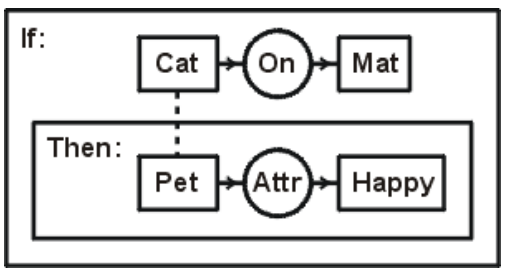
Conceptual Graphs

Patricia Naumann

Heinrich Heine Universität Düsseldorf

Wiebke Petersen: Frametheorie

3. Juli 2012



Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Was sind Conceptual Graphs?

- Formale Sprache zur Repräsentation von Wissen und Bedeutung am Computer

Conceptual Graphs als Teil der Wissensbasis

- Conceptual graphs
- Typhierarchie
- Relationshierarchie
- Individuenkatalog

Außerdem gibt es Lambda-Ausdrücke, Darstellung von Koreferenz, Kontexte und eingebettete Graphen. Lokatoren geben Individuen durch Individuenmarker an. Indexicals zeigen auf Referenten, die aus dem Kontext erschlossen werden.

Woraus bestehen Conceptual Graphs

```
[{*}]->(Attr)->[Conceptual]
```

Conceptual Graphs haben zwei Sorten von Knoten:

- Konzepte
- Relationen

Zwischen den Knoten gibt es gerichtete Arcs.

- Conceptual Graphs sind bipartite Graphen: Jeder Arc geht von Konzept zu Relation oder umgekehrt, aber nicht von Konzept zu Konzept oder Relation zu Relation.
- Jeder Arc gehört zu einer Relation und ist an ein Konzept gebunden.
- Konzepte können ohne Arcs und Relationen stehen (Bsp. [God]). Diese heißten Singletons.
- Relationen müssen immer über Arcs mit Konzepten verbunden sein.
- Leerer Conceptual Graph: Ein Blank. Er sagt über nichts etwas aus.

Woraus bestehen Conceptual Graphs

```
[{*}]->(Attr)->[Conceptual]
```

Conceptual Graphs haben zwei Sorten von Knoten:

- Konzepte
- Relationen

Zwischen den Knoten gibt es gerichtete Arcs.

- Conceptual Graphs sind bipartite Graphen: Jeder Arc geht von Konzept zu Relation oder umgekehrt, aber nicht von Konzept zu Konzept oder Relation zu Relation.
- Jeder Arc gehört zu einer Relation und ist an ein Konzept gebunden.
- Konzepte können ohne Arcs und Relationen stehen (Bsp. [God]). Diese heißten Singletons.
- Relationen müssen immer über Arcs mit Konzepten verbunden sein.
- Leerer Conceptual Graph: Ein Blank. Er sagt über nichts etwas aus.

Woraus bestehen Conceptual Graphs

```
[{*}]->(Attr)->[Conceptual]
```

Conceptual Graphs haben zwei Sorten von Knoten:

- Konzepte
- Relationen

Zwischen den Knoten gibt es gerichtete Arcs.

- Conceptual Graphs sind bipartite Graphen: Jeder Arc geht von Konzept zu Relation oder umgekehrt, aber nicht von Konzept zu Konzept oder Relation zu Relation.
- Jeder Arc gehört zu einer Relation und ist an ein Konzept gebunden.
- Konzepte können ohne Arcs und Relationen stehen (Bsp. [God]). Diese heißten Singletons.
- Relationen müssen immer über Arcs mit Konzepten verbunden sein.
- Leerer Conceptual Graph: Ein Blank. Er sagt über nichts etwas aus.

Woraus bestehen Conceptual Graphs

```
[{*}]->(Attr)->[Conceptual]
```

Conceptual Graphs haben zwei Sorten von Knoten:

- Konzepte
- Relationen

Zwischen den Knoten gibt es gerichtete Arcs.

- Conceptual Graphs sind bipartite Graphen: Jeder Arc geht von Konzept zu Relation oder umgekehrt, aber nicht von Konzept zu Konzept oder Relation zu Relation.
- Jeder Arc gehört zu einer Relation und ist an ein Konzept gebunden.
- Konzepte können ohne Arcs und Relationen stehen (Bsp. [God]). Diese heißten Singletons.
- Relationen müssen immer über Arcs mit Konzepten verbunden sein.
- Leerer Conceptual Graph: Ein Blank. Er sagt über nichts etwas aus.

Woraus bestehen Conceptual Graphs

```
[{*}]->(Attr)->[Conceptual]
```

Conceptual Graphs haben zwei Sorten von Knoten:

- Konzepte
- Relationen

Zwischen den Knoten gibt es gerichtete Arcs.

- Conceptual Graphs sind bipartite Graphen: Jeder Arc geht von Konzept zu Relation oder umgekehrt, aber nicht von Konzept zu Konzept oder Relation zu Relation.
- Jeder Arc gehört zu einer Relation und ist an ein Konzept gebunden.
- Konzepte können ohne Arcs und Relationen stehen (Bsp. [God]). Diese heißten Singletons.
- Relationen müssen immer über Arcs mit Konzepten verbunden sein.
- Leerer Conceptual Graph: Ein Blank. Er sagt über nichts etwas aus.

Wie werden CGs gelesen?

Hinweis

`[A]->(R)->[B]` und `[B]<-(R)<-[A]` sind äquivalent.

Das heißt...?

Weitere Beispiele:

```
[Rhino: Otto]->(Attr)->[Orange]
```

```
[Person: Julia]<-(Betw)-
    <-1-[Person: Tom]
    <-2-[Person: Brad]
```

Wie werden CGs gelesen?

Hinweis

`[A]->(R)->[B]` und `[B]<-(R)<-[A]` sind äquivalent.

Das heißt...?

Weitere Beispiele:

```
[Rhino: Otto]->(Attr)->[Orange]
```

```
[Person: Julia]<-(Betw)-
<-1-[Person: Tom]
<-2-[Person: Brad]
```

Wie werden CGs gelesen?

Hinweis

`[A]->(R)->[B]` und `[B]<-(R)<-[A]` sind äquivalent.

Das heißt...?

Weitere Beispiele:

```
[Rhino: Otto]->(Attr)->[Orange]
```

```
[Person: Julia]<-(Betw)-
    <-1-[Person: Tom]
    <-2-[Person: Brad]
```

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Konzepte

- Konzepttyp
- Referent

Konzepte sind der folgenden Form nach aufgebaut: `[Type: Referent]`
Der Referent kann weggelassen werden, der Typ aber nie.

Beispiel: `[Bus]->(Dest)->[City: Aalborg]`

- Typen sind Labels oder Namen für Gruppen von Entitäten, die ähnliche Eigenschaften haben.
- Typen stehen in einer Hierarchie, sodass Typen auch Subtypen haben können. In dieser Hierarchie heißt das TOP-Element „Entity“ und das BOTTOM-Element „Absurdity“.

Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

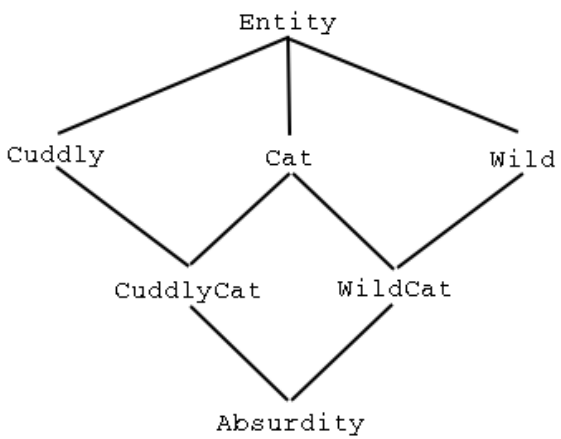
Relationen

- Beschreiben wie Konzepte zueinander stehen, also Beispielsweise örtliche Relationen, Besitzrelationen, Eventbeschreibungen etc.
- Relationen haben ebenfalls einen Typ, aber keinen Referenten. Ebenso beschreibt die Valenz einer Relation, mit wievielen Konzepten sie über Arcs verbunden ist. Die Valenz ist immer ≥ 1.
- Die Signatur bestimmt, welche Arten von Konzepten zu der Relation verbunden werden können. Signaturen sind geordnete Listen, die Typen beinhalten. Sie sind von der Form <t1, t2, ..., tn>. Die Reihenfolge in der Signatur ist entscheidend für die Richtung der Arcs.
- Die ersten n-1 Konzepte zeigen zur Relation hin und das n-te Konzept zeigt von der Relation weg.

Beispiel: `Agnt` (agent) mit der Signatur `<Act,Animate>`

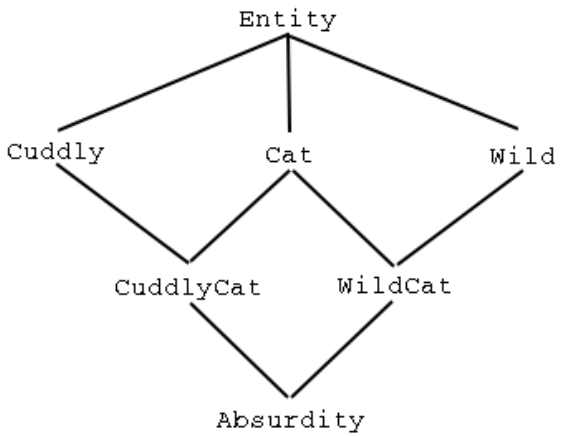
Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



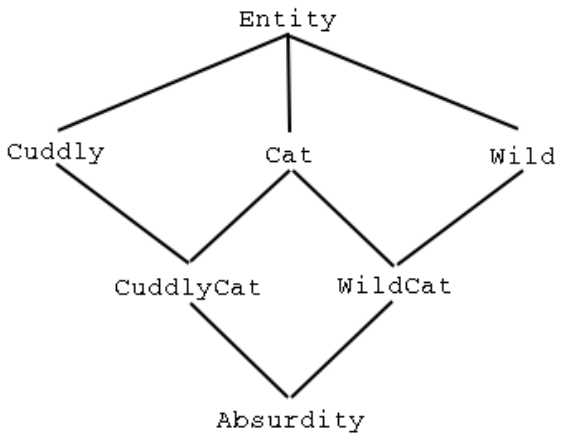
Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



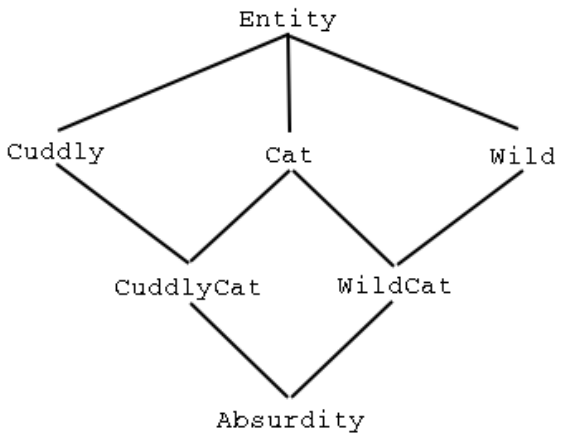
Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



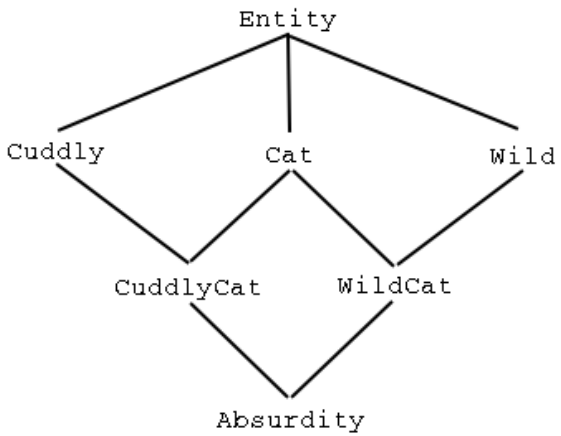
Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



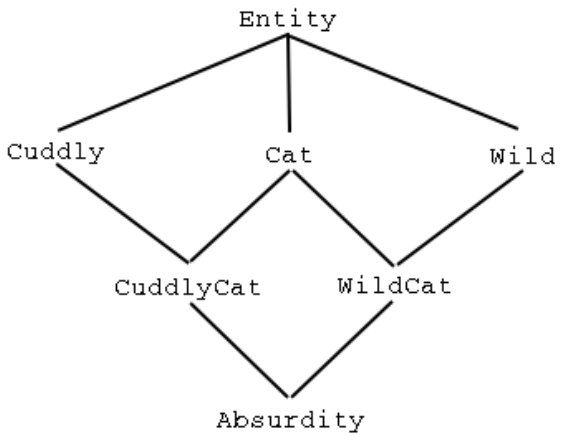
Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



Typenhierarchie

- Die Typen sind in einer partiellen Ordnung, genannt Subtyprelation. A $\leq$ B heißt, dass A Subtyp von B ist. A ist entweder B oder spezieller als B. Die Subtypenrelation ist transitiv.
- Entity: Der Universaltyp, der über allen anderen Typen steht. Er ist so allgemein, dass er über nichts etwas aussagt.
- Absurdity: Der absurde Typ steht unter allen anderen Typen. Es gibt keine Instanz von diesem Typ.
- Eigenschaften werden an die Subtypen vererbt.



Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, ∀,
  - Die „unspezifizierte Menge“, „{*}“, bedeutet „Plural“,
  - Quantitäten wie „@2“, „@18 Sekunden“, oder „@100 Gramm“,
  - Sammlungen wie „{Romeo, Juliet}“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“,
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „`#es`“, „`#du`“, „`#`“ (DEF), „`#42`“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, ∀,
  - Die „unspezifizierte Menge“, „{*}“, bedeutet „Plural“,
  - Quantitäten wie „@2“, „@18 Sekunden“, oder „@100 Gramm“,
  - Sammlungen wie „{Romeo, Juliet}“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, ∀,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Conceptual Graphs vs. Frames

- CGs sind atemporal, das heißt Tempus und Aspekt können mit ihnen nicht ausgedrückt werden. Es können Subgraphen gebildet werden, die ausgehend von Konzepten wie beispielsweise (Past) sind.
- Lambda-Ausdrücke ergänzen Relationen und Konzepte und können Wissen von außerhalb einbeziehen, beispielsweise bei der Modellierung von Paaren etc.
- Mit CGs werden ganze Statements und Sachverhalte beschrieben. Mit Frames wird unsere Vorstellung von der Welt dargestellt. Konzepte können im Gegensatz zu Frames quantifiziert werden, gezählt werden oder Mengen bilden.
- Designatoren und Lokatoren erhöhen die Ausdruckskraft:
  - Allquantor, $\forall$,
  - Die „unspezifizierte Menge“, „`{*}`“, bedeutet „Plural“,
  - Quantitäten wie „`@2`“, „`@18 Sekunden`“, oder „`@100 Gramm`“.
  - Sammlungen wie „`{Romeo,Juliet}`“
  - Definitheit mit Indexicals „#es“, „#du“, „#“ (DEF), „#42“.

Weitere Beispiele

`[Sing]->(Agnt)->[Bird: {*}]->(In)->[SycamoreTree]`

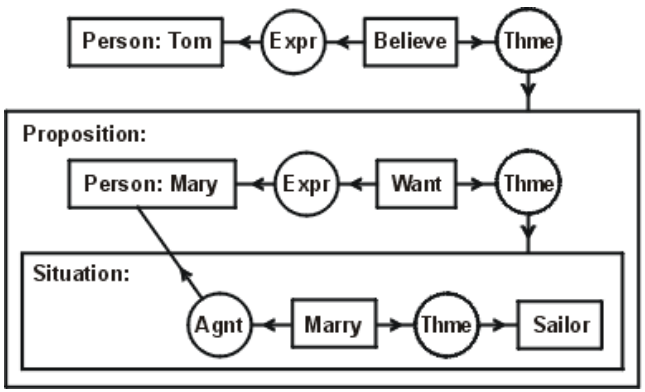
„Birds singing in a sycamore tree“

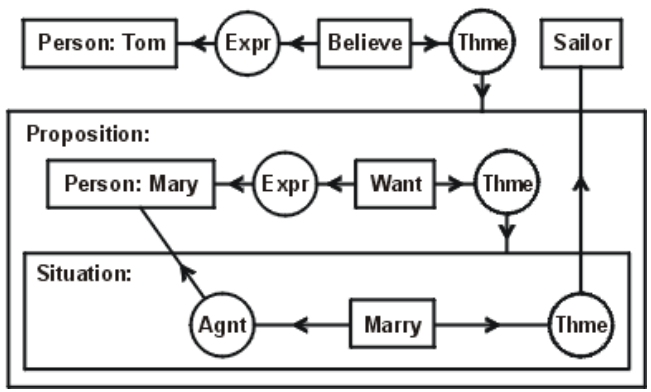
`[LivingFish: ∀]->(Attr)->[Wet]`

„All living fish are wet“

`[Person: Arthur]<-(Agnt)<-[Brew]->(Rslt)->[CupOfTea: #42]`

„Arthur brews cup of tea ’42’“





Referenzen

- http://cg.huminf.aau.dk/Module_I
- http://www.jfsowa.com/cg/cgif.htm
- Sowa, John: *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA 1984.
- Sowa, John: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.