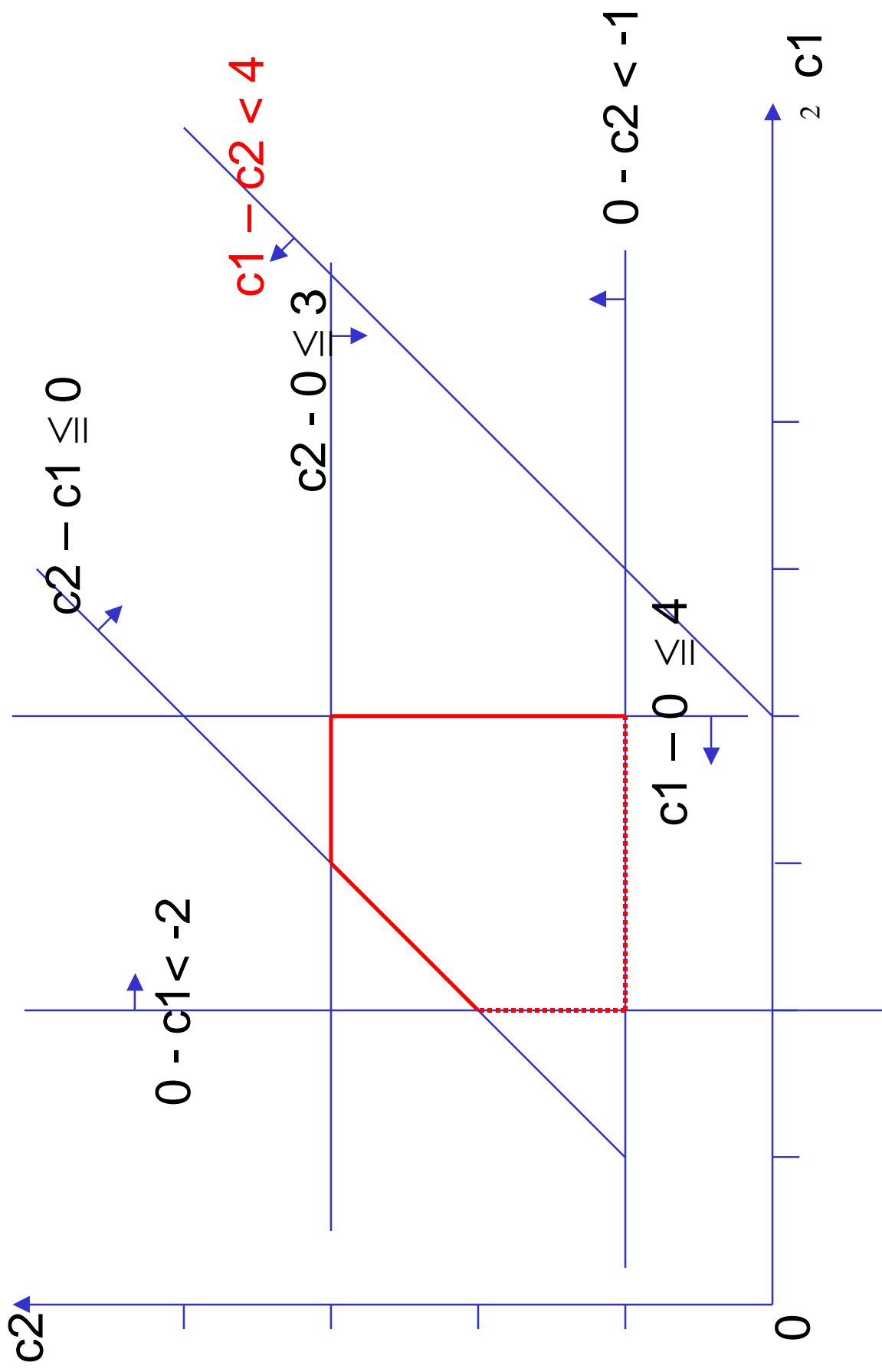


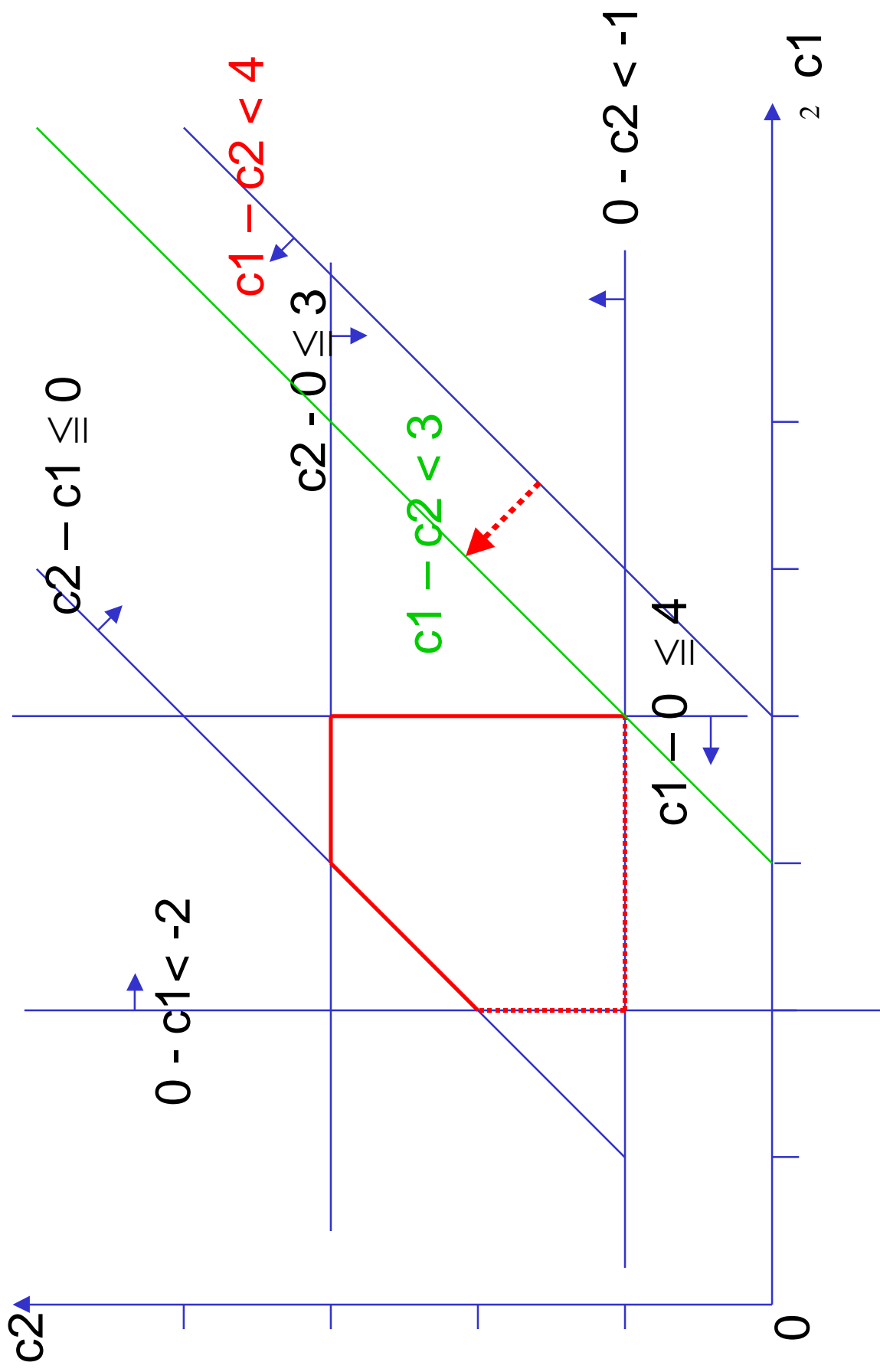
Computergestützte Verifikation

10.6.

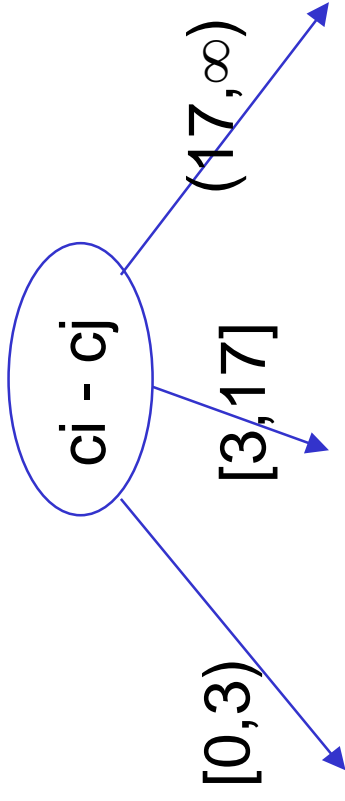
Normalisierung von Zonen



Normalisierung von Zonen



Clock Difference Diagrams



- Nachbildung aller BDD-Operationen, plus
- Projektion
- Öffnung
- Schnitt

aber: lange Zeit offen: Normalisierung
(inzwischen wohl gelöst für DDD)

6. Abstraktion

Simulation

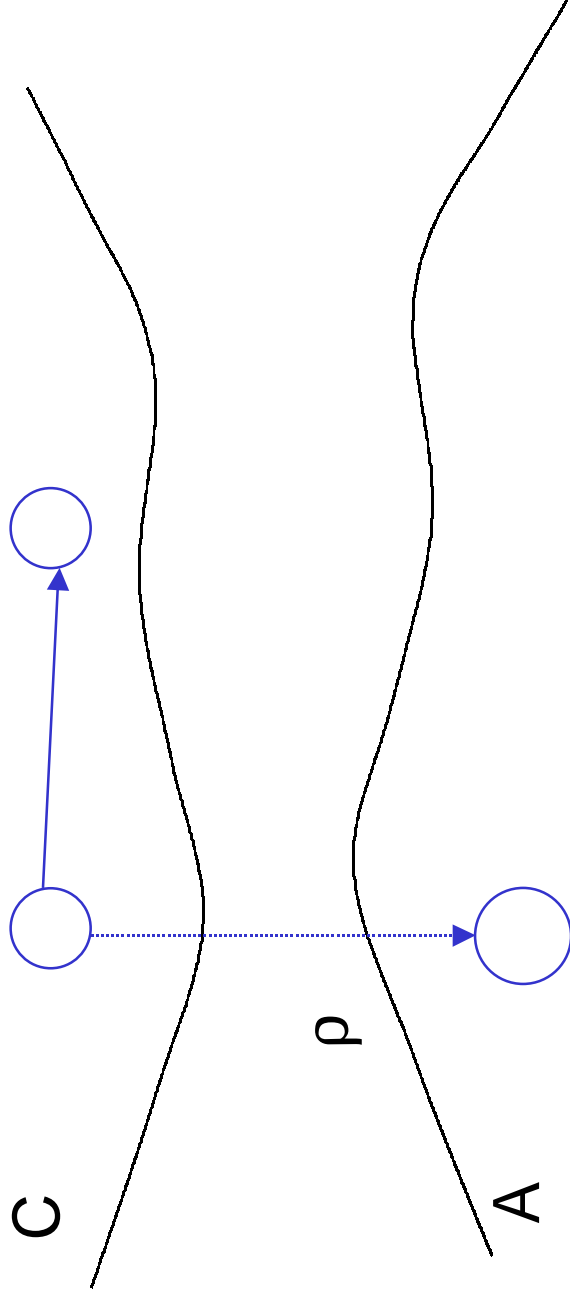
ρ ist Simulationsrelation, wenn für alle c, a, c' :

Wenn $c \rho a$ und $c \rightarrow c'$ in C , so ex. ein a' mit
 $c' \rho a'$ und $a \rightarrow a'$ in A

Simulation

ρ ist Simulationsrelation, wenn für alle c, a, c' :

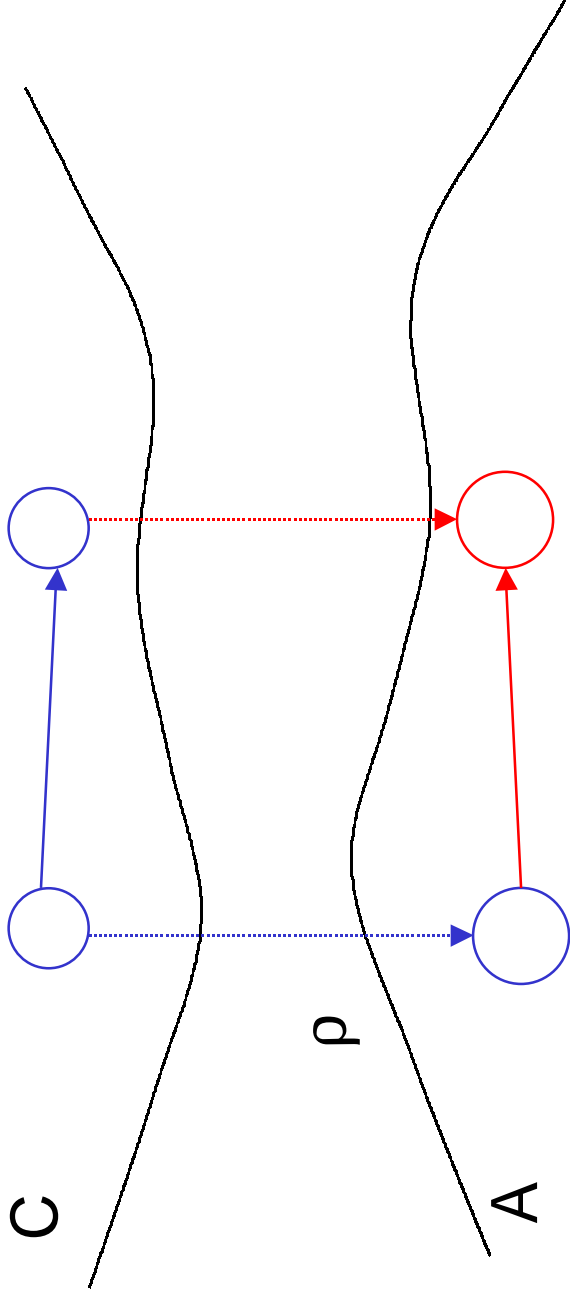
Wenn $c \rho a$ und $c \rightarrow c'$ in C , so ex. ein a' mit $c' \rho a'$ und $a \rightarrow a'$ in A



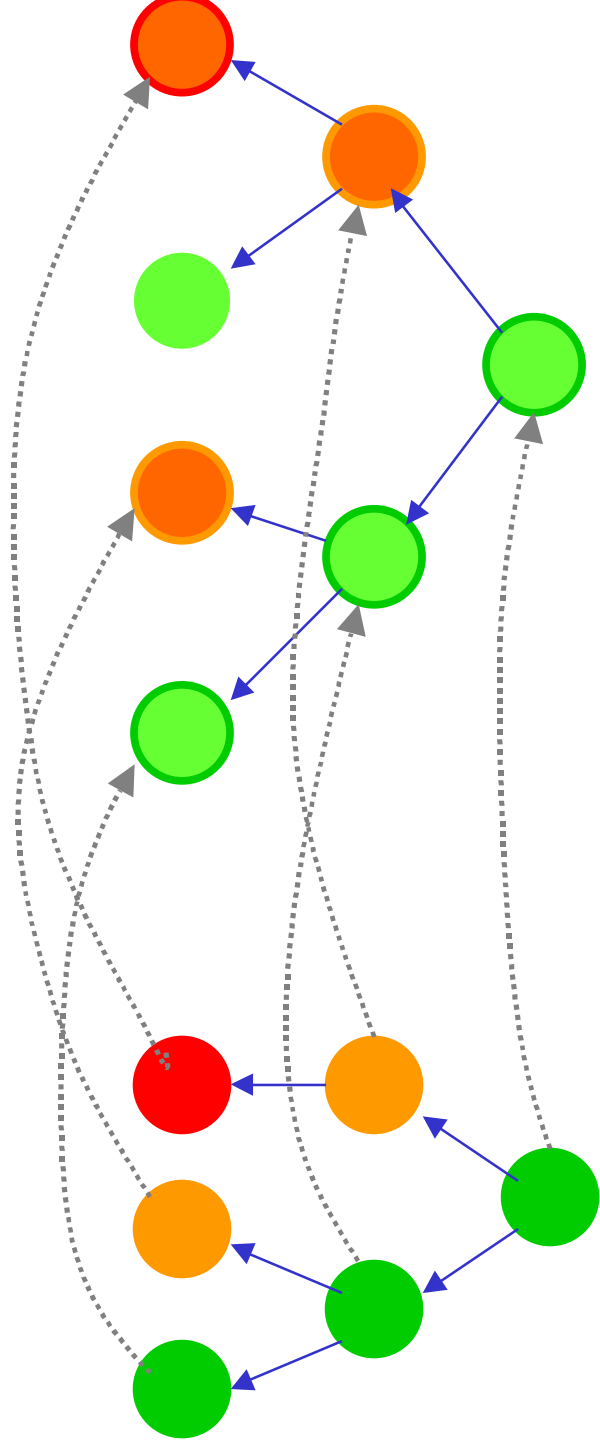
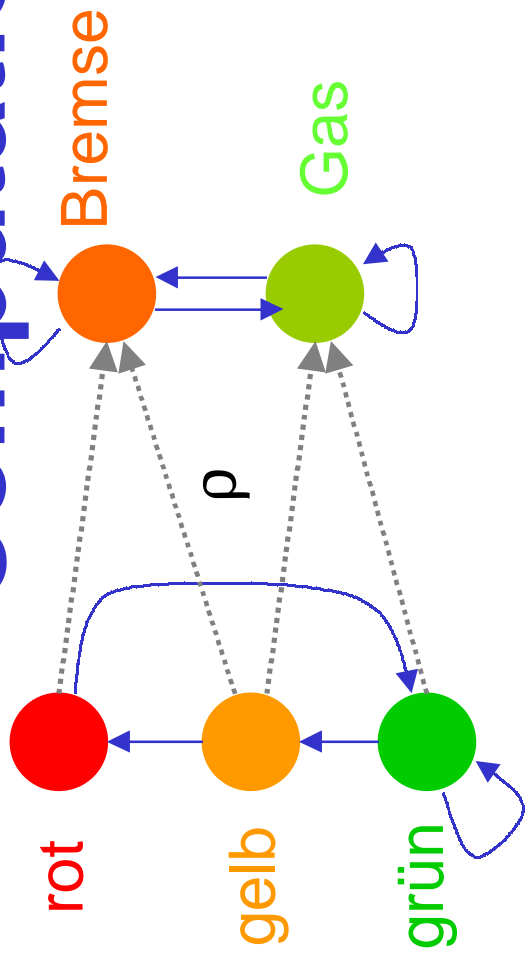
Simulation

ρ ist Simulationsrelation, wenn für alle c, a, c' :

Wenn $c \rho a$ und $c \rightarrow c'$ in C , so ex. ein a' mit $c' \rho a'$ und $a \rightarrow a'$ in A



Simulation und Computation Tree

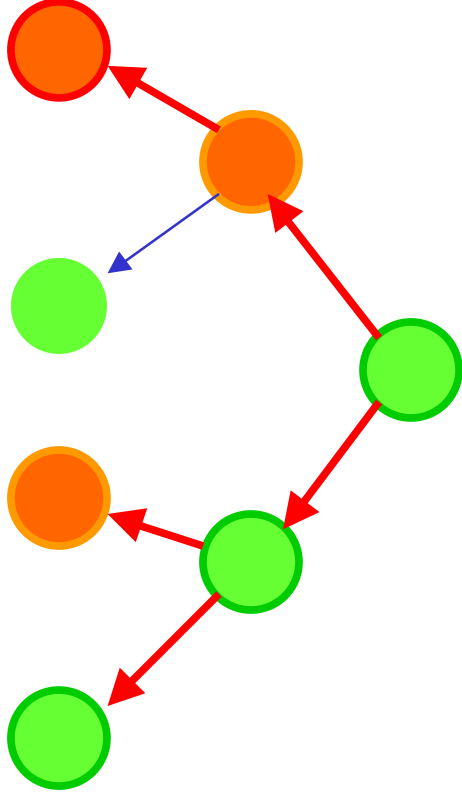


Bewahrung von ACTL*

Fazit: Berechnungsbaum von C findet sich “als Teilbaum”
des Berechnungsbaums von A wieder

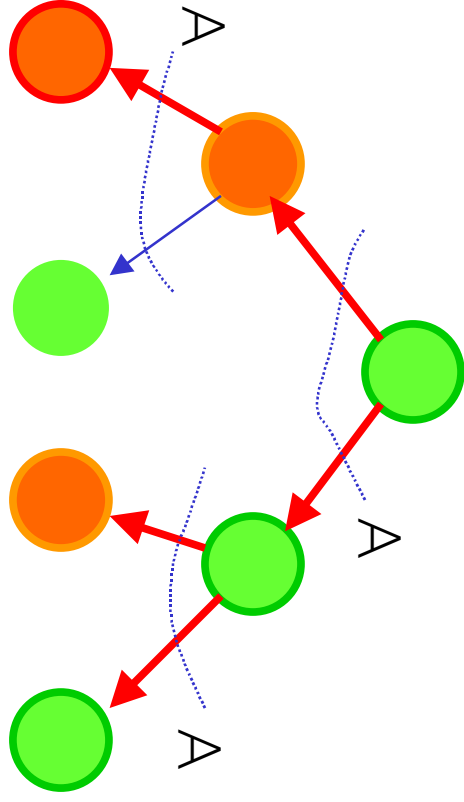
Bewahrung von ACTL*

Fazit: Berechnungsbaum von C findet sich “als Teilbaum”
des Berechnungsbaums von A wieder



Bewahrung von ACTL*

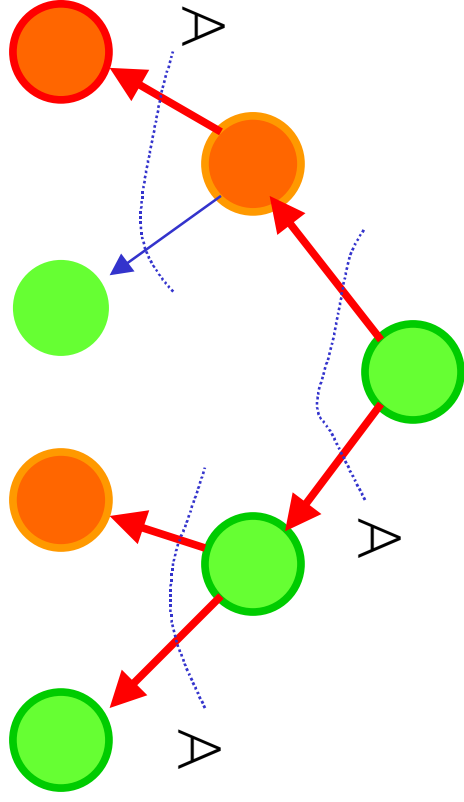
Fazit: Berechnungsbaum von C findet sich “als Teilbaum”
des Berechnungsbaums von A wieder



ACTL* quantifiziert nur
universell über Pfade

Bewahrung von ACTL*

Fazit: Berechnungsbaum von C findet sich “als Teilbaum” des Berechnungsbaums von A wieder



ACTL* quantifiziert nur
universell über Pfade

→ Satz: Wenn C A simuliert, so gilt jede ACTL*-Eigenschaft von A auch in C

Ein Transitionssystem simuliert jede Überapproximation

Sei $[S, E, A]$ ein Transitionssystem

Wenn $S \subseteq S'$ und $E \subseteq E'$, so ist $[S', E', A]$ eine Überapproximation von S

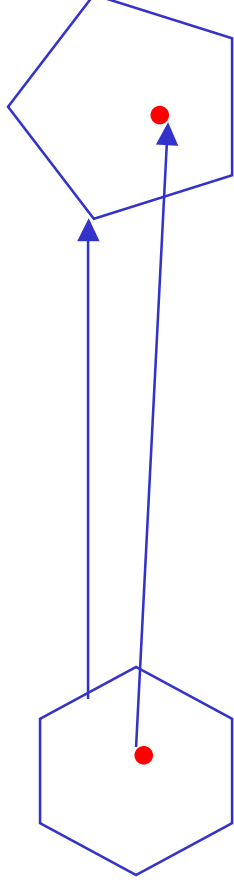
$\rho: Id$

→ Wenn eine ACTL*-Eigenschaft in einer Überapproximation gilt, so auch im originalen System

Interessant für symbolische Verifikation

Konstruktion von Abstraktionen

- geg: Konkretes System $C = [S, E]$, Menge A von abstrakten Zuständen, Relation ρ von C in A
- ges: E' , so daß ρ Simulationsrelation zwischen C und A wird

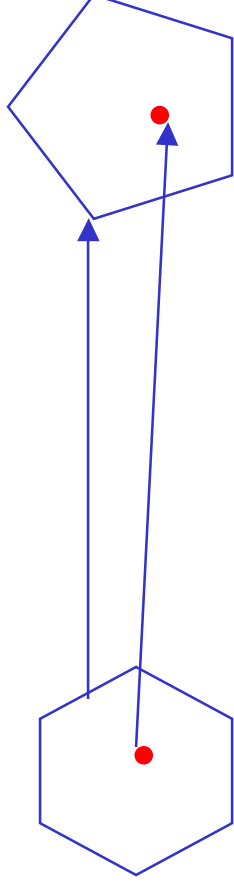


Konstruktion von Abstraktionen

geg: Konkretes System $C = [S, E]$, Menge A von abstrakten Zuständen, Relation ρ von C in A

ges: E' , so daß ρ Simulationsrelation zwischen C und A wird

Lösung: $a \rightarrow a'$ gdw. es gibt c, c' mit $c \rho a$ und $c' \rho a'$ und $c \rightarrow c'$



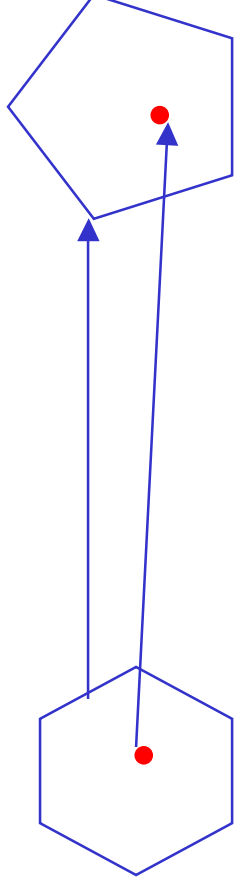
Konstruktion von Abstraktionen

geg: Konkretes System $C = [S, E]$, Menge A von abstrakten Zuständen, Relation ρ von C in A

ges: E' , so daß ρ Simulationsrelation zwischen C und A wird

Lösung: $a \rightarrow a'$ gdw. es gibt c, c' mit $c \rho a$ und $c' \rho a'$ und $c \rightarrow c'$

“Existential Abstraction”



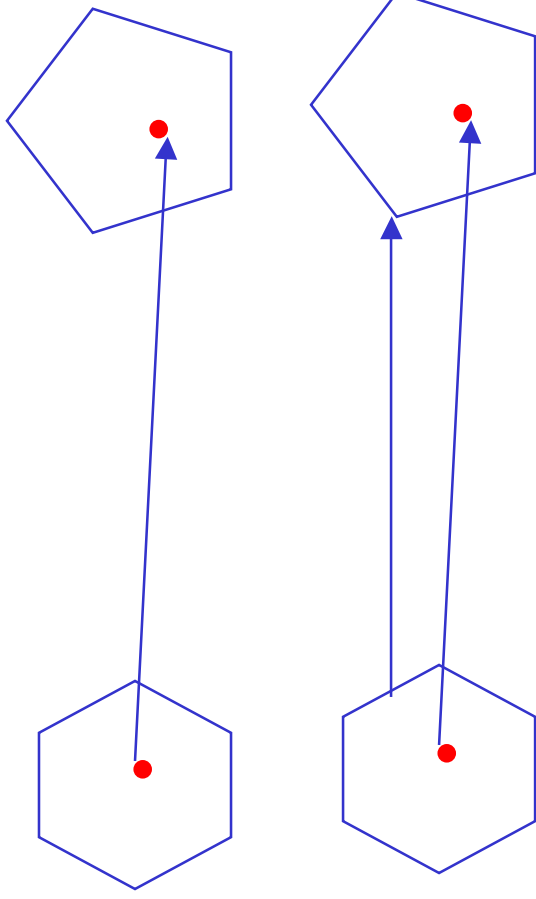
Konstruktion von Abstraktionen

geg: Konkretes System $C = [S, E]$, Menge A von abstrakten Zuständen, Relation ρ von C in A

ges: E' , so daß ρ Simulationsrelation zwischen C und A wird

Lösung: $a \rightarrow a'$ gdw. es gibt c, c' mit $c \rho a$ und $c' \rho a'$ und $c \rightarrow c'$

“Existential Abstraction”



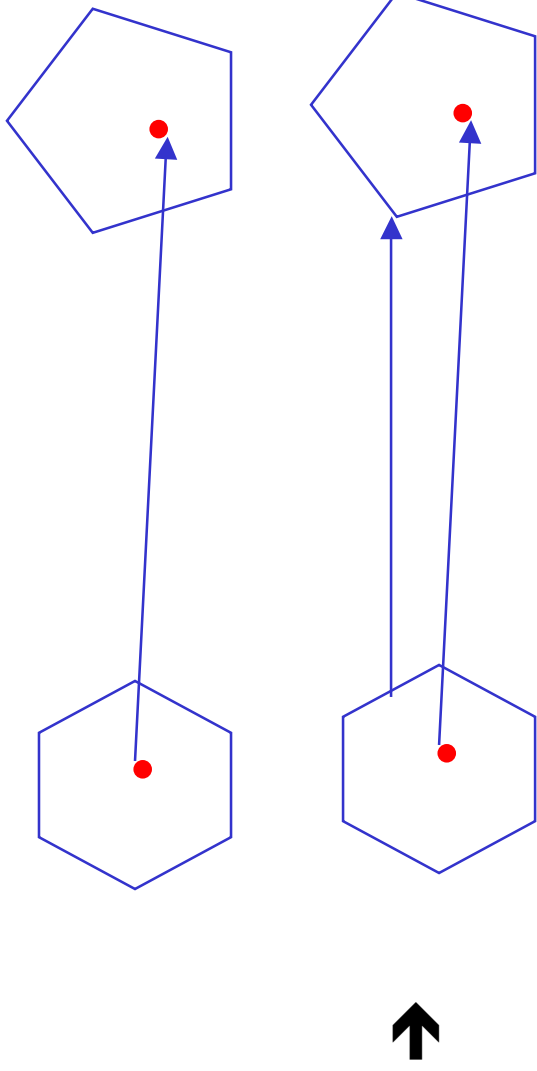
Konstruktion von Abstraktionen

geg: Konkretes System $C = [S, E]$, Menge A von abstrakten Zuständen, Relation ρ von C in A

ges: E' , so daß ρ Simulationsrelation zwischen C und A wird

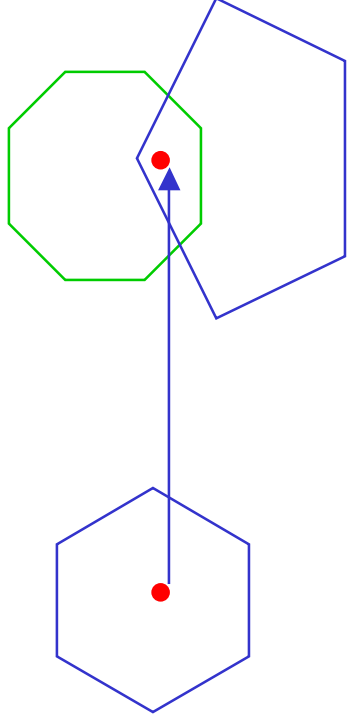
Lösung: $a \rightarrow a'$ gdw. es gibt c, c' mit $c \rho a$ und $c' \rho a'$ und $c \rightarrow c'$

“Existential Abstraction”

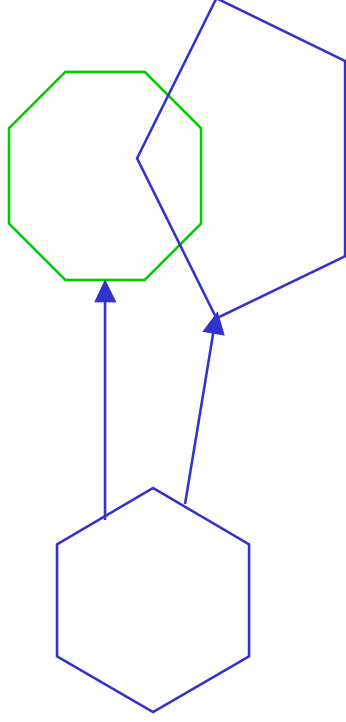


Existential Abstraction vs Simulation

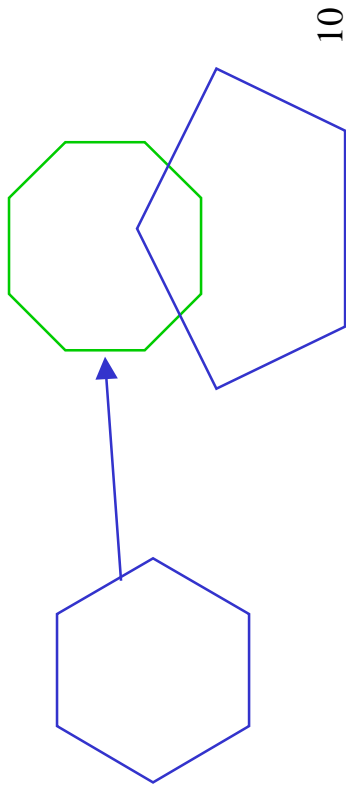
Wenn sich abstrakte Zustände überlappen können, fordert existential abstraction mehr Ereignisse als die Def. von Simulation



Existential Abstraction:



nach Def. Simulation reicht:



Abstraktionsfunktionen

Wenn abstrakte Zustände zu disjunkten konkreten Zustandsmengen korreliert sind ($c \rho a$ und $c \rho a' \rightarrow a = a'$) dann fallen Simulation und Existential Abstraction zusammen, d.h. die durch Existential Abstraction definierte Ereignismenge ist die kleinste, die ρ zu einer Simulationsrelation macht.

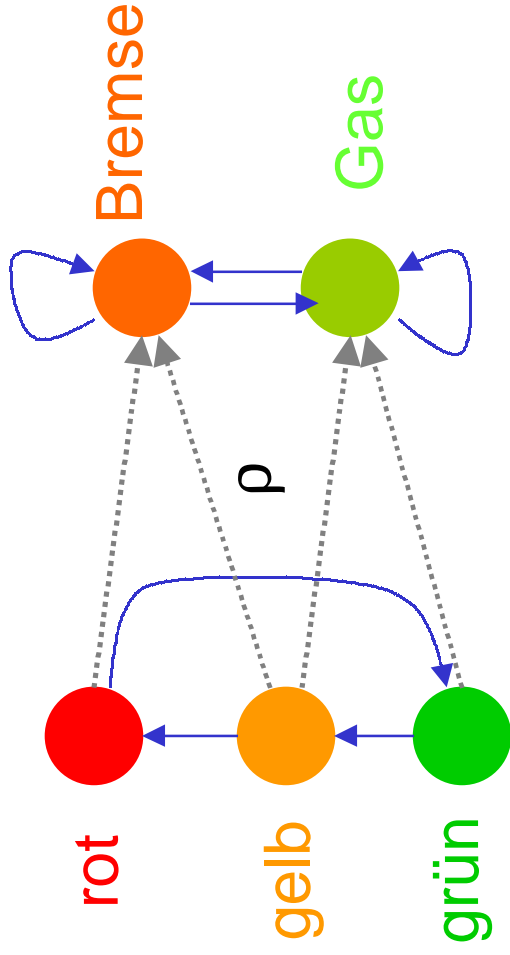
Simulation: Wenn $c \rho a$ und $c \rightarrow c'$, so ex. a' mit $c' \rho a'$ und $a \rightarrow a'$

Ex. Abstraction: Wenn $c \rho a$ und $c \rightarrow c'$ und $c' \rho a'$, so $a \rightarrow a'$

- Regionen
- Symmetrie

Existential Abstraction

- Sehr praktikabel:
1. Wähle abstrakte Zustandsmenge (nach Kriterien wie z.B. Bewahrung elementarer Eigenschaften)
 2. Ergänze kanonisch eine Übergangsrelation



Beispiele

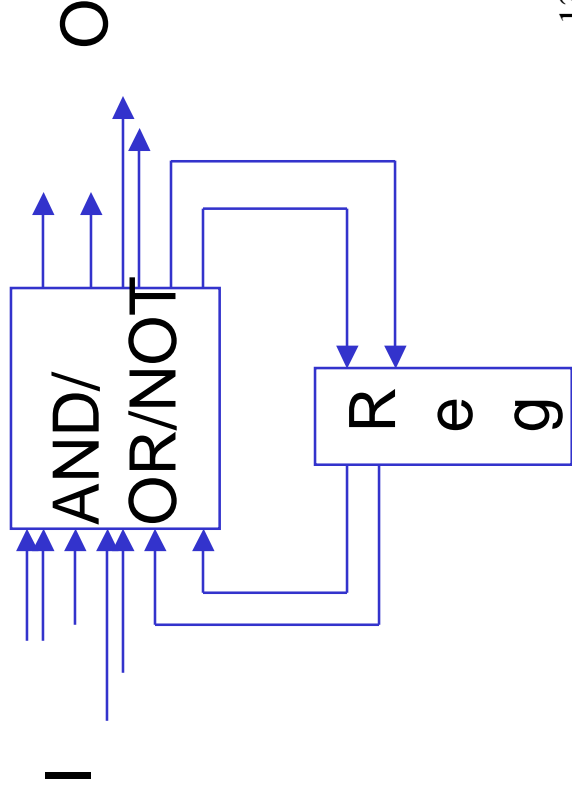
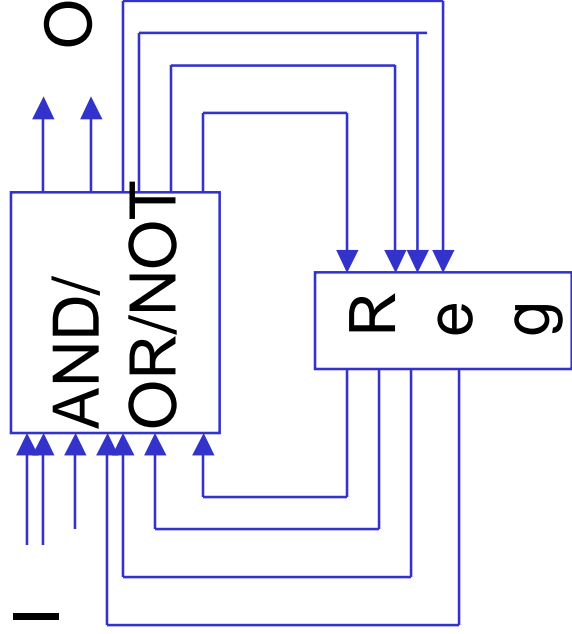
1. Vorzeichenabstraktion

2. Logarithmische Abstraktion

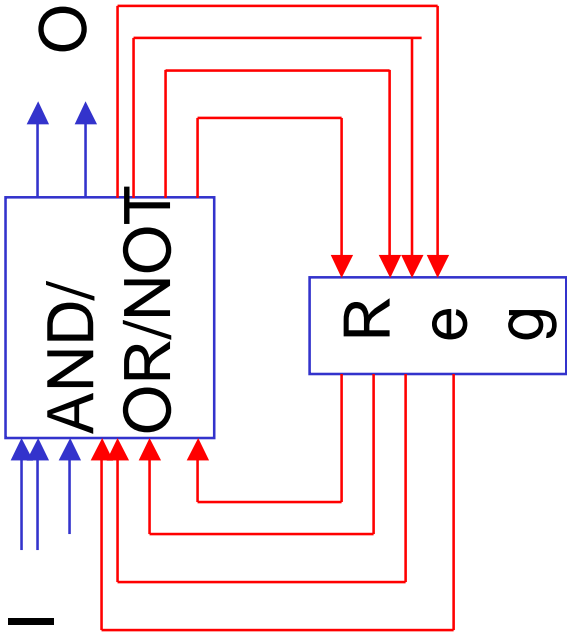
$\text{Nat} \rightarrow \{< 0, 0, > 0\}$

$\text{Nat} \rightarrow \text{Nat}; i \rightarrow \lg i = \log_2(i+1)$

3. Bit-Abstraktion



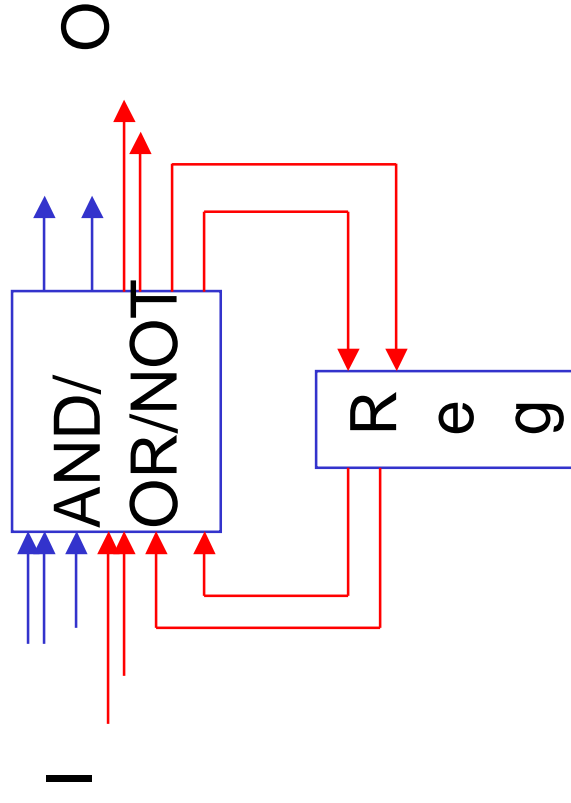
Bit-Abstraktion



$$T(x_1 \dots x_n, x_1' \dots x_n') =$$

$$\exists i_1 \dots i_m$$

$$R(x_1 \dots x_n, i_1 \dots i_m, x_1' \dots x_n')$$



$$T(x_1 \dots x_j, x_1' \dots x_j') =$$

$$\exists i_1 \dots i_m \exists x_{j+1} \dots x_n \exists x_{j+1}' \dots x_n'$$

$$R(x_1 \dots x_n, i_1 \dots i_m, x_1' \dots x_n')$$

Bisimulation

Wenn sowohl ρ als auch ρ^{-1} Simulationsrelationen sind, heißen C und A bisimilar, und ρ heißt Bisimulationsrelation

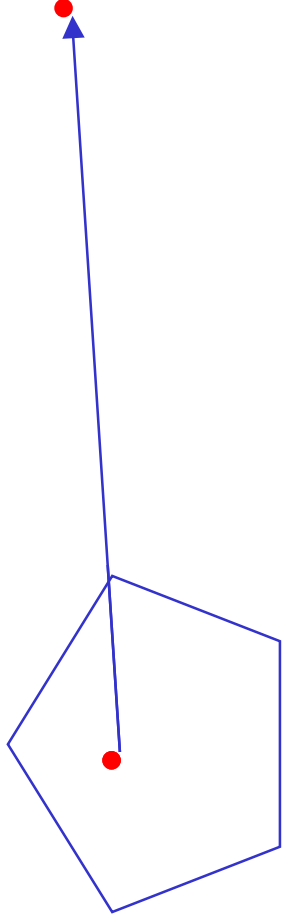
“bisimilar” ist schärfer als “ A simuliert C und C simuliert A ”!

graphische Veranschaulichung

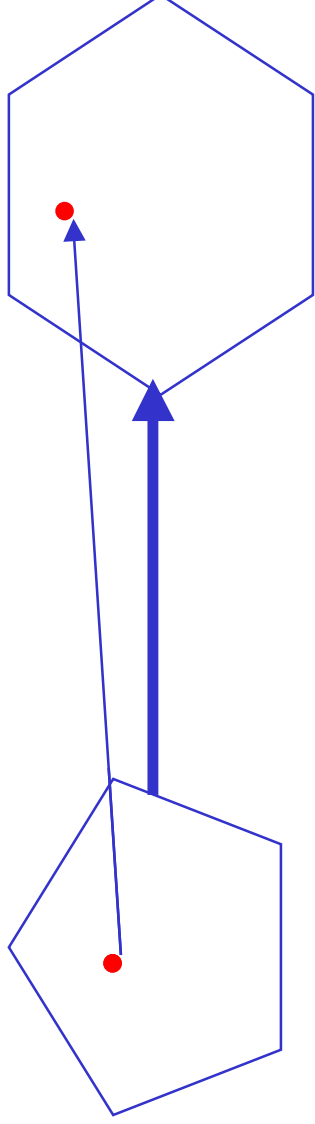
graphische Veranschaulichung



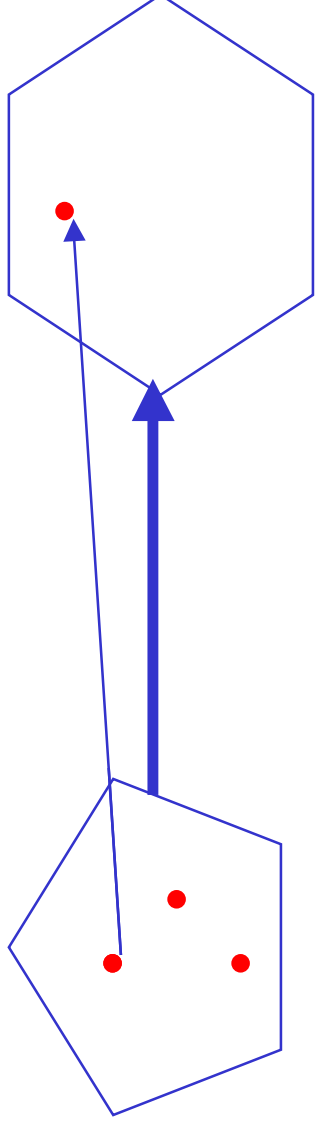
graphische Veranschaulichung



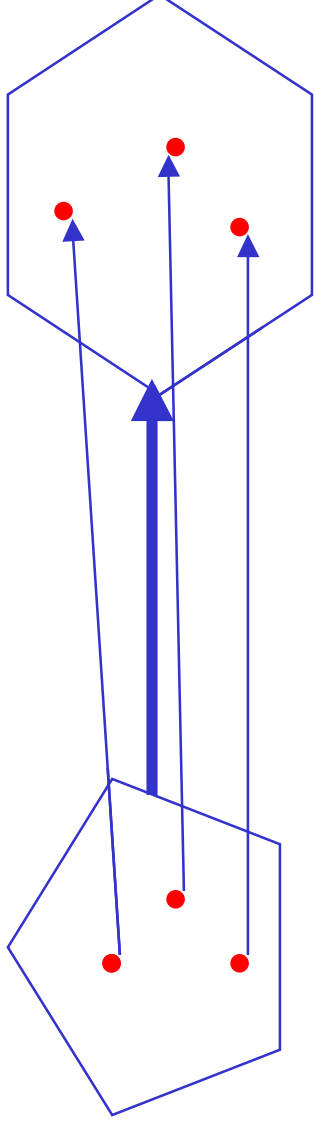
graphische Veranschaulichung



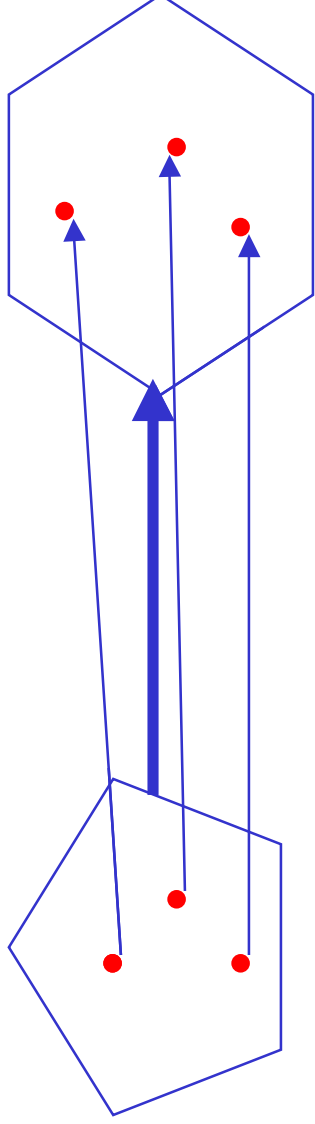
graphische Veranschaulichung



graphische Veranschaulichung



graphische Veranschaulichung



Sei $a \rightarrow a'$. Daraus folgt

für alle $c: c \rho a$, **es existiert** $c': c \rightarrow c'$ und $c' \rho a'$ (Bisimulation)

zum Vergleich:

es existiert $c: c \rho a$ **es existiert** $c': c \rightarrow c'$ und $c' \rho a'$ (exist. Abstr.)

Beispiele für Bisimulationen

1. Symmetrien

Wenn $[s] \rightarrow [s']$, so gibt es ein s'' in $[s']$ mit $s \rightarrow s''$

2. Regionen

(mit “Selbstscheifentrück” wie bei Zonen)

Bisimulation und CTL(*)

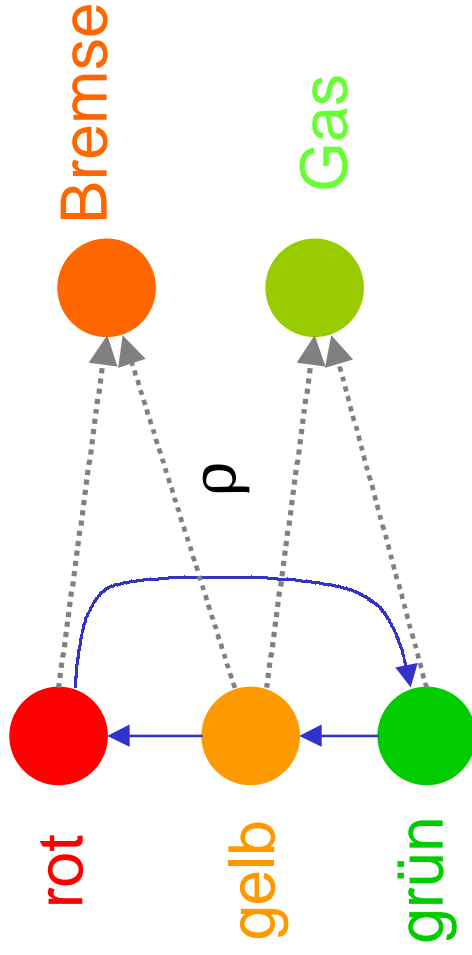
Satz 1: Wenn A und C bisimilar sind, so erfüllen sie die gleichen CTL*-Formeln

Satz 2: Wenn A und C die gleichen CTL-Formeln erfüllen, so sind sie bisimilar

Das heißt: Wenn es eine CTL*-Formel gibt, die A und C unterscheidet, so gibt es bereits eine CTL-Formel.

Konstruktion von Bisimulationen

Nicht jede abstrakte Zustandsmenge kann zu einem bisimilaren Transitionssystem ergänzt werden



Konstruktion von bisimilaren Transitionssystemen aus abstrakter Zustandsmenge erfordert Abstraktionsverfeinerung

Schwache Bisimulation

Erweitertes Spiel (aktionsbasiert):
spezielle Aktion τ : “unsichtbar”
Spieler 1 macht einen Zug (ein Schritt)
Spieler 2 darf einen Zug machen, der aus der von Spieler 1 gewählten Aktion, umrahmt von beliebig vielen τ -Aktionen, besteht, und zu einem Zustand führt, der in Relation ρ zum Zustand von Spieler 1 steht

→ C simuliert A schwach

→ schwache Bisimulation (ρ und ρ^{-1} ..., wie gehabt)
... bewahrt CTL*-X

Partial order reduction für CTL* liefert schwach bisimilares
reduziertes Transitionssystem

Zusammenfassung

Abstraktion

zentral: Simulation/Bisimulation

Simulation \rightarrow ACTL*

Bisimulation \rightarrow CTL*

Jede Menge abstrakter Zustände kann zu Simulation fortgesetzt werden, aber nicht immer zu einer Bisimulation

Die meisten Reduktionstechniken konstruieren Abstraktionen

7. Abstraktionsverfeinerung

7.1 allgemeine Abstraktionsverfeinerung

geg.: Abstraktion ges: aussagekräftigere
Abstraktion

7.2 Gegenbeispielgesteuerte Verfeinerung

geg.: Abstraktion, Scheingegenbeispiel
ges.: Abstraktion, die Gegenbeispiel ausschließt²²

Spaltung abstrakter Zustände

z.B. Zonen:

$$Z1 := Z \wedge c_i - c_j < k$$

$$Z2 := Z \wedge c_i - c_j \geq k$$

z.B. Bit-Abstraktion:

weitere Variablen sichtbar machen

7.1 allgemeine Verfeinerung

geg.: simulierende Abstraktionsrelation
z.B. Zonengraph

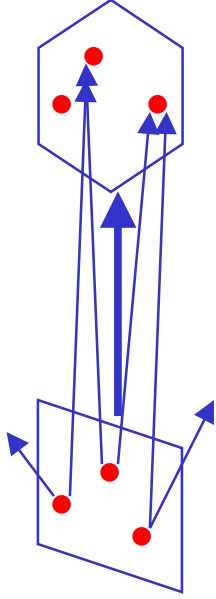
ges.: mögl. grobe Verfeinerung, die zus. Bedingungen erfüllt.
z.B. Bisimulation oder Vererbung von ACTL* in die
andere Richtung
oder Bewahrung weiterer
Elementaraussagen

Mittel: Spaltung von abstrakten Zuständen

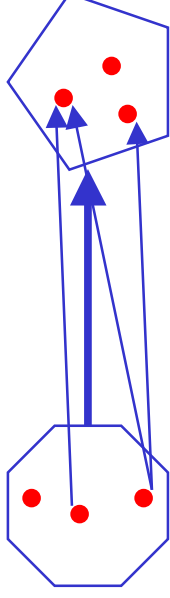
Wo spalten?

Bisimulation:

gut



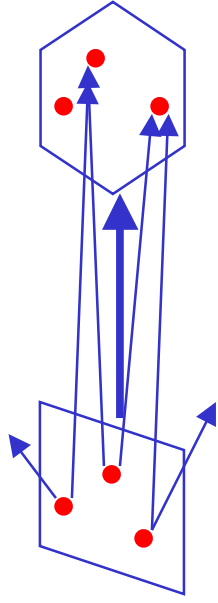
schlecht



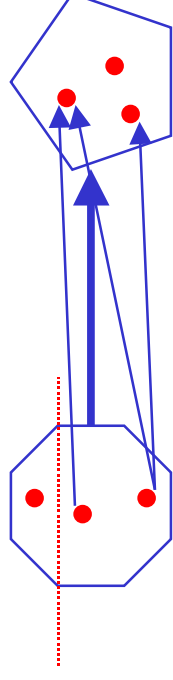
Wo spalten?

Bisimulation:

gut



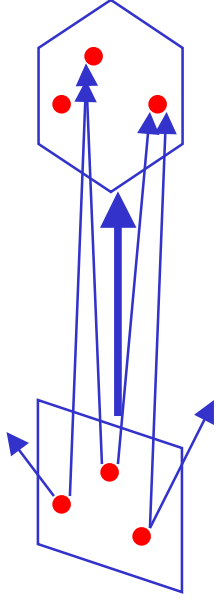
schlecht



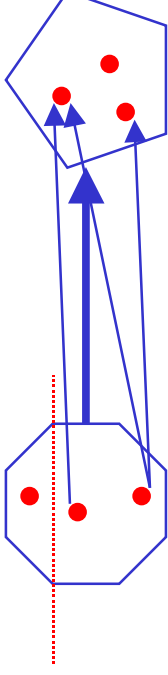
Wo spalten?

Bisimulation:

gut

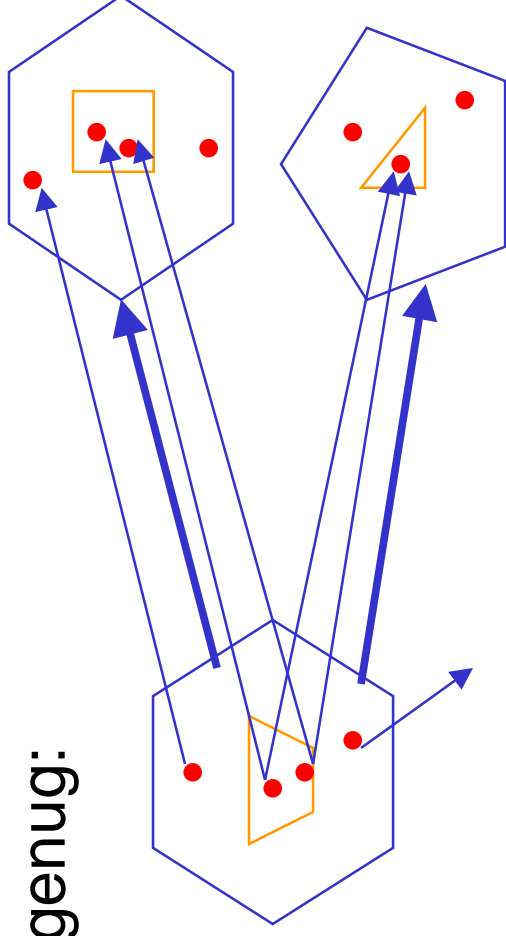


schlecht



Bewahrung von ACTL* in die "andere" Richtung:

gut genug:



"Core"

Core muss sich wie
Bisimulation verhalten

Wie spalten?

Beispiel: Zonen

Welche Uhrenstellung in Z1 hat Nachfolger in Z2?

Wie spalten?

Beispiel: Zonen

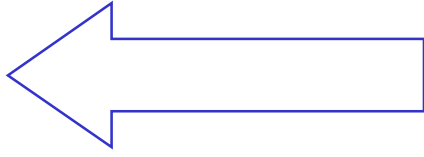
Welche Uhrenstellung in Z1 hat Nachfolger in Z2?

1. Schnitt mit Constraint
2. Nullstellen
3. Schnitt mit Invariante des diskreten Nachfolgers
4. Zeitverlauf
5. Schnitt mit Invariante

Wie spalten?

Beispiel: Zonen

Welche Uhrenstellung in Z1 hat Nachfolger in Z2?

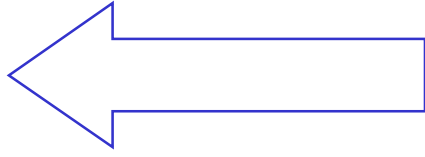


1. Schnitt mit Constraint
2. Nullstellen
3. Schnitt mit Invariante des diskreten Nachfolgers
4. Zeitverlauf
5. Schnitt mit Invariante

Wie spalten?

Beispiel: Zonen

Welche Uhrenstellung in Z1 hat Nachfolger in Z2?



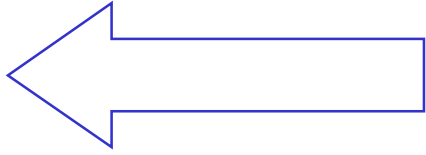
1. Schnitt mit Constraint
2. Nullstellen
3. Schnitt mit Invariante des diskreten Nachfolgers
4. Zeitverlauf
5. Schnitt mit Invariante

1. Schnitt mit Null
2. Öffnung "nach oben"
3. Schnitt mit Constraint
4. Schnitt mit Zone

Wie spalten?

Beispiel: Zonen

Welche Uhrenstellung in Z1 hat Nachfolger in Z2?



1. Schnitt mit Constraint
2. Nullstellen
3. Schnitt mit Invariante des diskreten Nachfolgers
4. Zeitverlauf
5. Schnitt mit Invariante

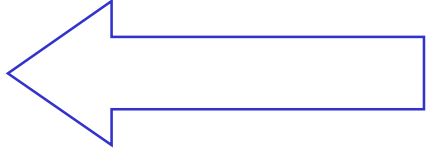
1. Schnitt mit Null
2. Öffnung "nach oben"
3. Schnitt mit Constraint
4. Schnitt mit Zone

Komplement nicht unbedingt konvex
→ ggf. mehrere Zonen als Spaltprodukt

Wie spalten?

Beispiel: Zonen

Welche Uhrenstellung in Z1 hat Nachfolger in Z2?



1. Schnitt mit Constraint
2. Nullstellen
3. Schnitt mit Invariante des diskreten Nachfolgers
4. Zeitverlauf
5. Schnitt mit Invariante

1. Schnitt mit Null
2. Öffnung “nach oben”
3. Schnitt mit Constraint
4. Schnitt mit Zone

Komplement nicht unbedingt konvex

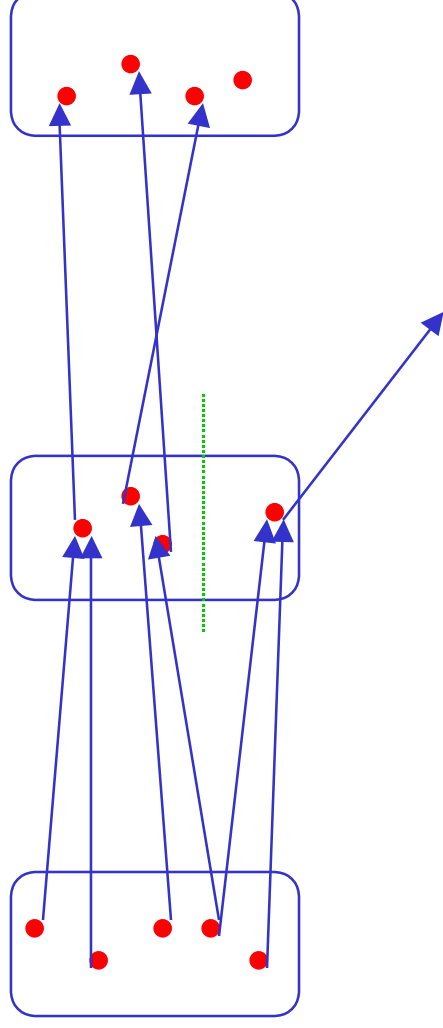
→ ggf. mehrere Zonen als Spaltprodukt

→ lernen: brauchen Möglichkeit,

abstrakten Vorgänger zu berechnen

Propagierung

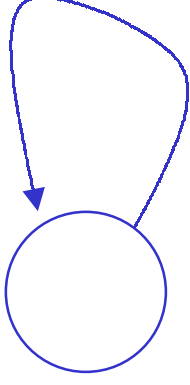
Spaltung eines abstrakten Zustandes kann weitere Spaltungen notwendig machen → Propagation nach rückwärts...



... bis sich nix mehr ändert

Konsequenteste Umsetzung

Beginne mit



Entwickle Transitionssystem ausschliesslich per Verfeinerung

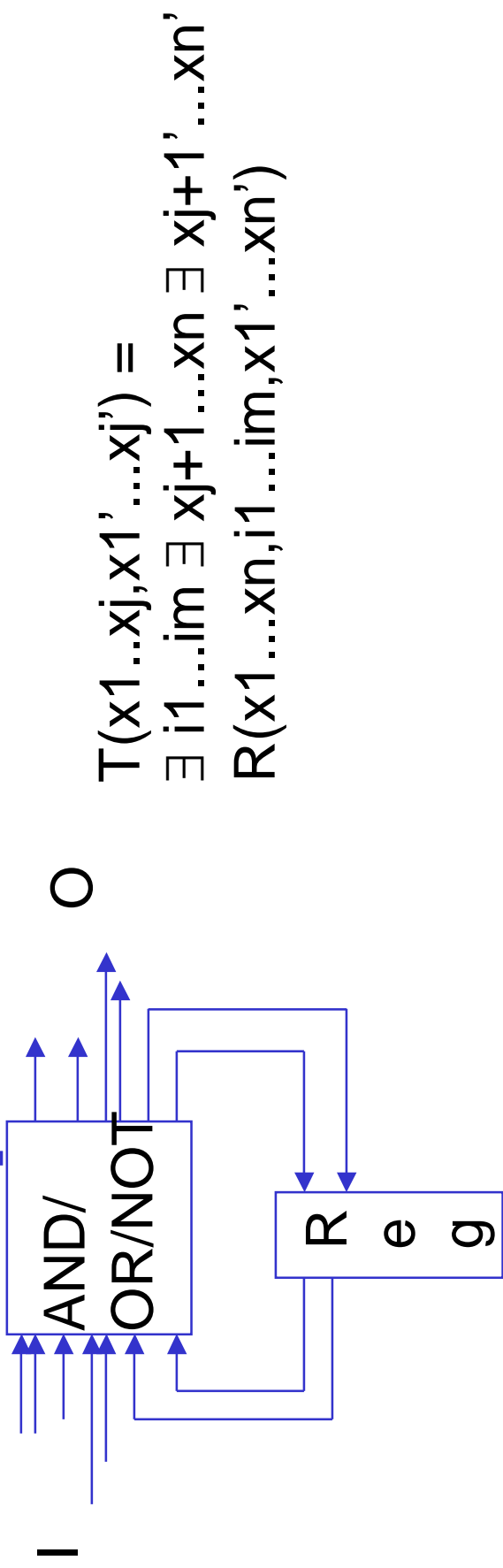
→ Splitting Tree: dokumentiert Spaltgeschichte
= Datenstruktur für Zustandsmenge
(Zustand = Pfad im Splitting Tree)

7.2 Gegenbeispielgesteuerte Verfeinerung

Algorithmus:

1. initiale Abstraktion
2. Verifikation von ϕ (aus ACTL*) im abstrakten System
3. ϕ gilt $\rightarrow \phi$ gilt auch im konkreten System
4. ϕ gilt nicht \rightarrow abstraktes Gegenbeispiel
5. Prüfe, ob Gegenbeispiel im konkreten System realisierbar ist
6. ja $\rightarrow \phi$ gilt nicht
7. nein \rightarrow verfeinere Abstraktion derart, daß Gegenbeispiel auch im abstrakten System unrealisierbar wird
8. GOTO 2

Beispiel: Bit-Abstraktion



$$T(x_1 \dots x_j, x_1' \dots x_j') = \\ \exists i_1 \dots i_m \exists x_{j+1} \dots x_n \exists x_{j+1}' \dots x_n' \\ R(x_1 \dots x_n, i_1 \dots i_m, x_1' \dots x_n')$$

initiale Abstraktion: möglichst wenige Variablen sichtbar
z.B. die in der Formel vorkommenden

Existential Abstraction (Abstraktion ist Funktion)

Verifikation mit BDD- oder SAT-basiertem Model Checking³⁰

Abstraktes Gegenbeispiel

einfachster Fall: Gegenbeispiel = endl. Pfad

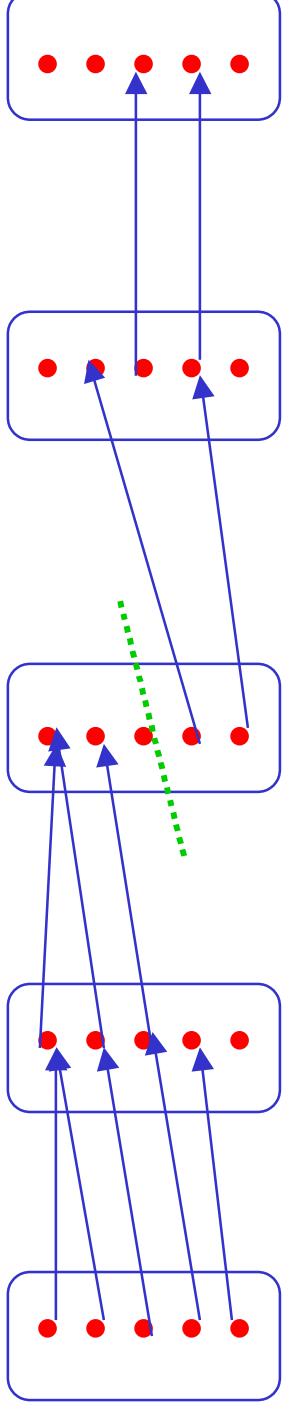
= Folge von Belegungen der sichtbaren Variablen $c1^{(k)} .. cj^{(k)}$

Realisierbarkeit im konkreten System =
Erfüllbarkeit von

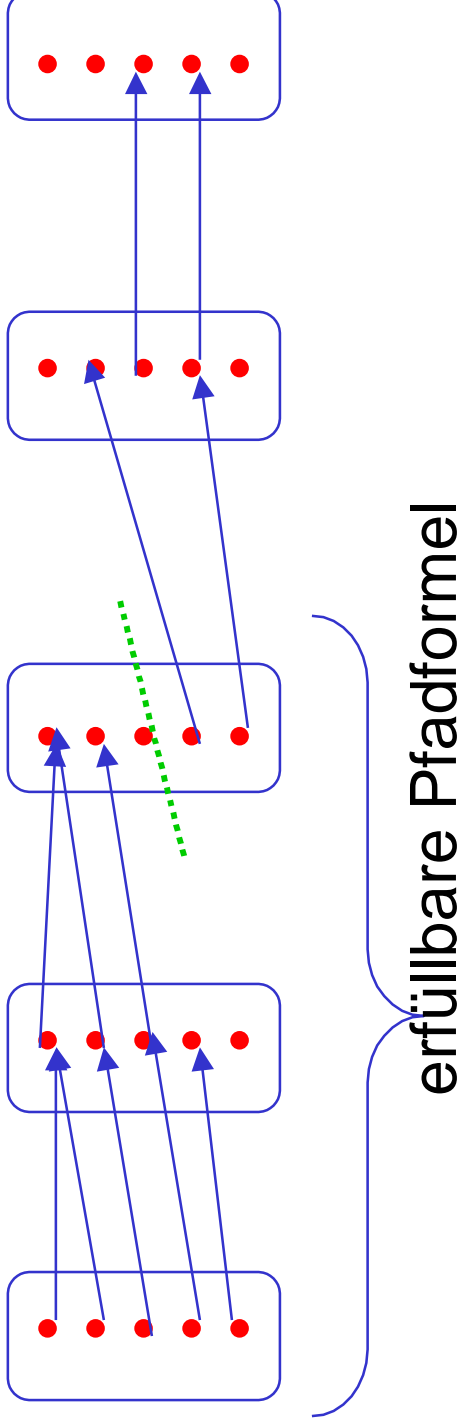
$$\begin{aligned} & I(c1^{(0)} \dots cj^{(0)}, x(j+1)^{(0)} \dots x(n)^{(0)}) \wedge \\ & T(c1^{(0)} \dots cj^{(0)}, x(j+1)^{(0)} \dots x(n)^{(0)}, c1^{(1)} \dots cj^{(1)}, x(j+1)^{(1)} \dots x(n)^{(1)}) \wedge \\ & \dots \end{aligned}$$

→ SAT-Checker

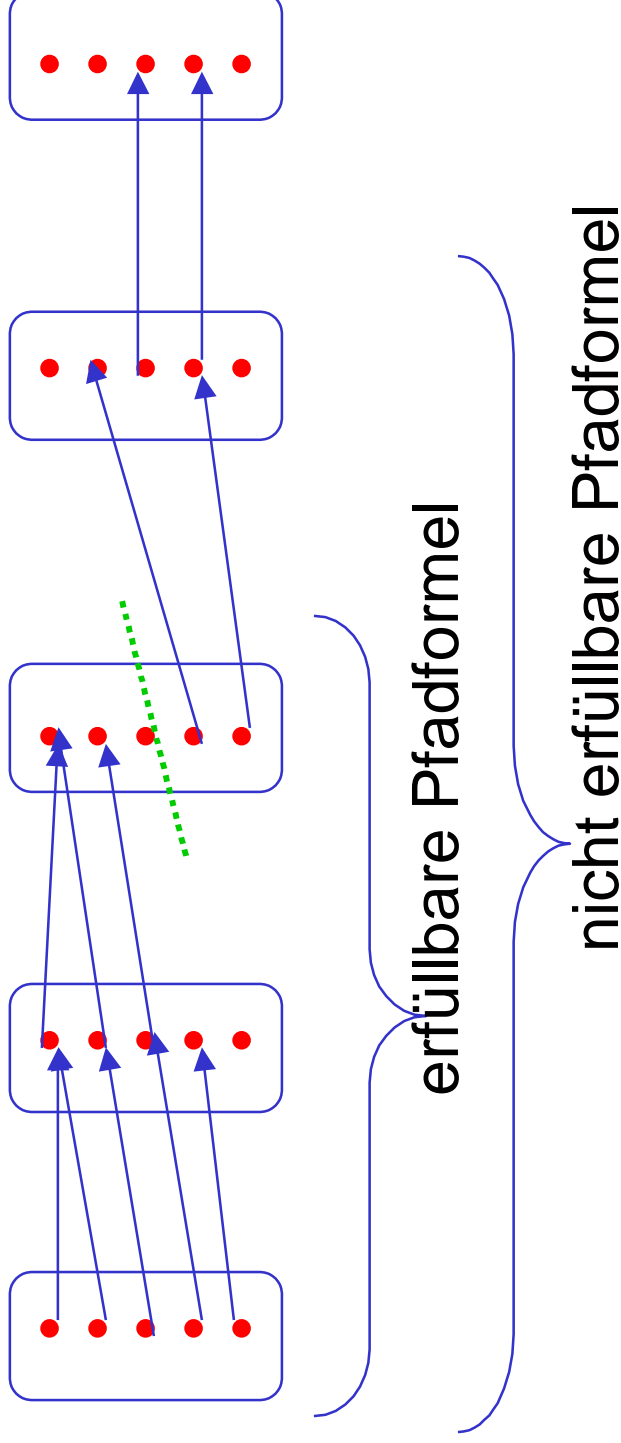
Scheingegenbeispiele



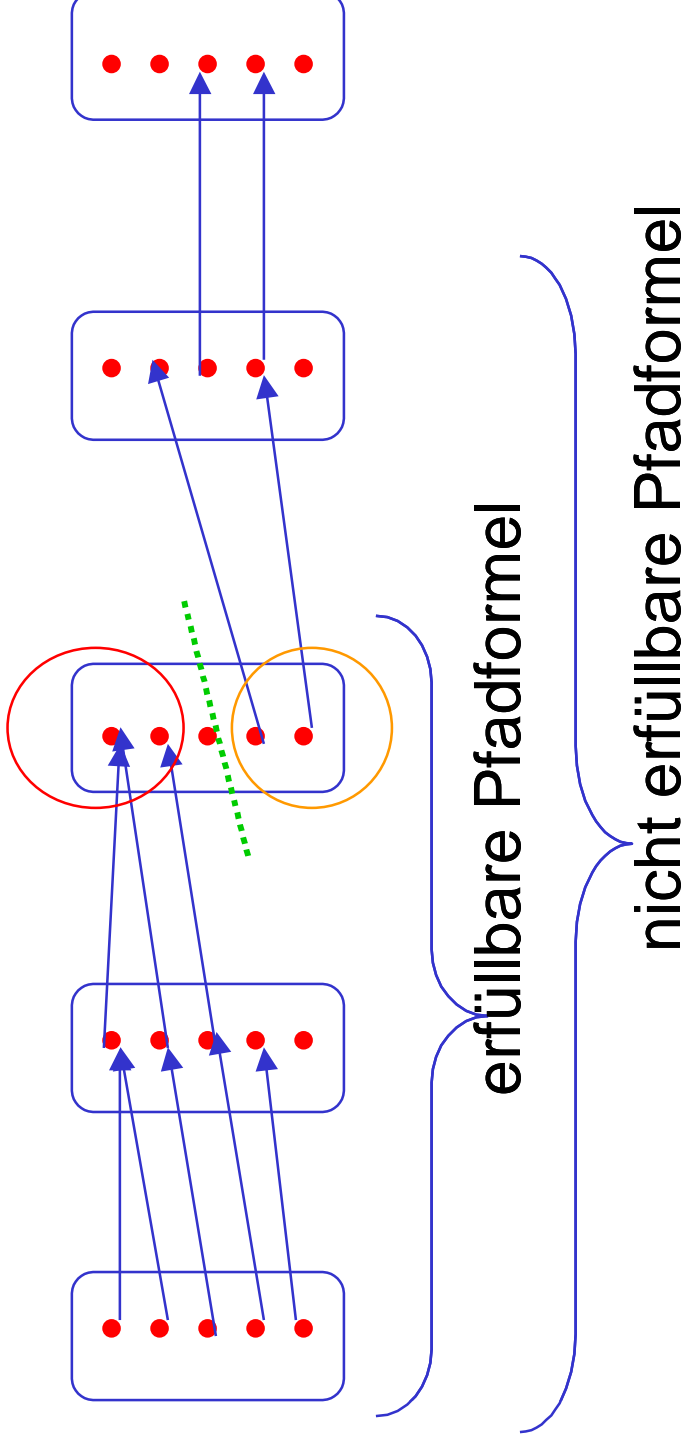
Scheingegenbeispiele



Scheingegenbeispiele

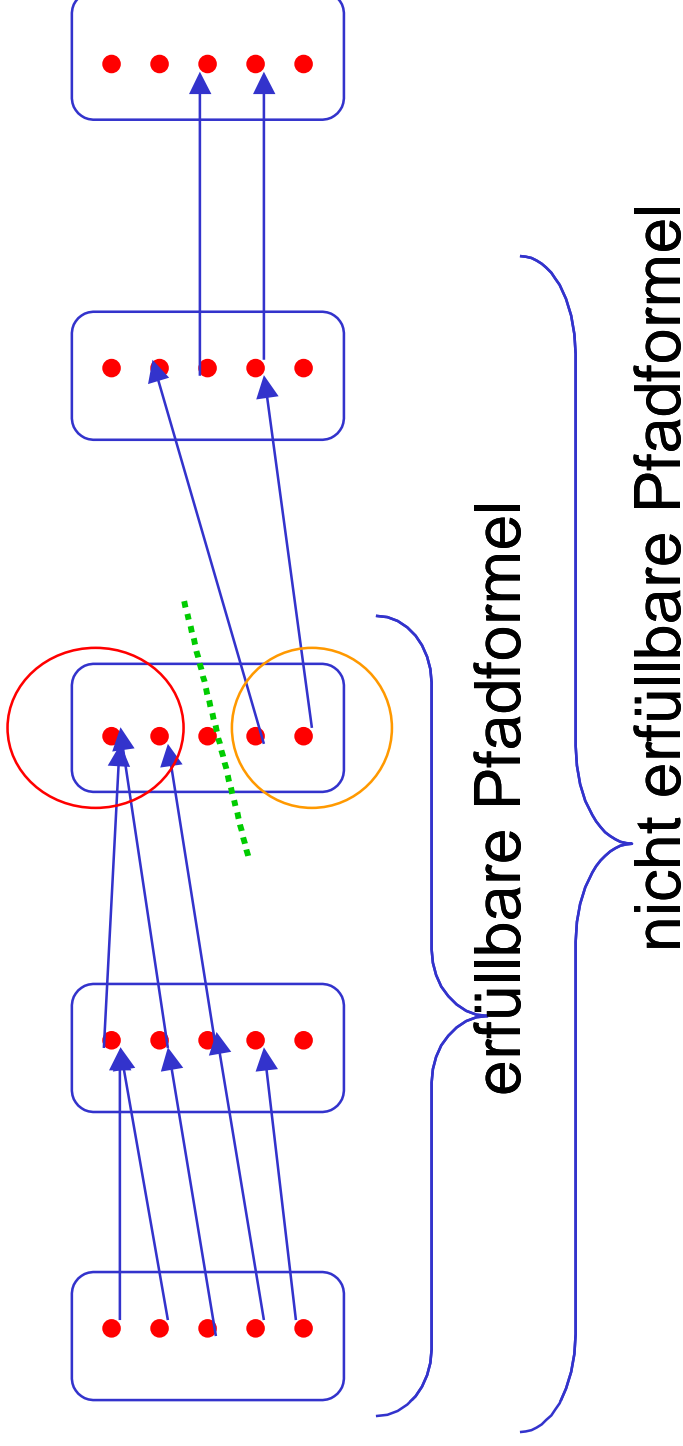


Scheingegenbeispiele



dead end state = erreichbar von einem konkreten Zustand im
ersten abstrakten Zustand
bad state = hat Nachfolger

Scheingegenbeispiele



dead end state = erreichbar von einem konkreten Zustand im
ersten abstrakten Zustand
bad state = hat Nachfolger

Verfeinerung = trenne dead end und bad states