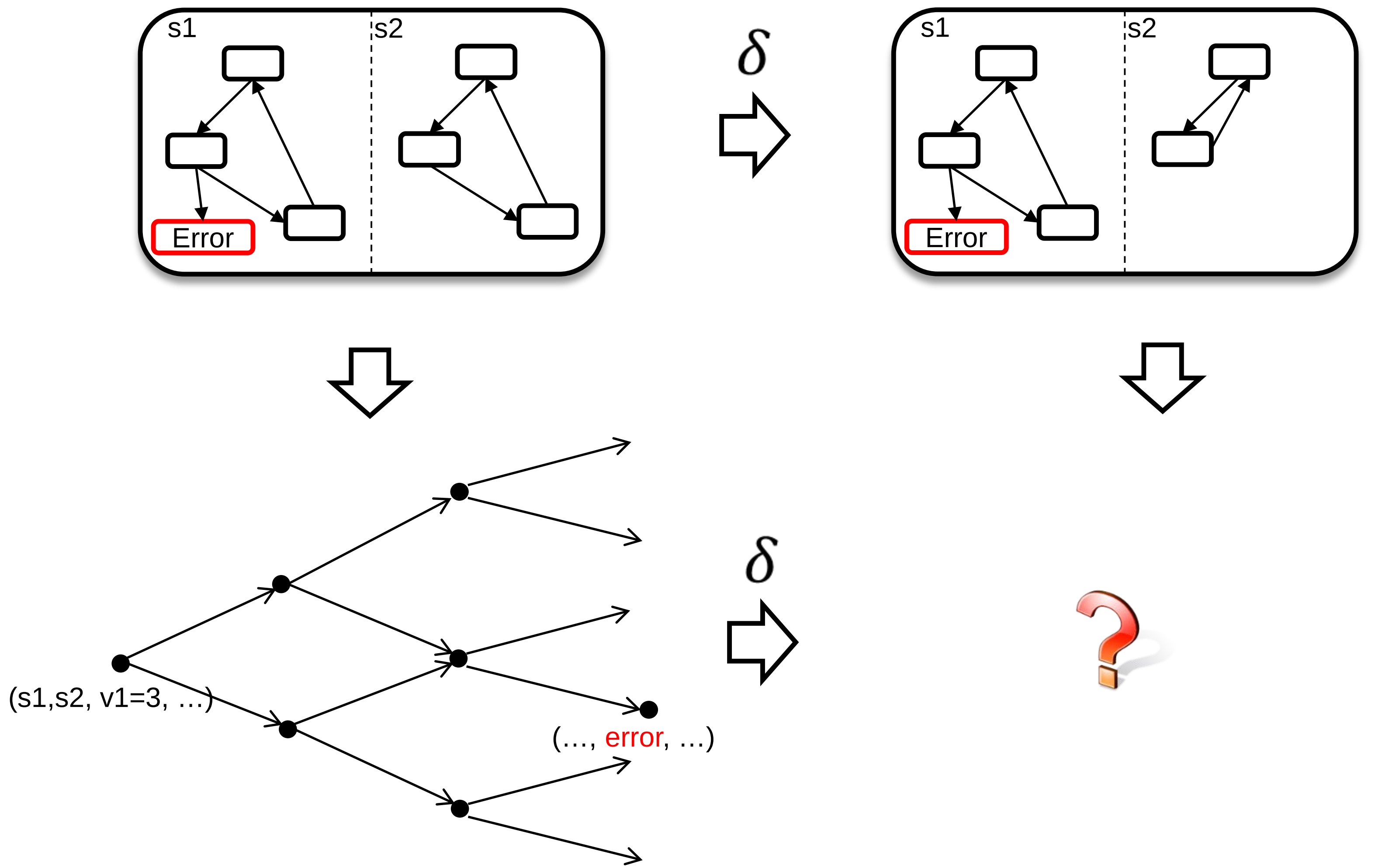
$\models \quad AG(\neg\, Error)$



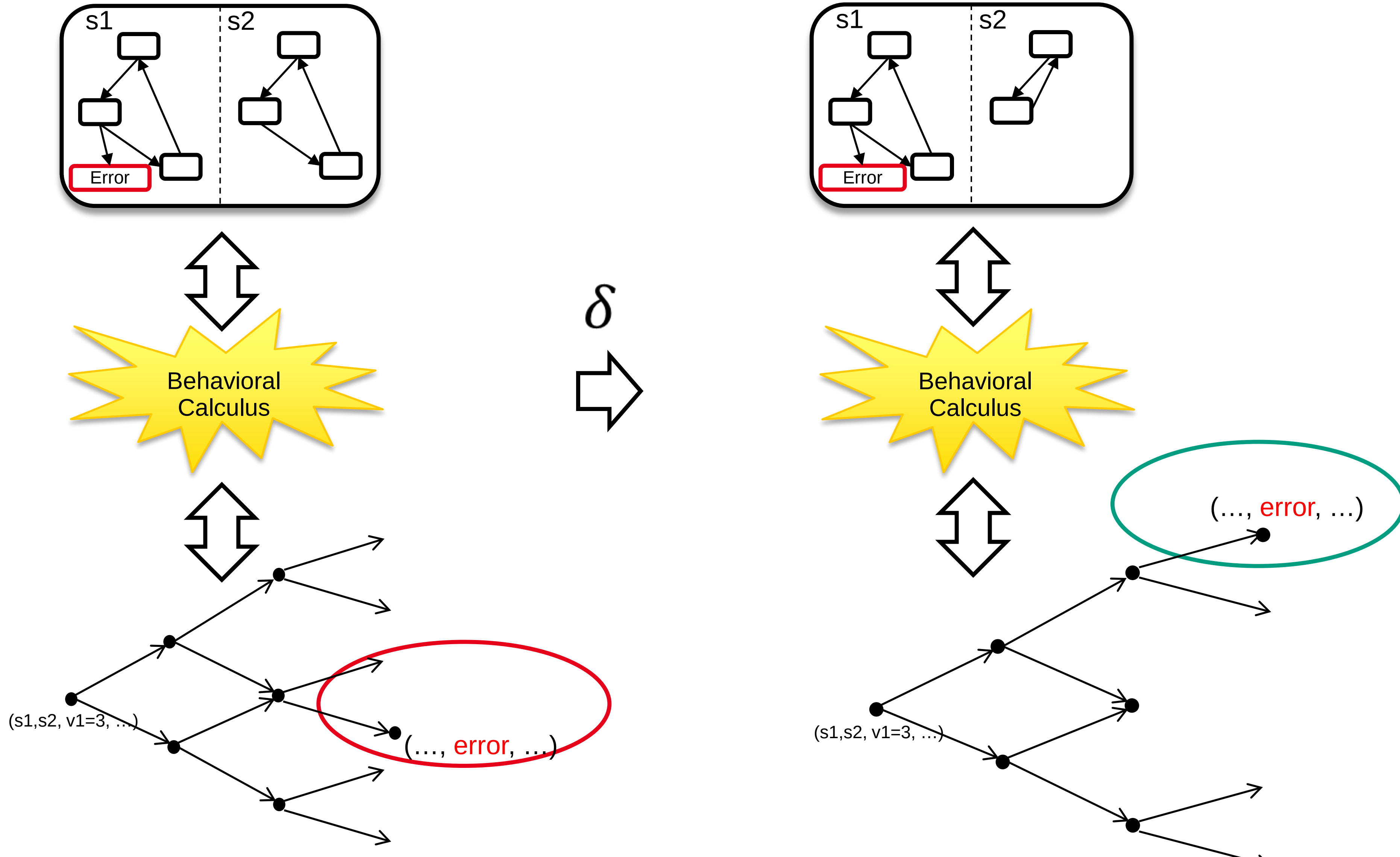
$\cap$



Syntactical vs. Semantical Changes



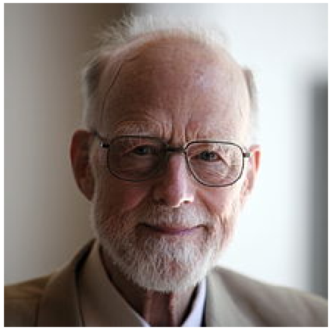
Semantical Models



A Short History of Behavioral Core Calculi

λ-Calculus
(1930)

CCS
(~1980)



CSP
(~1978)

… π-Calculus, Ambient-Calculus,
Join-Calculus (1990s)…

CCS Syntax

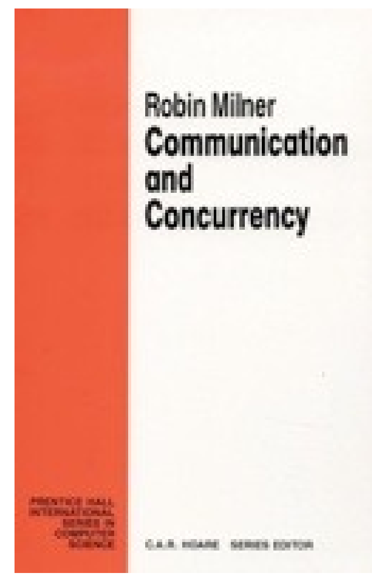
action prefix | (unguarded) choice | composition | relabeling | hiding | recursion

$$P ::= \alpha.P \mid \sum_{i\in I} P_i \mid P \mid P \mid P[f] \mid P \setminus L \mid X$$

- Process Terms $P \in \mathrm{P}(Act, K)$
- Set of Actions $\alpha \in Act = \{a, \bar{a} \mid a \in \aleph\}$, where $\bar{\bar{a}} = a$
- Finite Index set $I \in \mathbb{N}_0$
- Relabeling function $f: Act \to Act$
- Invisible Label Set $L \subseteq \aleph$
- Process Names $X := P \in \mathrm{P}(Act, K)$

Some Syntactic Sugar:

- $\sum_{i\in\{1,2\}} P_i = P_1 + P_2$
- $\sum_{i\in\emptyset} P_i = \overline{\mathbf{0}}$



Structural Operational Semantics of CCS

$$(\text{pre})\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad (\text{rec})\ \frac{P \xrightarrow{\alpha} P' \quad K \overset{Def}{=} P}{K \xrightarrow{\alpha} P'} \qquad (\text{choice})\ \frac{P_j \xrightarrow{\alpha} P'_j \quad j \in I}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j}$$

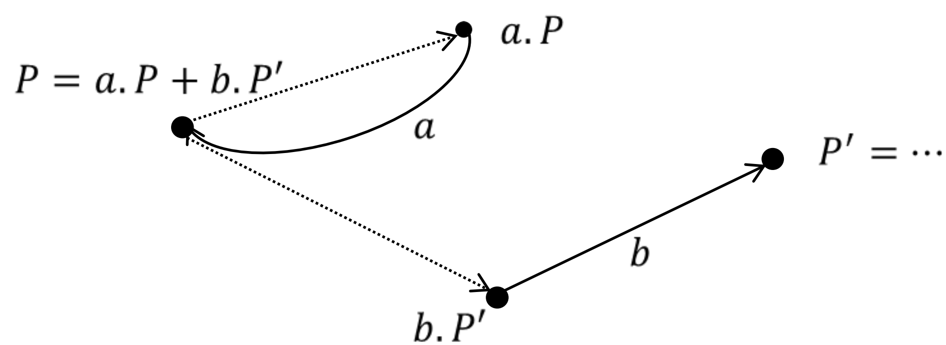
$$(\text{par-1})\ \frac{P \xrightarrow{\alpha} P'}{P \,|\, Q \xrightarrow{\alpha} P' \,|\, Q} \qquad (\text{par-2})\ \frac{Q \xrightarrow{\alpha} Q'}{P \,|\, Q \xrightarrow{\alpha} P \,|\, Q'} \qquad (\text{comm})\ \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'}$$

Invisible Action τ

$$(\text{rel})\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad (\text{hide})\ \frac{P \xrightarrow{\alpha} P' \quad \alpha, \overline{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

Labeled Transition System $[\![P]\!]_{\text{CCS}} = (\mathcal{P}(\text{Act}, \mathcal{K}), \longrightarrow, P)$

- States are Process Terms
- Transitions
  - are labeled over Actions and
  - rewrite Process Terms



PL-CCS [Gruler et al., 2009]

$$P ::= \alpha.P \mid \sum_{i \in I} P_i \mid P|P \mid P[f] \mid P \setminus L \mid X \mid \bigoplus_{\{j \in J\}} g_j.P_j$$

- (External) Selection condition g_j
- **Guarded Choice** emulates Variability

$$a.P_1 + b.P_2 \quad \text{vs.} \quad g_1.P_1 \oplus g_2.P_2$$

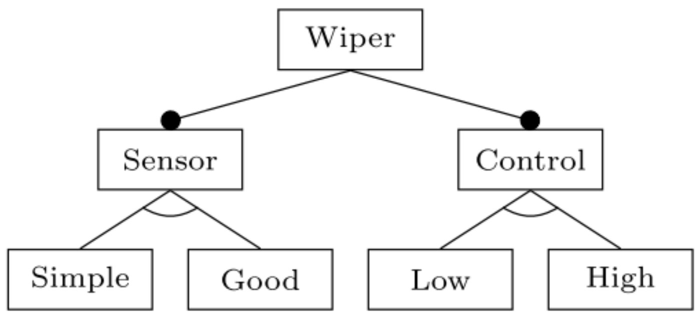
- Semantics: Choice Decision is preserved after Recursion
- Verification: HML Algorithm „tracks" selection conditions

Family-based 150%
PL Anaylsis

Behavioral Choice vs. Behavioral Change

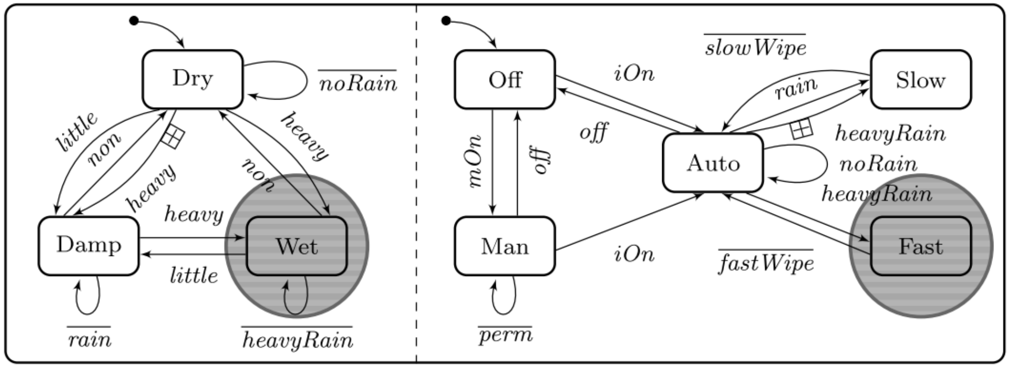
	PL-CCS	DeltaCCS
Syntax	New CCS Operator	Plain CCS + Change Modules
Semantics	Variability <u>adds</u> Term Reduction Rule	Variability <u>overrides</u> Term Rewriting Rule
PL Analysis	150% family-based	Incremental
Variability Theory	Transformation into Normal Form	Change-Sensitive Term Congruence
Supported Adaptivity	Closed World	(Semi-)Open World
Higher Order Variability	No	Yes
Orthogonality of Variability	No - Mixes Core and Variable Behaviors	Yes - Separates Core from Variable Behaviors

Delta-oriented Software Product Lines



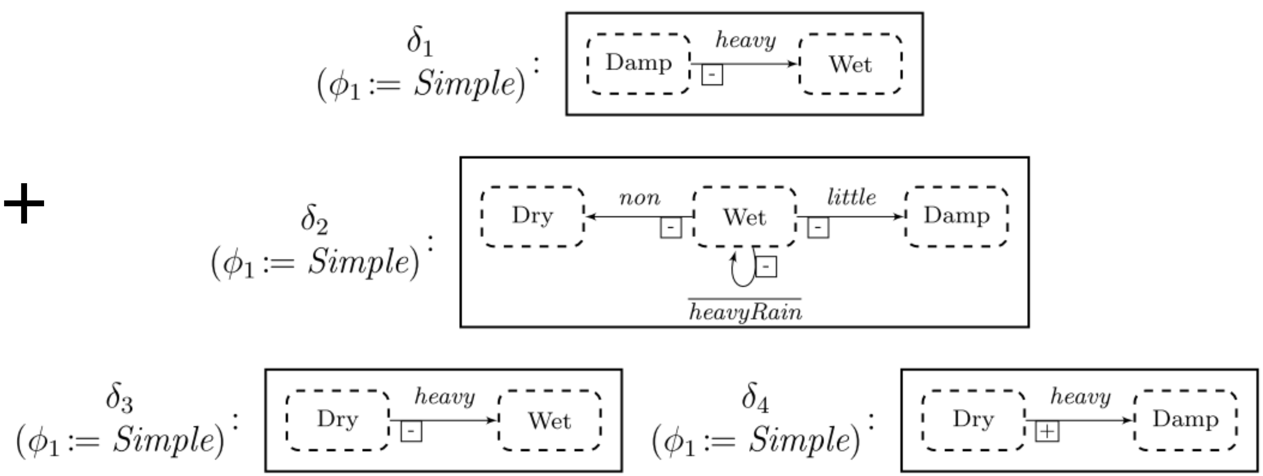
Feature Model

+



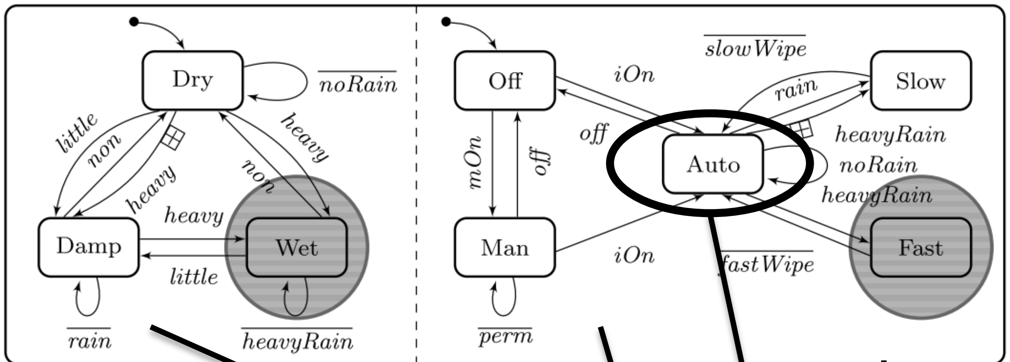
Core Product
(Good Sensor, High Control)

+



Deltas
(Good Sensor => Simple Sensor)

Translation of State Machines into CCS



Initial process for root state
with sub machines

$$P_c = \mathrm{Dry} \,|\, \mathrm{Off}$$

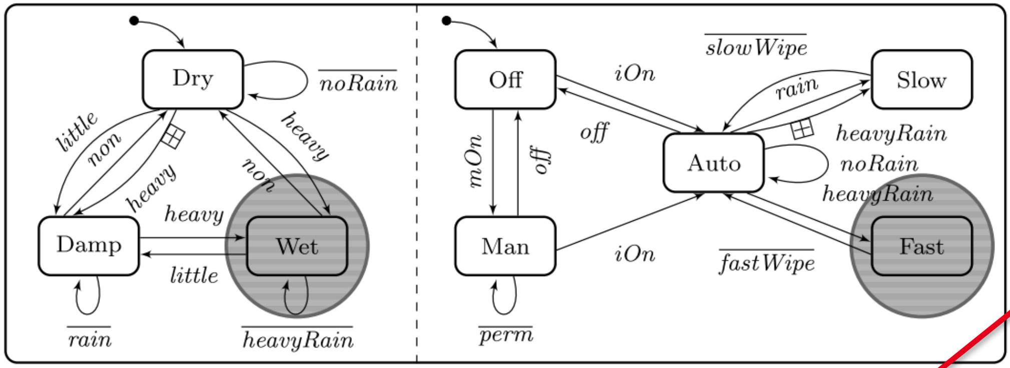
parallel sub processes
for sub machines

$$\mathrm{Dry} \overset{Def}{=} \overline{noRain}.\mathrm{Dry} + little.\mathrm{Damp} + heavy.\mathrm{Wet}$$
$$\mathrm{Damp} \overset{Def}{=} \overline{rain}.\mathrm{Damp} + heavy.\mathrm{Wet} + non.\mathrm{Dry}$$
$$\mathrm{Wet} \overset{Def}{=} \overline{heavyRain}.\mathrm{Wet} + little.\mathrm{Damp} + non.\mathrm{Dry}$$

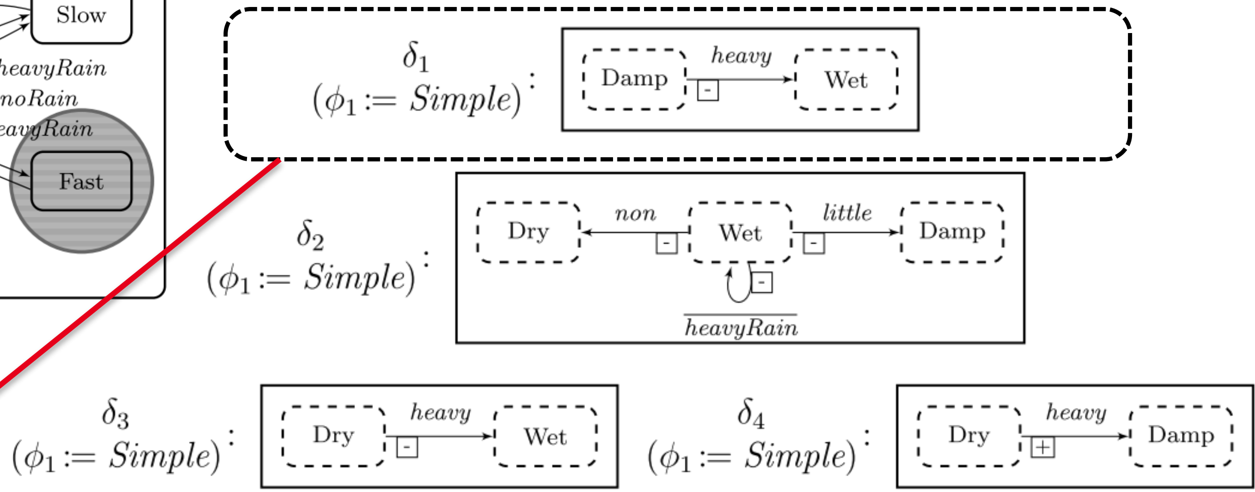
$$\mathrm{Off} \overset{Def}{=} mOn.\mathrm{Man} + iOn.\mathrm{Auto}$$
$$\mathrm{Man} \overset{Def}{=} \overline{perm}.\mathrm{Man} + off.\mathrm{Off} + ion.\mathrm{Auto}$$
$$\mathrm{Auto} \overset{Def}{=} noRain.\mathrm{Auto} + rain.\mathrm{Slow} + heavyRain.\mathrm{Fast}$$
$$\mathrm{Slow} \overset{Def}{=} \overline{slowWipe}.\mathrm{Auto}$$
$$\mathrm{Fast} \overset{Def}{=} \overline{fastWipe}.\mathrm{Auto}$$

one process constant per state with
choice over outgoing transitions

CCS Deltas



change transitions of state



$$\delta_1 = (\mathrm{Damp}, \mathit{Simple}, \langle \mathrm{Damp}' \stackrel{Def}{=} \overline{rain}.\mathrm{Damp}' + non.\mathrm{Dry}\rangle)$$

change process constant def. of state

$$\delta_2 = (\mathrm{Wet}, \mathit{Simple}, \langle \mathrm{Wet}' \stackrel{Def}{=} \mathbf{0}\rangle)$$

$$\delta_3 = (\mathrm{Dry}, \mathit{Simple}, \langle \mathrm{Dry}' \stackrel{Def}{=} \overline{noRain}.\mathrm{Dry}' + little.\mathrm{Damp}\rangle)$$

$$\delta_4 = (\mathrm{Dry}', \mathit{Simple}, \langle \mathrm{Dry}'' \stackrel{Def}{=} \overline{noRain}.\mathrm{Dry}'' + little.\mathrm{Damp} + heavy.\mathrm{Damp}\rangle)$$

DeltaCCS

Assumption: change of CCS specifications = change meaning of process names
$X := K \in \mathrm{P}(Act, K)$

$$\boxed{A \text{ CCS delta } \textit{is a triple } (K, \varphi, K') \in \mathcal{K} \times \Phi \times \mathcal{K}.}$$

A DeltaCCS specification *is a*
pair (P_c, Δ) *where* $P_c \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$
and Δ *is a finite set of CCS deltas.*

CCS Delta Application

CCS Delta Application *is defined by the function*

$$apply : \mathcal{P}(Act, \mathcal{K}) \times \Delta(\mathcal{K}, \Phi) \rightarrow \mathcal{P}(Act, \mathcal{K})$$

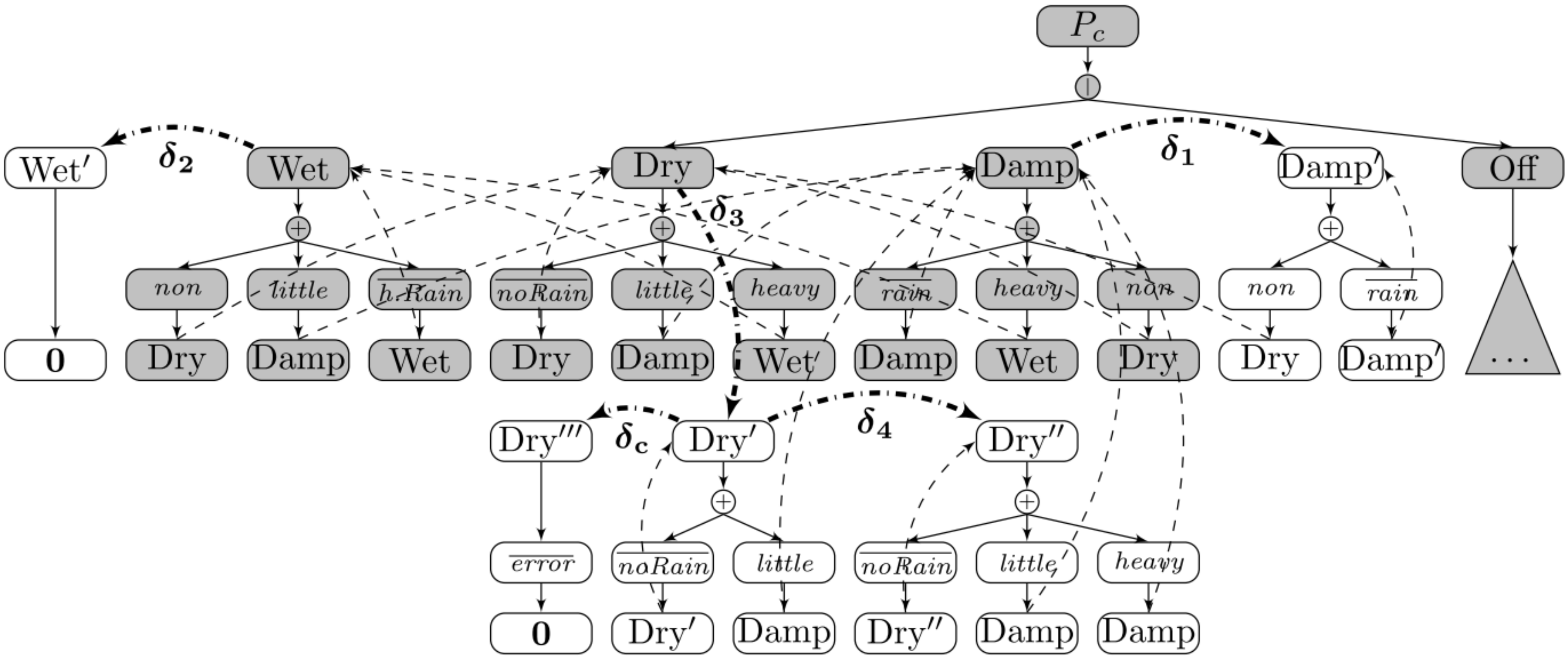
such that in $P' = apply(P, \delta)$ *with* $\delta = (K, \phi, K')$, *every occurrence of* K *in* P *is substituted by* K'.

We write $\delta(P) := apply(P, \delta)$

Applying Sets of Deltas:

$$\Delta'(P) := \{\delta_{i_1}(\ldots\delta_{i_n}(P)\ldots) \mid |\Delta'| = n \wedge k \neq \ell \Rightarrow \delta_{i_k} \neq \delta_{i_\ell}\}$$

Delta Dependency Graph



Definition 3.5 (CCS Delta Dependency). *Let* (P_c, Δ) *be a DeltaCCS specification,* $\delta_i = (K_i, \phi_i, K'_i) \in \Delta$ $(i = 1, 2)$*, and* $Dep(P_c, \Delta) = (\mathcal{V}, \mathcal{E})$.

$\delta_1 \prec \delta_2 :\Leftrightarrow$ *for all paths* $p = (P_c, \ldots, K_2, K'_2) : p = (P_c, \ldots, K_1, K'_1, \ldots, K_2, K'_2)$.

Delta Independence and Delta Conflict

Two deltas are *independent* iff

- … they have no *direct conflict* (i.e., they replace different constants) and
- … they do not (mutually) obstruct their applicability

Definition 3.6 (CCS Delta Conflict). *Let* (P_c, Δ) *be a DeltaCCS specification and* $\delta_i = (K_i, \phi_i, K_i') \in \Delta$ $(i = 1, 2)$ *with* $\delta_1 \not\prec \delta_2$ *and* $\delta_2 \not\prec \delta_1$*.* δ_1 *and* δ_2 *are* in conflict *iff* δ_1 *and* δ_2 *are not independent.*

Note: these are syntactic notions on CCS term structures

DeltaCCS Semantics (1/2)

Definition 3.7 (DeltaCCS Variant Semantics). *Let* (P_c, Δ) *be a DeltaCCS specification. The LTS* $[\![(P_c, \Delta)]\!]^{\Delta'}_{\mathsf{CCS}} = (\mathcal{P}(\mathsf{Act}, \mathcal{K}), \longrightarrow_{\Delta'}, P)$ *for* $P \in \Delta'(P_c)$ *defines the semantics of the* process variant *for* $\Delta' \subseteq \Delta$ *where*

$$\longrightarrow_{\Delta'} \subseteq (\mathcal{P}(\mathsf{Act}, \mathcal{K}) \times (\mathsf{Act} \cup \{\tau\}) \times \mathcal{P}(\mathsf{Act}, \mathcal{K}))$$

is the least relation satisfying the rule (dstep) $\dfrac{P \xrightarrow{\alpha} P' \qquad P'' \in \Delta'(P')}{P \xrightarrow{\alpha}_{\Delta'} P''}$.

Repetitive Delta Application after Term Rewriting

DeltaCCS Semantics (2/2)

override recursion rule

track applied deltas

$$\text{(delta)}\ \frac{(K', \Sigma) \xrightarrow{\alpha} (P, \Sigma') \qquad \delta = (K, \phi, K') \in \Delta}{(K, \Sigma) \xrightarrow{\alpha} (P, \Sigma' \cup \{\delta\})}
\qquad
\text{(drec)}\ \frac{(P, \Sigma) \xrightarrow{\alpha} (P', \Sigma') \quad K \overset{Def}{=} P \qquad \Sigma \cap (\{K\} \times \Phi \times \mathcal{K}) = \emptyset}{(K, \Sigma) \xrightarrow{\alpha} (P', \Sigma')}$$

$$\text{(dpref)}\ \frac{}{(a.P, \Sigma) \xrightarrow{\alpha} (P, \Sigma)}
\qquad
\text{(dchoice)}\ \frac{(P_j, \Sigma) \xrightarrow{\alpha} (P'_j, \Sigma') \quad j \in I}{(\sum_{i \in I} P_i, \Sigma) \xrightarrow{\alpha} (P'_j, \Sigma')}$$

$$\text{(dpar-1)}\ \frac{(P, \Sigma) \xrightarrow{\alpha} (P', \Sigma')}{(P \,|\, Q, \Sigma) \xrightarrow{\alpha} (P' \,|\, Q, \Sigma')}
\qquad
\text{(dpar-2)}\ \frac{(Q, \Sigma) \xrightarrow{\alpha} (Q', \Sigma')}{(P \,|\, Q, \Sigma) \xrightarrow{\alpha} (P \,|\, Q', \Sigma')}$$

$$\text{(dcomm)}\ \frac{(P, \Sigma) \xrightarrow{\alpha} (P', \Sigma') \quad (Q, \Sigma) \xrightarrow{\overline{\alpha}} (Q', \Sigma'') \quad \mathit{compatible}(\Sigma', \Sigma'')}{(P \,|\, Q, \Sigma) \xrightarrow{\tau} (P' \,|\, Q', \Sigma' \cup \Sigma'')}$$

delta consistency of concurrent processes

$$\text{(dhide)}\ \frac{(P, \Sigma) \xrightarrow{\alpha} (P', \Sigma') \quad \alpha, \overline{\alpha} \notin L}{(P \setminus L, \Sigma) \xrightarrow{\alpha} (P' \setminus L, \Sigma')}
\qquad
\text{(drel)}\ \frac{(P, \Sigma) \xrightarrow{\alpha} (P', \Sigma')}{(P\,[m]\,, \Sigma) \xrightarrow{m(\alpha)} (P'\,[m]\,, \Sigma')}$$

Definition 3.8 (DeltaCCS Semantics). *Let (P_c, Δ) be a DeltaCCS specification. The LTS $[\![(P_c, \Delta)]\!]_\Delta = (\mathcal{P}(\mathsf{Act}, \mathcal{K}) \times \Delta(\mathcal{K}, \Phi), \longrightarrow, (P_c, \emptyset))$ defines the semantics of (P_c, Δ) where*

$$\longrightarrow \subseteq (\mathcal{P}(\mathsf{Act}, \mathcal{K}) \times \Delta(\mathcal{K}, \Phi)) \times (\mathsf{Act} \cup \{\tau\}) \times (\mathcal{P}(\mathsf{Act}, \mathcal{K}) \times \Delta(\mathcal{K}, \Phi))$$

is the least relation satisfying the rules in Fig. 5.

Bisimulation Equivalence

Definition 3.1 *(Simulation).* Let $P_1, P_2 \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$. A *simulation between P_1 and P_2* is a relation $S \subseteq \mathcal{P}(\mathsf{Act}, \mathcal{K}) \times \mathcal{P}(\mathsf{Act}, \mathcal{K})$ such that $(P_1, P_2) \in S$ and for every $(P, Q) \in S$ it holds that if $P \xrightarrow{\alpha} P'$, then there exists a Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in S$. P_2 *simulates* P_1, denoted by $P_1 \sqsubseteq P_2$, iff there exists a simulation between P_1 and P_2. P_1 and P_2 are *simulation equivalent*, $P_1 \simeq P_2$, iff P_1 simulates P_2 and P_2 simulates P_1.

Definition 3.2 *(Bisimulation).* Let $P_1, P_2 \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$. A *bisimulation between P_1 and P_2* is a relation $R \subseteq \mathcal{P}(\mathsf{Act}, \mathcal{K}) \times \mathcal{P}(\mathsf{Act}, \mathcal{K})$ such that $(P_1, P_2) \in R$ and for every $(P, Q) \in R$ it holds that

- if $P \xrightarrow{\alpha} P'$, then there exists a Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in R$, and
- if $Q \xrightarrow{\alpha} Q'$, then there exists a P' such that $P \xrightarrow{\alpha} P'$ and $(P', Q') \in R$.

DeltaCSS Semantics is Correct

Theorem 3.1 (Correctness). *Let* (P_c, Δ) *be a DeltaCCS specification,* $\Delta' \subseteq \Delta$ *conflict-free,* $[\![(P_c, \Delta')]\!]_\Delta = (Q_1, \longrightarrow, q_1)$ *and* $[\![P_c]\!]^{\Delta'}_{\mathsf{CCS}} = (Q_2, \longrightarrow_{\Delta'}, q_2)$. *Then for* $q_1' = (P_c, \Delta') \in Q_1$ *it holds that* $q_1' \simeq q_2$.

… and even in case of delta conflicts we have at least soundness:

$$q_1' \sqsubseteq q_2$$

Formalizing the Notion of (Correct) Behavior

$$P \models \varphi$$

- What is φ? temporal properties (safety, liveness)
- How to verify φ on CCS specifications P? model checking
- What happens with φ in $\delta(P)$? it depends…

What is φ?

Definition 4.1 (Modal μ-Calculus). *A modal μ-calculus formula is an expression following the form $\nu Z.\psi \wedge [\alpha]Z$ or $\mu Z.\psi \vee \langle\alpha\rangle Z$ where*

$$\psi \quad ::= \quad tt \mid ff \mid q \mid \neg q \mid Z \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle\alpha\rangle\psi \mid [\alpha]\psi$$

where $q \in \mathcal{P}$, $\alpha \in \mathsf{Act}$ and $Z \in \mathsf{Var}$. Let $P \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$ and φ a formula. Given an evaluation of atomic propositions $\mathcal{I}_{\mathcal{P}} : \mathcal{P} \to 2^{\mathcal{P}(\mathsf{Act},\mathcal{K})}$ and an evaluation of variables $\mathcal{I}_{\mathsf{Var}} : \mathsf{Var} \to 2^{\mathcal{P}(\mathsf{Act},\mathcal{K})}$, $P \models \varphi$ iff $P \in \|\varphi\|^{\mathcal{I}_{\mathcal{P}}}_{\mathcal{I}_{\mathsf{Var}}}$ (cf. Fig. 6).

"[…] the system is intended to perform *fast wiping* whenever receiving the input *heavy… […]*":

$$\varphi := \mu Z.\langle heavy\rangle\langle\langle\overline{fastWipe}\rangle\rangle\mathtt{tt} \vee \langle - \rangle Z$$

How to verify φ on CCS specifications P?

$$
\begin{aligned}
\|q\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \mathcal{I}_{\mathcal{P}}(q) \\
\|\neg q\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \mathcal{P}(\mathsf{Act}, \mathcal{K}) \setminus \mathcal{I}_{\mathcal{P}}(q) \\
\|Z\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \mathcal{I}_{\mathsf{Var}}(Z) \\
\|\varphi_1 \wedge \varphi_2\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \|\varphi_1\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \cap \|\varphi_2\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \\
\|\varphi_1 \vee \varphi_2\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \|\varphi_1\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \cup \|\varphi_2\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \\
\|[\alpha]\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \left\{ P \mid \forall P' : P \xrightarrow{\alpha} P' \Rightarrow P' \in \|\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \right\} \\
\|\langle\alpha\rangle\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \left\{ P \mid \exists P' : P \xrightarrow{\alpha} P' \wedge P' \in \|\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} \right\} \\
\|\nu Z.\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \bigcup \left\{ \Pi \subseteq \mathcal{P}(\mathsf{Act}, \mathcal{K}) \mid \Pi \subseteq \|\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}[Z:=\Pi]} \right\} \\
\|\mu Z.\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}} &= \bigcap \left\{ \Pi \subseteq \mathcal{P}(\mathsf{Act}, \mathcal{K}) \mid \Pi \supseteq \|\varphi\|_{\mathcal{I}_{\mathcal{P}}}^{\mathcal{I}_{\mathsf{Var}}[Z:=\Pi]} \right\}
\end{aligned}
$$

What happens with φ in $\delta(P)$? (1/4)

(1) $P \simeq P'$ iff P and P' satisfy the same set of μ-calculus formulae

[Stirling et al.]

(2) $P \equiv P'$ implies $P \simeq P'$

[Milner et al.]

➢ Standard Congruence on CCS Terms:

- $P + Q \equiv Q + P, P + P \equiv P, \ldots$
- $P|Q \equiv Q|P, P|\mathbf{0} \equiv P$
- $\alpha.(P + Q) \not\equiv \alpha.P + \alpha.Q$
- ....

What happens with φ in $\delta(P)$? (2/4)

- Delta-aware CCS congurence

Proposition 4.1. *Let* $\delta = (K, \phi, K') \in \Delta(\mathcal{K}, \Phi)$ *and* $P, Q, P' \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$.

$$
\begin{aligned}
\delta(\alpha.P) &\equiv \alpha.\delta(P) && (1)\\
\delta(P + Q) &\equiv \delta(P) + \delta(Q) && (2)\\
\delta(P \,|\, Q) &\equiv \delta(P) \,|\, Q \textit{ if } \delta(Q) \equiv Q && (3)\\
\delta(X) &\equiv \delta(P') \textit{ if } K \neq X \textit{ and } X \overset{Def}{=} P' && (4)\\
\delta(P) &\equiv P \textit{ if } \delta \textit{ is not applicable in } P && (5)
\end{aligned}
$$

What happens with φ in $\delta(P)$? (3/4)

- Handling delta sets:

Lemma 4.1. *Let* $\delta_1 = (K_1, \phi_1, K_1') \in \Delta(\mathcal{K}, \Phi)$, $\delta_2 = (K_2, \phi_2, K_2') \in \Delta(\mathcal{K}, \Phi)$ *and* $P \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$. *Then* $\delta_1(\delta_2(P)) \equiv \delta_2(\delta_1(P))$ *iff* δ_1 *and* δ_2 *are independent.*

What happens with φ in $\delta(P)$? (4/4)

Proposition 4.2. *Let* $P, Q, R \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$ *and* φ *a* μ*-calculus formula.*

$$P \models \varphi \wedge P \equiv Q \quad \Rightarrow \quad Q \models \varphi \tag{6}$$

$$P \models \varphi \wedge Q \models \varphi \quad \Rightarrow \quad P + Q \models \varphi \tag{7}$$

$$P \mid R \models \varphi \wedge P \equiv Q \quad \Rightarrow \quad Q \mid R \models \varphi \tag{8}$$

Theorem 4.1. *Let* $\delta = (K, \phi, K') \in \Delta(\mathcal{K}, \Phi)$ *and* $P \in \mathcal{P}(\mathsf{Act}, \mathcal{K})$. *If* $\delta(P) \equiv P$ *and* $P \models \varphi$ *then* $\delta(P) \models \varphi$.

Referenzen

- Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay: **Symbolic model checking of software product lines.** ICSE 2011: 321-330

- Malte Lochau, Stephan Mennicke, Hauke Baller, Lars Ribbeck: **Incremental model checking of delta-oriented software product lines.** J. Log. Algebr. Meth. Program. 85(1): 245-267 (2016)