A High Performance Semantic Web Query Answering Engine

Description Logic Workshop '05

Michael Wessel and Ralf Möller

Software Systems Group (STS)
Hamburg University of Technology (TUHH)
Hamburg, Germany
`{mi.wessel | r.f.moeller}@tuhh.de`

Overview of Talk

- Background & Requirements
- The nRQL Query Language
  - Introductory examples
  - Syntax (query atoms, variables, . . .)
  - Querying OWL Documents
  - Semantics
- The nRQL Query Processing Engine
  - Incremental Query Processing
  - Configurable Completeness
  - Simple Rules
  - Query Reasoning, Optimization
- Outlook & Conclusion

Background & Requirements

- Racer(Pro) is an ABox DL reasoner for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ aka $\mathcal{SHIQ}(\mathcal{D}^-)$
- Expressive ABox queries demanded by users
- Question: how to design a query language which
  - satisfies these user requests
  - offers "full query access" to all Racer features (e.g., concrete domain)
  - allows to query OWL documents (datatype & annotation properties, . . .)
  - can be implemented efficiently
  - has a simple orthogonal syntax and semantics

$\Rightarrow$ nRQL has been "tailored" for Racer

. . . what's new since DL '04

- more language features (e.g., a projection operator)
- serious implementation of query answering engine (concurrency control, thread pooling, . . .)
- (preliminary) GUI support
  - tools for manipulating and inspecting the states of queries (RacerPorter)
  - life cycle management of queries
- (very) simple rules
- additional representation layers for Racer for storing data (rationale? see below)
- nRQL access to these additional layers

nRQL Language – Overview

- compositional syntax and semantics
- compound/complex queries build from query atoms, using boolean connectors
- allows for arbitrary shaped conjunctive queries
- Query atoms contain variables and individuals
- variables: `?x`, `$?x`; individuals: `betty`

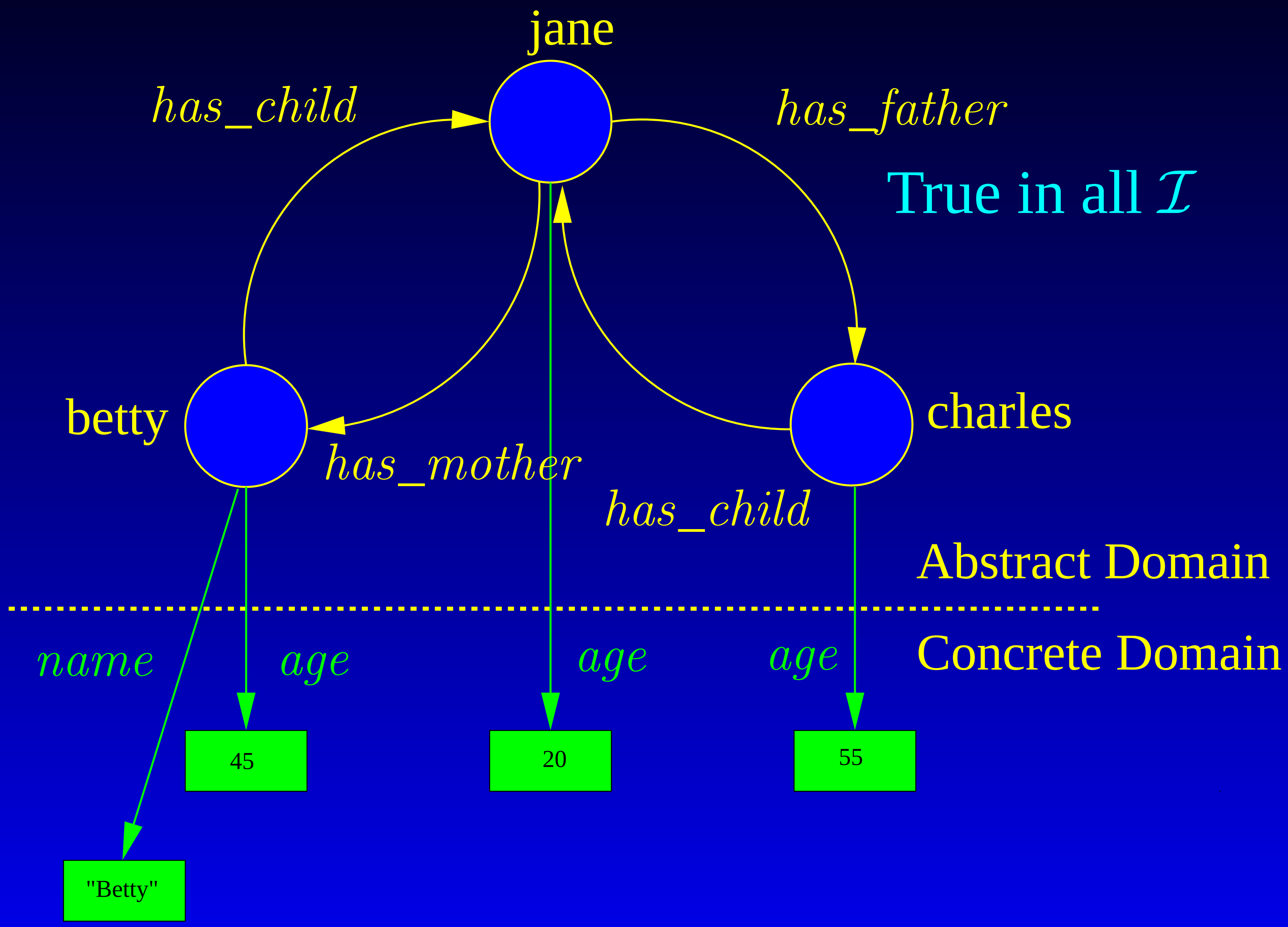
```
(retrieve (?x (told-value (age ?x)))
          (and (?x (and woman (an age)))
               (?x ?y has-child)
               (?y ?y (constraint
                          (has-father age)
                          (has-mother age)
                          (> age-1 (+ age-2 8))))))
```

nRQL Language – Overview

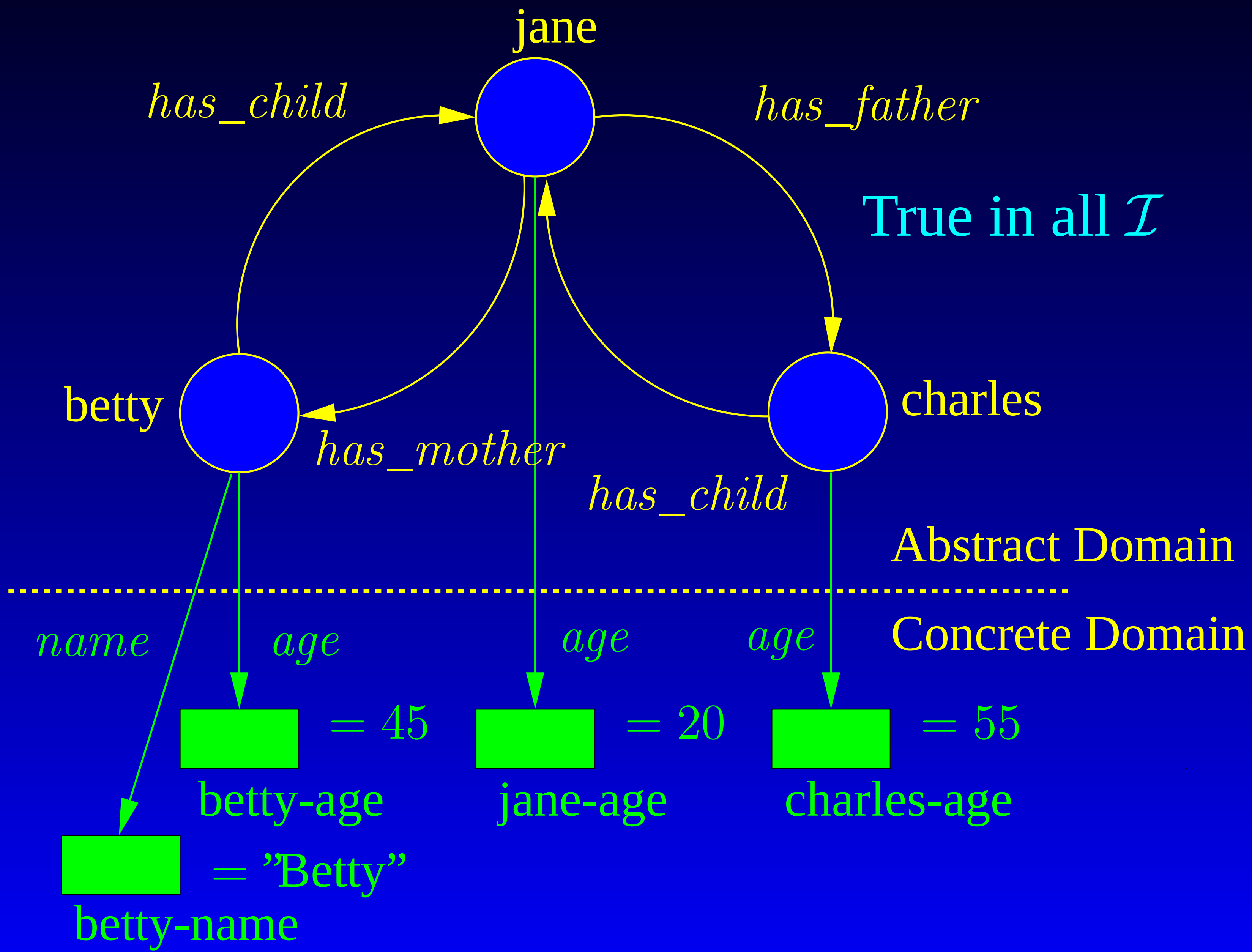
- compositional syntax and semantics
- compound/complex queries build from query atoms, using boolean connectors
- allows for arbitrary shaped conjunctive queries
- Query atoms contain variables and individuals
- variables: `?x`, `$?x`; individuals: `betty`

```
(((?x betty) ((told-value (age ?x)) 45))
 ((?x diana) ((told-value (age ?x)) 55))
      ⋮
       )
```

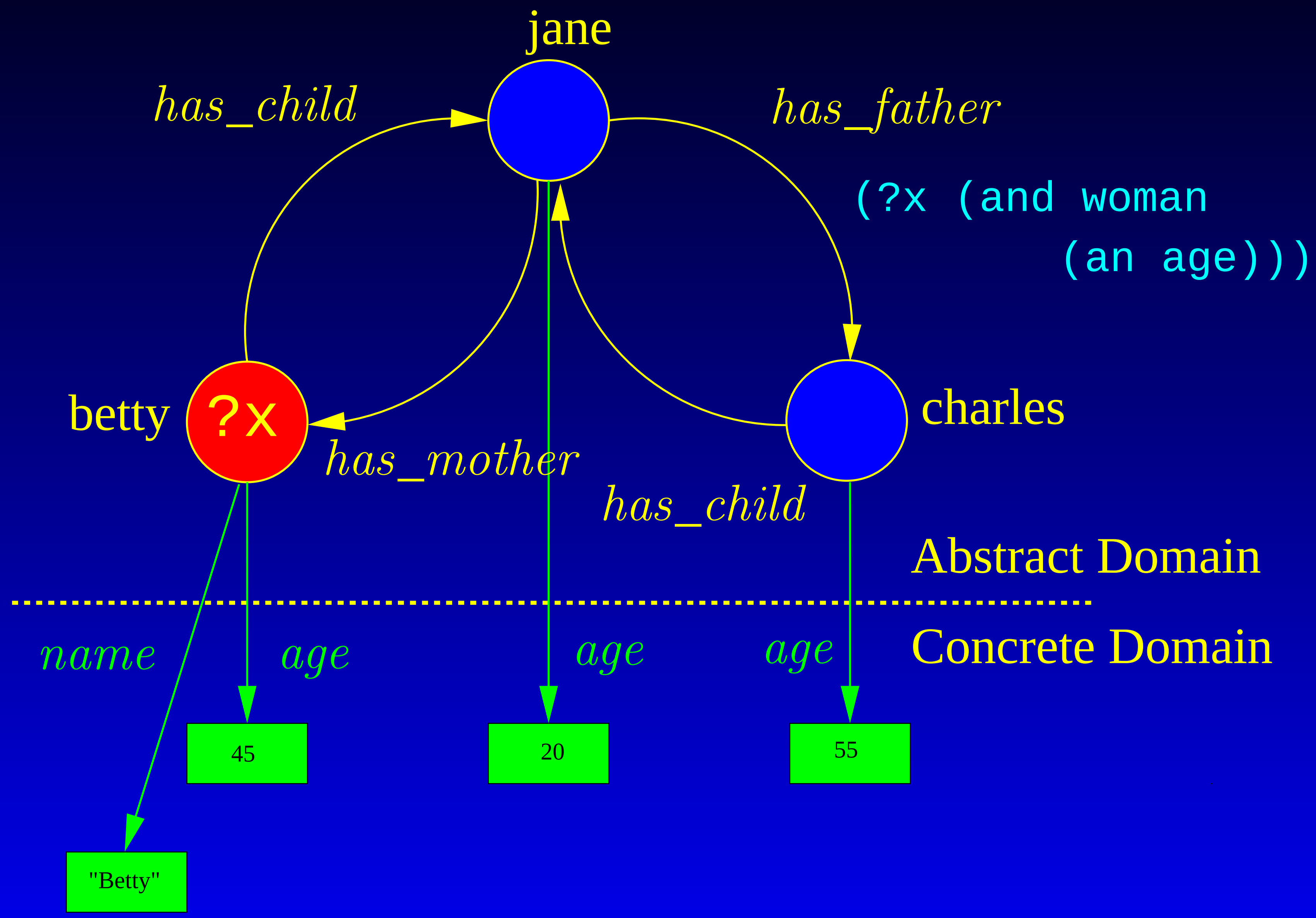
nRQL Language – Example



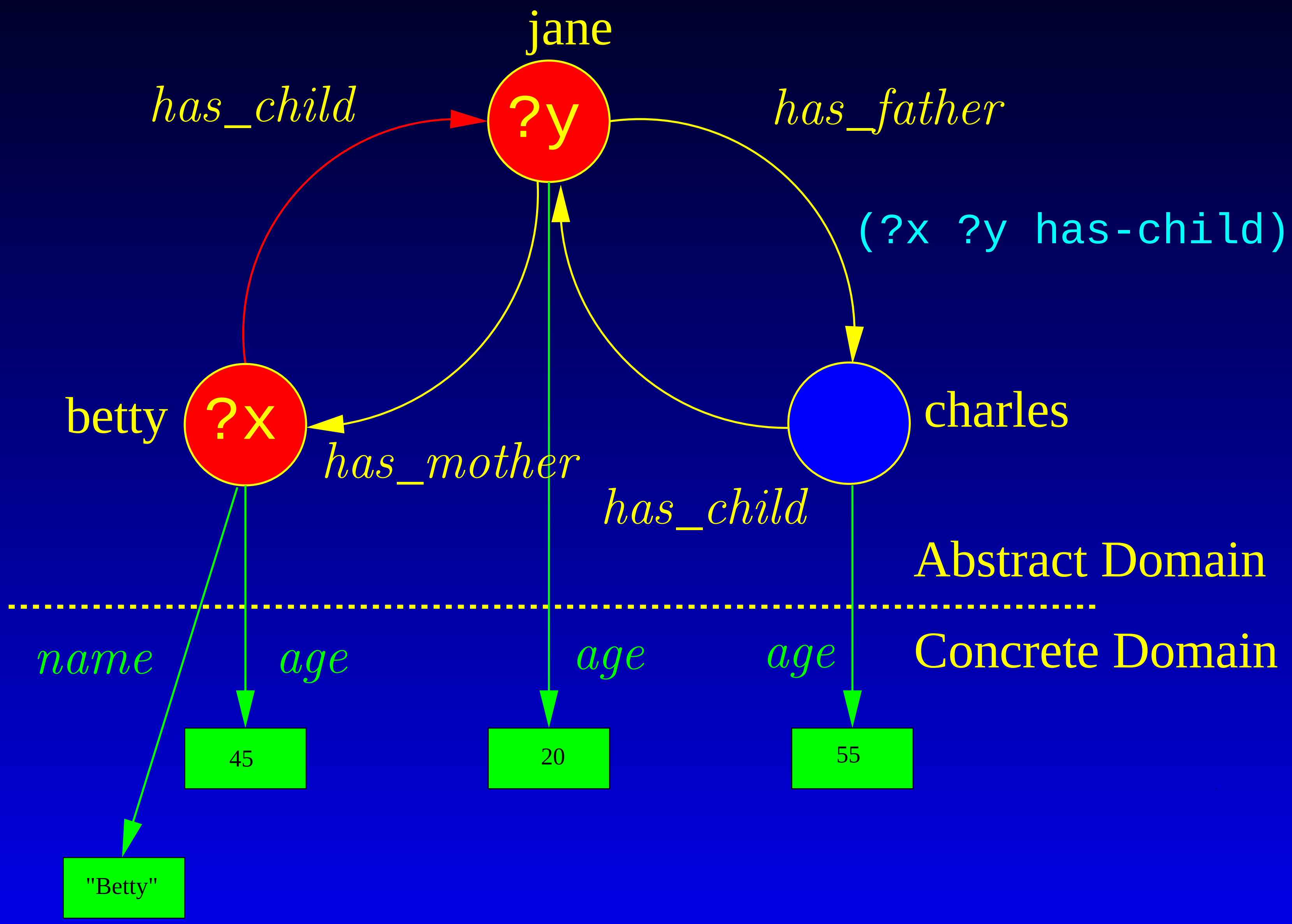
nRQL Language – Example



nRQL Language – Example

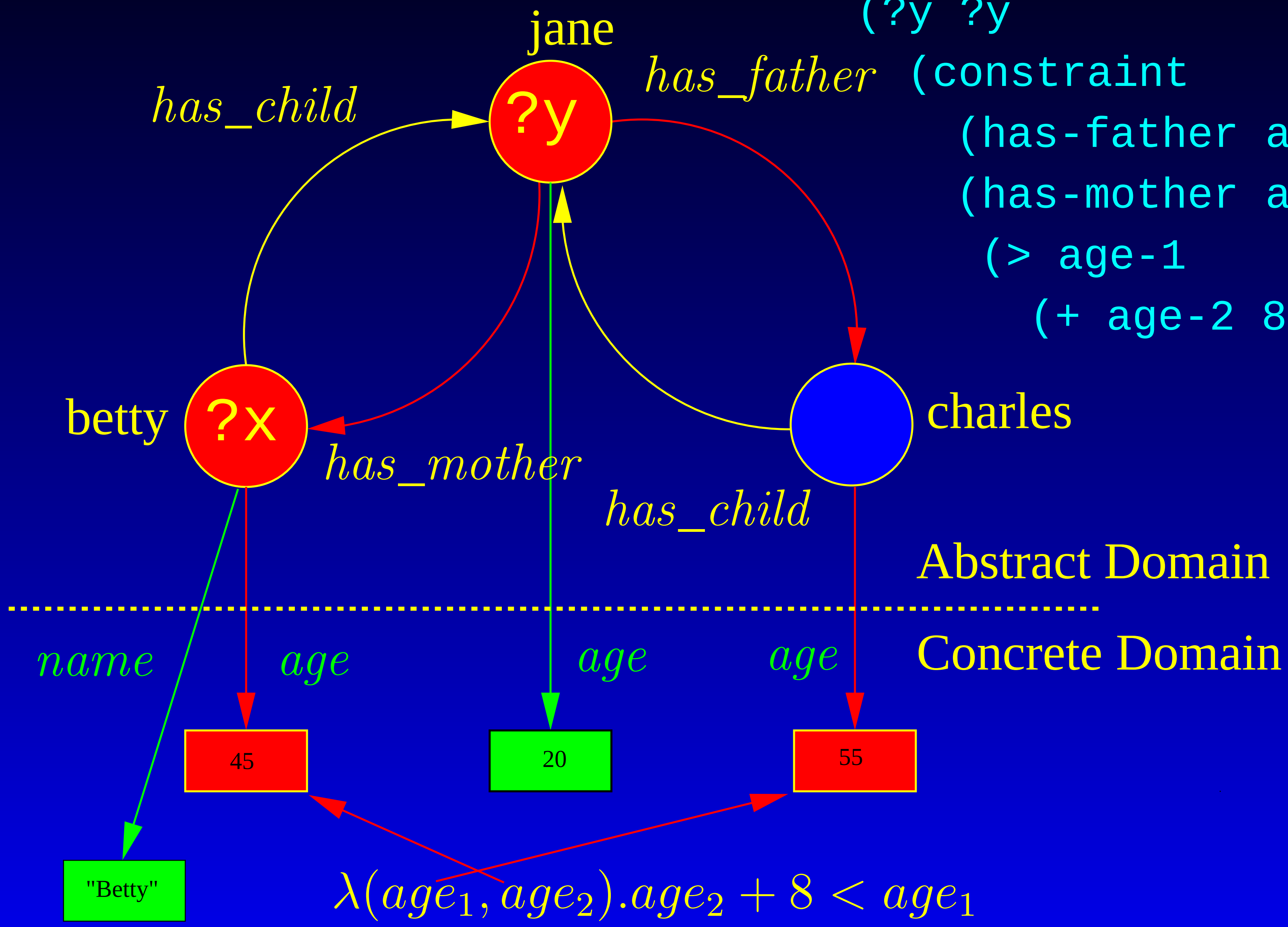


nRQL Language – Example



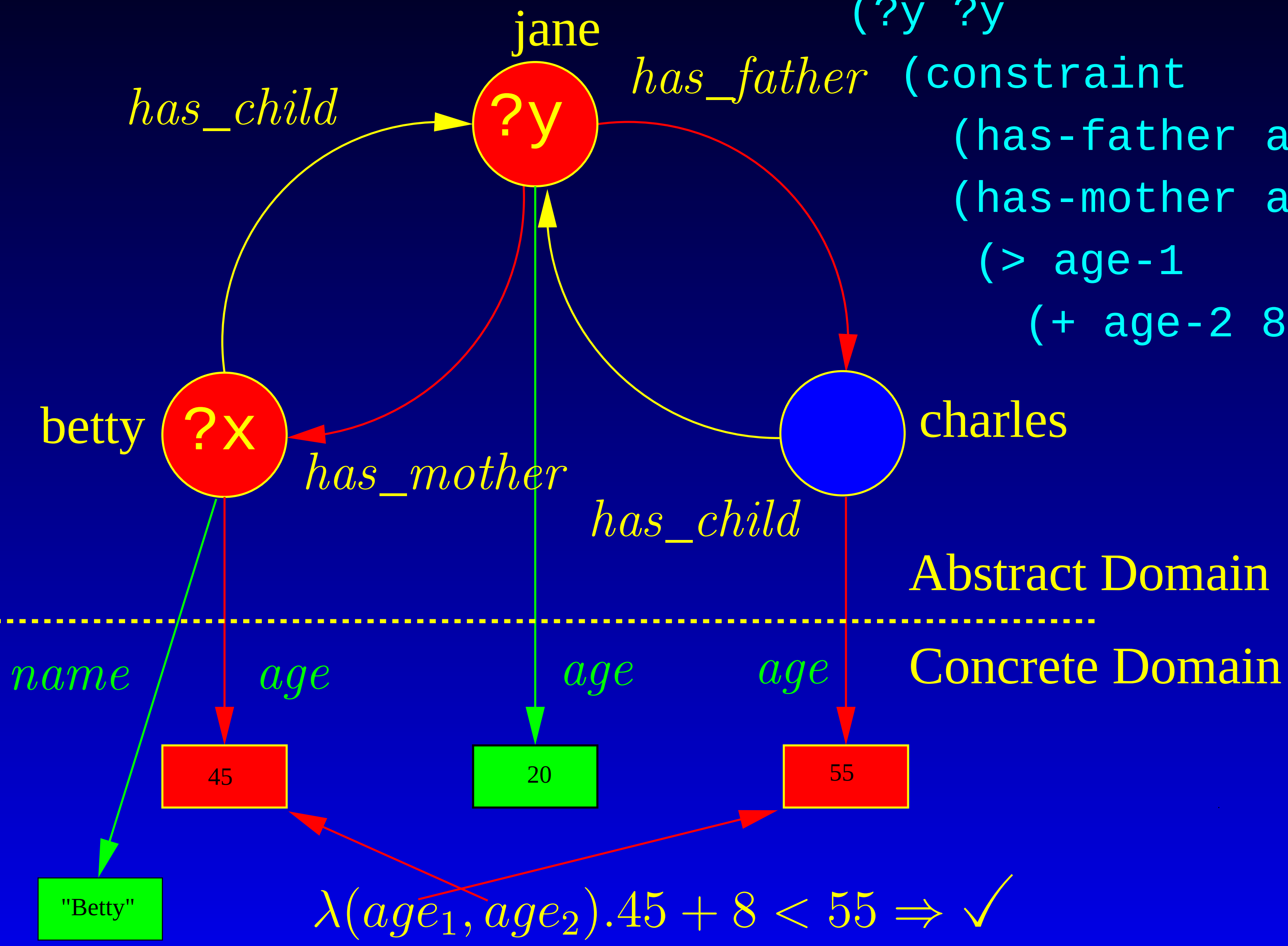
nRQL Language – Example

```
(?y ?y
 (constraint
   (has-father age)
   (has-mother age)
    (> age-1
      (+ age-2 8))))
```

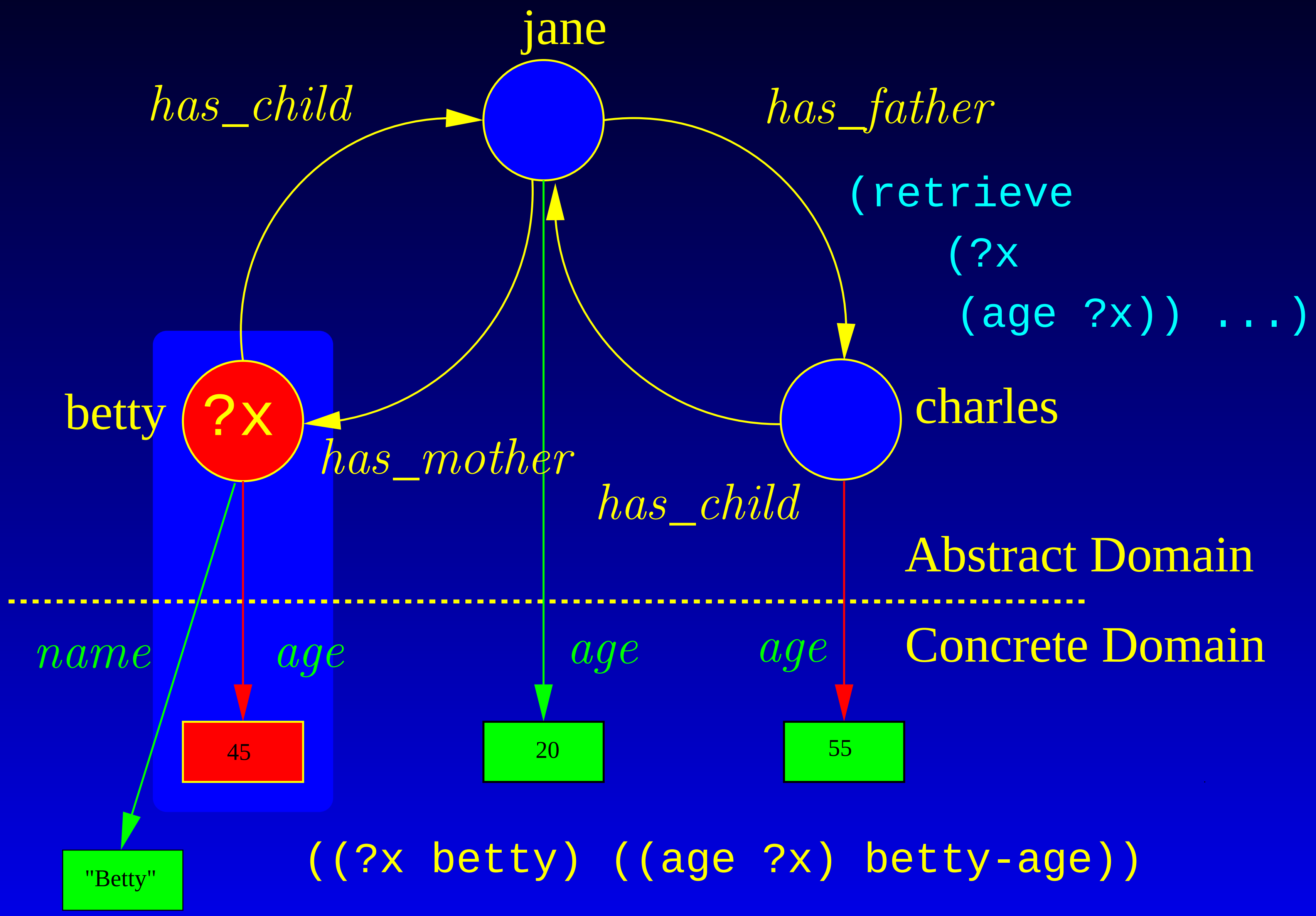


nRQL Language – Example

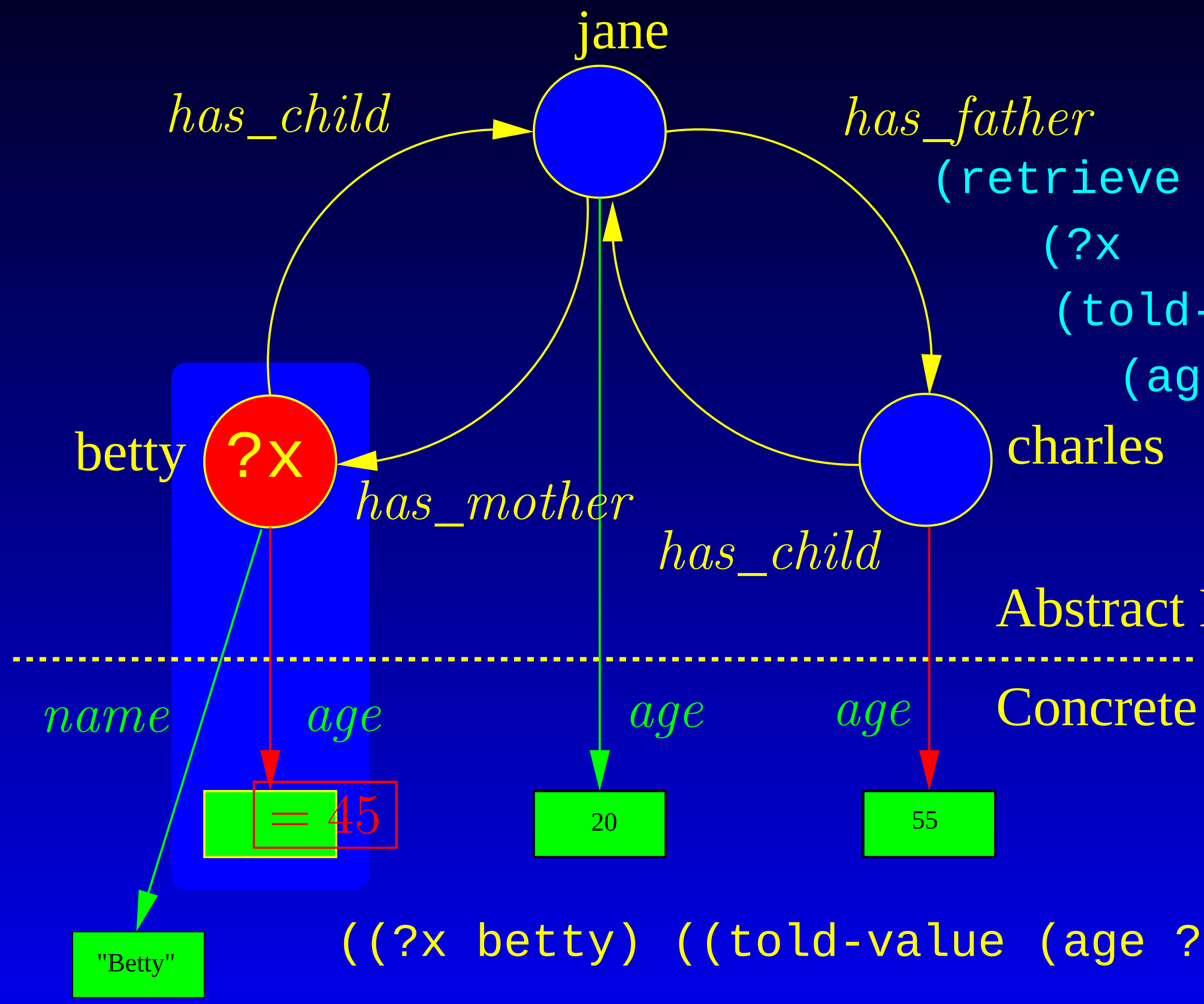
```
(?y ?y
 (constraint
  (has-father age)
  (has-mother age)
   (> age-1
     (+ age-2 8))))
```



nRQL Language – Example



nRQL Language – Example




```
(retrieve
   (?x
    (told-value
      (age ?x))) ...)
```

```
((?x betty) ((told-value (age ?x)) 45))
```

nRQL Language – Syntax

- Queries have a head and a body:
  `(retrieve <head> <body>)`
- Syntax for `<head>`

$$
\begin{array}{rcl}
\mathit{head} & := & (\mathit{head\_entry}^{*}) \\
\mathit{head\_entry} & := & \mathit{object} \mid \mathit{head\_projection\_operator} \\
\mathit{object} & := & \mathit{variable} \mid \mathit{individual} \\
\mathit{variable} & := & \text{a symbol beginning with “?”} \\
\mathit{individual} & := & \text{a symbol} \\
\mathit{head\_projection\_operator} & := & \\
 & & (\mathit{cd\_attribute}\ \mathit{object}) \mid \\
 & & (\texttt{told-value}\ (\mathit{cd\_attribute}\ \mathit{object})) \mid \\
 & & (\texttt{told-value}\ (\mathit{datatype\_property}\ \mathit{object})) \mid \\
 & & (\texttt{annotations}\ (\mathit{annotation\_property}\ \mathit{object}))
\end{array}
$$

nRQL Language – Syntax (2)

- Syntax for `<body>`

nRQL Language – Syntax (2)

- Syntax for <body>

$$body \quad := \quad atom \mid (\{\texttt{and} \mid \texttt{union}\} \; body^*) \mid (\{ \texttt{neg} \mid \texttt{inv}\} \; body) \mid$$
$$(\texttt{project-to} \; (object^*) \; body)$$

nRQL Language – Syntax (2)

- Syntax for <body>

$$body \;:=\; atom \mid (\{\texttt{and} \mid \texttt{union}\}\ body^*) \mid (\{\texttt{neg} \mid \texttt{inv}\}\ body) \mid$$
$$(\texttt{project-to}\ (object^*)\ body)$$

nRQL Language – Syntax (2)

- Syntax for <body>

$$
\begin{aligned}
body \;\; := \;\; & atom \mid (\,\{\texttt{and} \mid \texttt{union}\}\; body^*) \mid (\,\{\,\texttt{neg} \mid \texttt{inv}\}\; body\,) \mid \\
& (\texttt{project-to}\;\; (object^*)\; body)
\end{aligned}
$$

nRQL Language – Syntax (2)

- Syntax for <body>

$$body \quad := \quad atom \mid (\,\{\texttt{and} \mid \texttt{union}\}\ body^*) \mid (\,\{\,\texttt{neg} \mid \texttt{inv}\}\ body\,) \mid$$
$$(\texttt{project-to}\ \ (object^*)\ body)$$

nRQL Language – Syntax (2)

- Syntax for `<body>`

$body \;:=\; atom \mid (\,\{$`and`$\mid$`union`$\}\; body^{*}) \mid (\,\{$`neg`$\mid$`inv`$\}\; body\,) \mid$
$\quad ($`project-to` $(object^{*})\; body)$

$atom \;:=\; (object\ concept\_expr) \mid (object\ object\ role\_expr) \mid$
$\quad (object\ object\ ($`constraint` $chain\ chain\ constraint\_expr)) \mid$
$\quad ($`same-as` $variable\ individual)$

$chain \;:=\; (role\_expr^{*}\ cd\_attribute)$

Example concept query atoms

- `(?x woman)`
- `(betty woman)`

nRQL Language – Syntax (2)

- Syntax for <body>

$body \;:=\; atom \mid (\{\texttt{and} \mid \texttt{union}\}\; body^*) \mid (\{\texttt{neg} \mid \texttt{inv}\}\; body) \mid$
$(\texttt{project-to}\; (object^*)\; body)$

$atom \;:=\; (object\; concept\_expr) \mid (object\; object\; role\_expr) \mid$
$(object\; object\; (\texttt{constraint}\; chain\; chain\; constraint\_expr)) \mid$
$(\texttt{same-as}\; variable\; individual)$

$chain \;:=\; (role\_expr^*\; cd\_attribute)$

Example role query atoms

- `(?x ?y has-child)`
- `(betty ?child-of-betty has-child)`
- `(?x ?y (inv has-child))`
- `(?x ?y (not has-father))`

nRQL Language – Syntax (2)

- Syntax for `<body>`

$$body \quad := \quad atom \mid (\{\texttt{and} \mid \texttt{union}\}\ body^*) \mid (\{\texttt{neg} \mid \texttt{inv}\}\ body) \mid (\texttt{project-to}\ (object^*)\ body)$$

$$atom \quad := \quad (object\ concept\_expr) \mid (object\ object\ role\_expr) \mid (object\ object\ (\texttt{constraint}\ \ chain\ chain\ constraint\_expr)) \mid (\texttt{same-as}\ \ variable\ individual)$$

$$chain \quad := \quad (role\_expr^*\ cd\_attribute)$$

Example constraint query atoms

- ```
  (?x ?y (constraint (has-mother age)
                     (has-father age) <))
  ```
- ```
  (?x ?y (constraint (has-brother age)
                     (age)
                     (= age-1 (+ 8 age-2))))
  ```

nRQL Language – Syntax (2)

- Syntax for `<body>`

$$body \;:=\; atom \mid (\,\{\texttt{and} \mid \texttt{union}\}\; body^*) \mid (\,\{\,\texttt{neg} \mid \texttt{inv}\}\; body\,) \mid$$
$$(\texttt{project-to}\;\; (object^*)\; body)$$

$$atom \;:=\; (object\; concept\_expr) \mid (object\; object\; role\_expr) \mid$$
$$(object\; object\; (\texttt{constraint}\;\; chain\; chain\; constraint\_expr)) \mid$$
$$(\texttt{same-as}\;\; variable\; individual)$$
$$chain \;:=\; (role\_expr^*\; cd\_attribute)$$

Example `same-as` query atoms

- `(same-as ?x betty)`

nRQL Language – Syntax (2)

- Syntax for `<body>`

$$body \;:=\; atom \mid (\,\{\texttt{and} \mid \texttt{union}\}\; body^*) \mid (\,\{\texttt{neg} \mid \texttt{inv}\}\; body\,) \mid$$
$$(\texttt{project-to}\; (object^*)\; body)$$

$$atom \;:=\; (object\ concept\_expr) \mid (object\ object\ role\_expr) \mid$$
$$(object\ object\ (\texttt{constraint}\ \ chain\ chain\ constraint\_expr)) \mid$$
$$(\texttt{same-as}\ \ variable\ individual)$$
$$chain \;:=\; (role\_expr^*\ cd\_attribute)$$

$concept\_expr$	$:=$	a Racer concept, with some extensions for OWL
$role\_expr$	$:=$	a Racer role $\mid$ $(\texttt{inv}\ \ role\_expr)$ $\mid$ $(\texttt{not}\ \ role\_expr)$
$constraint\_expr$	$:=$	a Racer concrete predicate
$cd\_attribute$	$:=$	a Racer concrete domain attribute
$datatype\_property$	$:=$	a Racer role used as OWL datatype property
$annotation\_property$	$:=$	a Racer role used as OWL annotation property

nRQL Variables

- Variables can only be bound to ABox individuals, not to concrete domain objects or even concrete domain values
- nRQL offers two kinds of variables: `?x`, `$?x`
  - `?x` prohibits binding to individuals which are already bound by other variables, e.g. `?y` (mapping must be injective)
  - "UNA" for variables

```
? (retrieve (?x ?y) (and (?x man) (?y man)))
> NIL
? (retrieve ($?x $?y) (and ($?x man)
                           ($?y man)))
> ((($?X CHARLES) ($?Y CHARLES)))
```

Complex nRQL Queries

- Compound nRQL queries are defined inductively
  - Every query atom `ai` is a body.
  - If `a1 ...an` are query bodies, then the following expressions are also bodies
    - `(neg ai)`
    - `(inv ai)`
    - `(and a1 ...an)`
    - `(union a1 ...an)`
    - `(project-to (objects-in-ai) ai)`
- Each variable creates a new axis in an n-dimensional tuple space
- A projection (specified by `<head>`) is made before that set is returned.

Illustration of Semantics

```
(retrieve (?x)
          (and (?x woman)
               (?y man)
               (?x ?y has-child)))
```

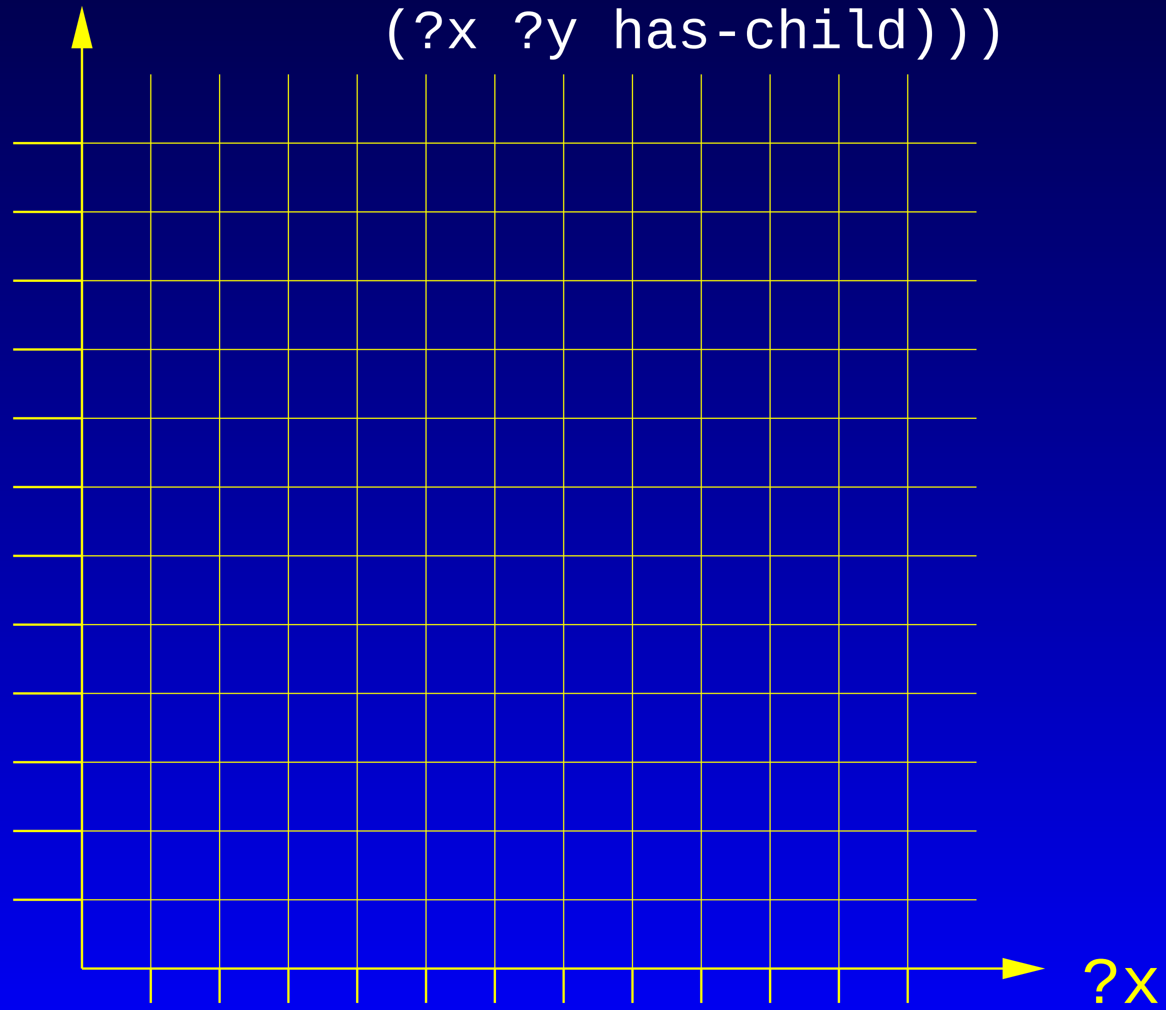


Illustration of Semantics

```
(retrieve (?x)
          (and (?x woman)
               (?y man)
               (?x ?y has-child)))
```

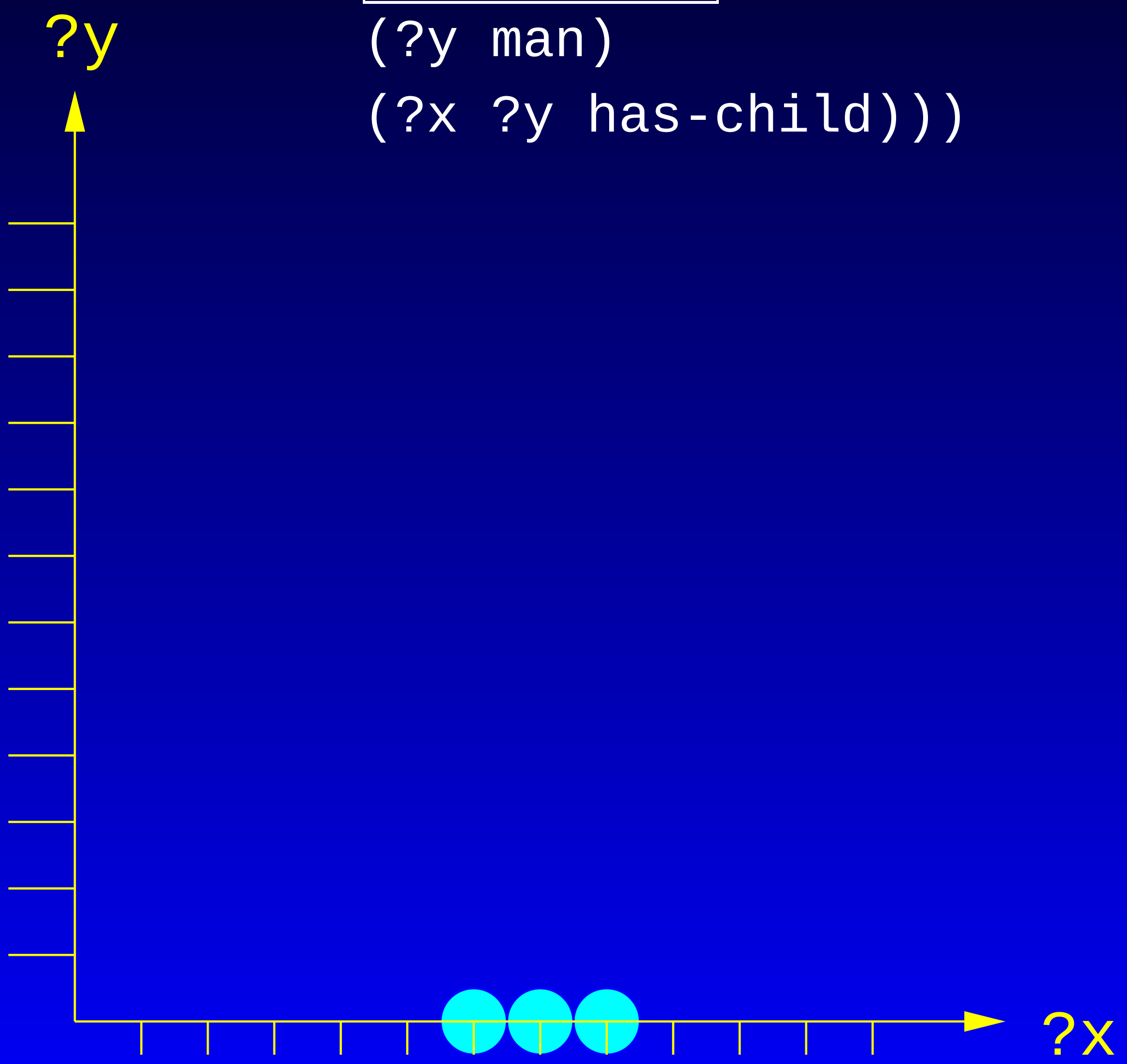


Illustration of Semantics

```
(retrieve (?x)
          (and  (and (?x woman) (?y top))
                (?y man)
                (?x ?y has-child)))
```

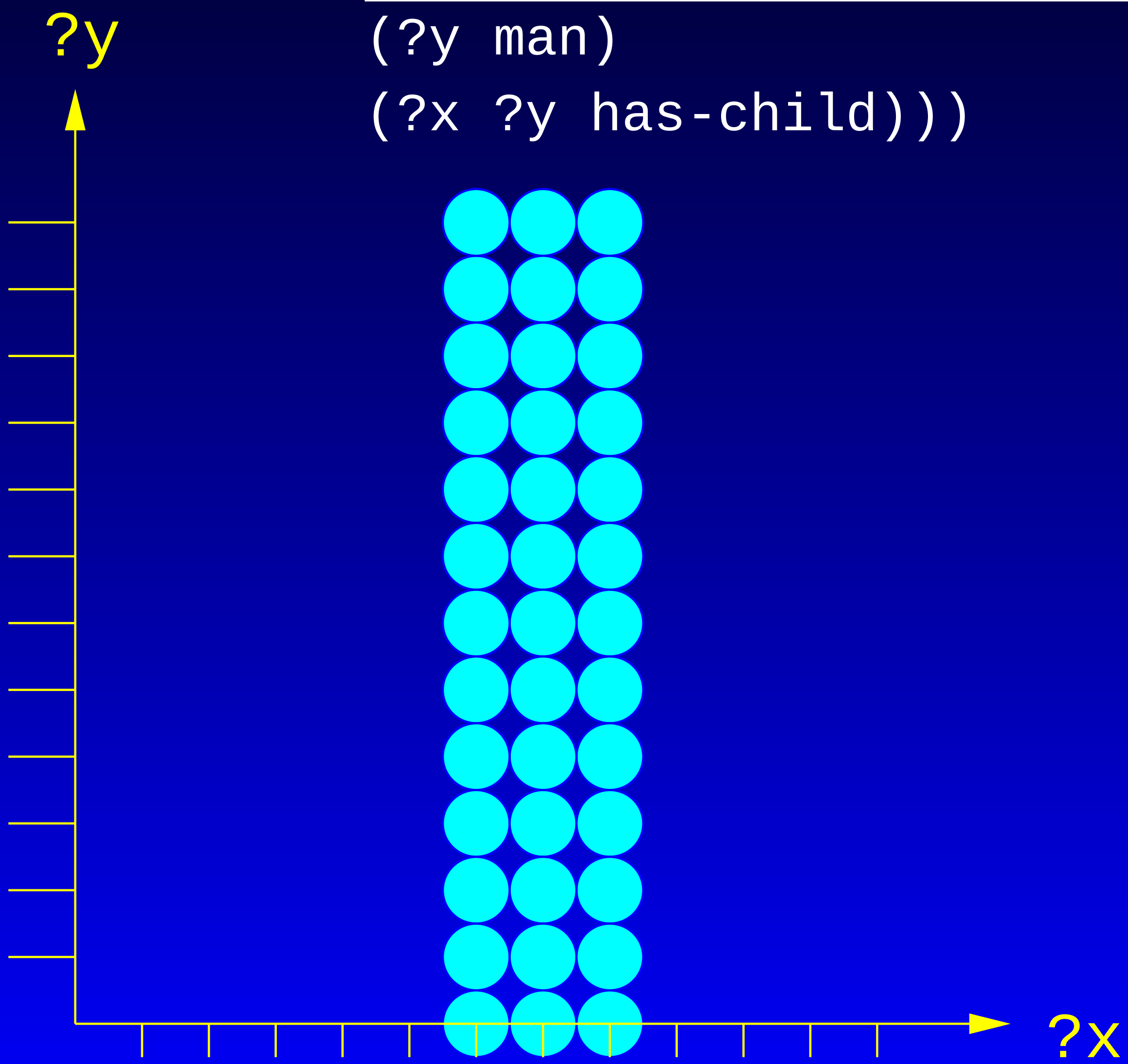


Illustration of Semantics

```
(retrieve (?x)
          (and (and (?x woman) (?y top))
               (?y man)
               (?x ?y has-child)))
```

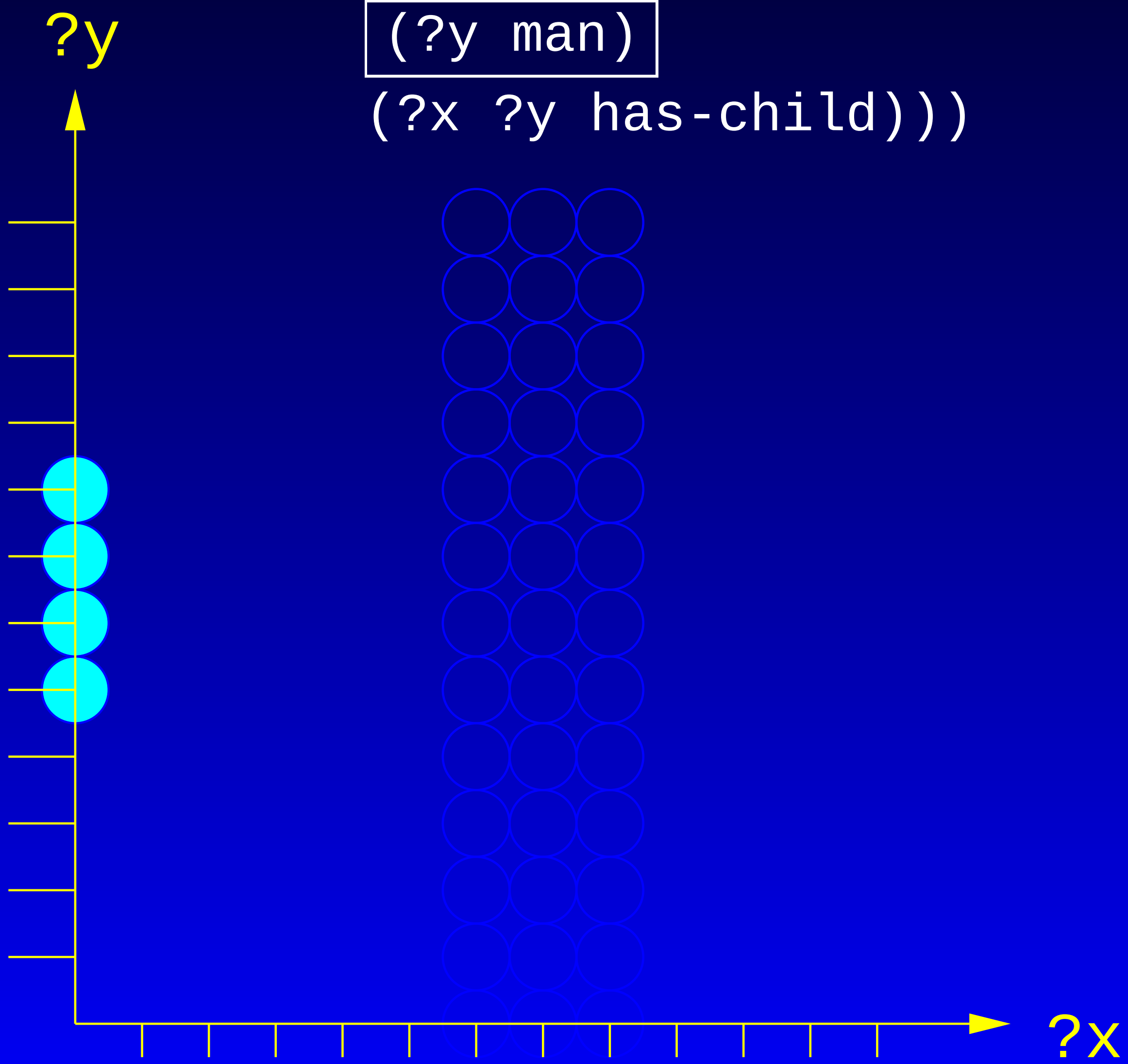


Illustration of Semantics

```
(retrieve (?x)
          (and (and (?x woman) (?y top))
               (and (?x top) (?y man))
               (?x ?y has-child)))
```

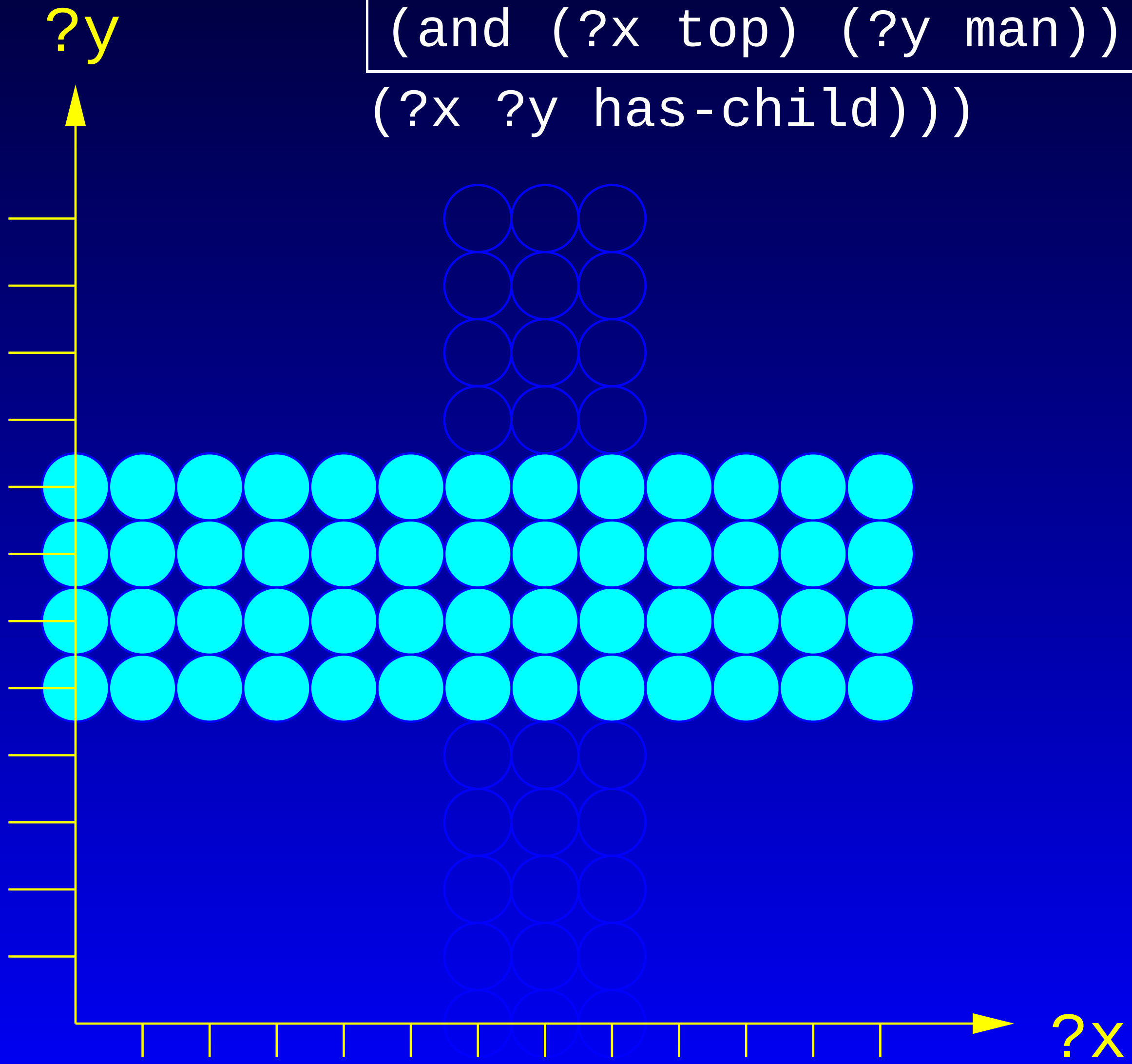


Illustration of Semantics

```
(retrieve (?x)
          ( and  (and (?x woman) (?y top))
                 (and (?x top) (?y man))
                 (?x ?y has-child)))
```

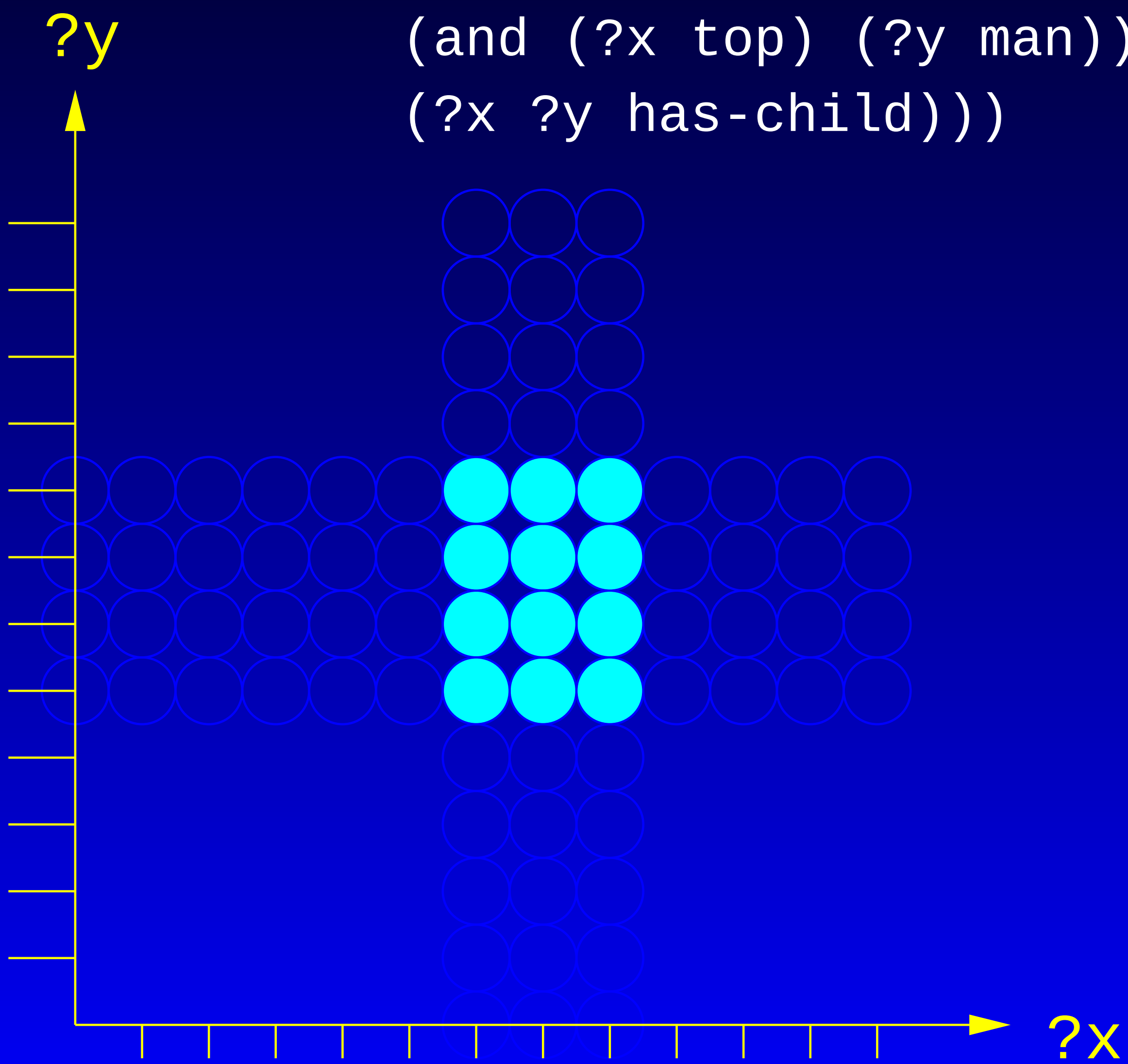


Illustration of Semantics

```
(retrieve (?x)
          (and (and (?x woman) (?y top))
               (and (?x top) (?y man))
               (?x ?y has-child)))
```

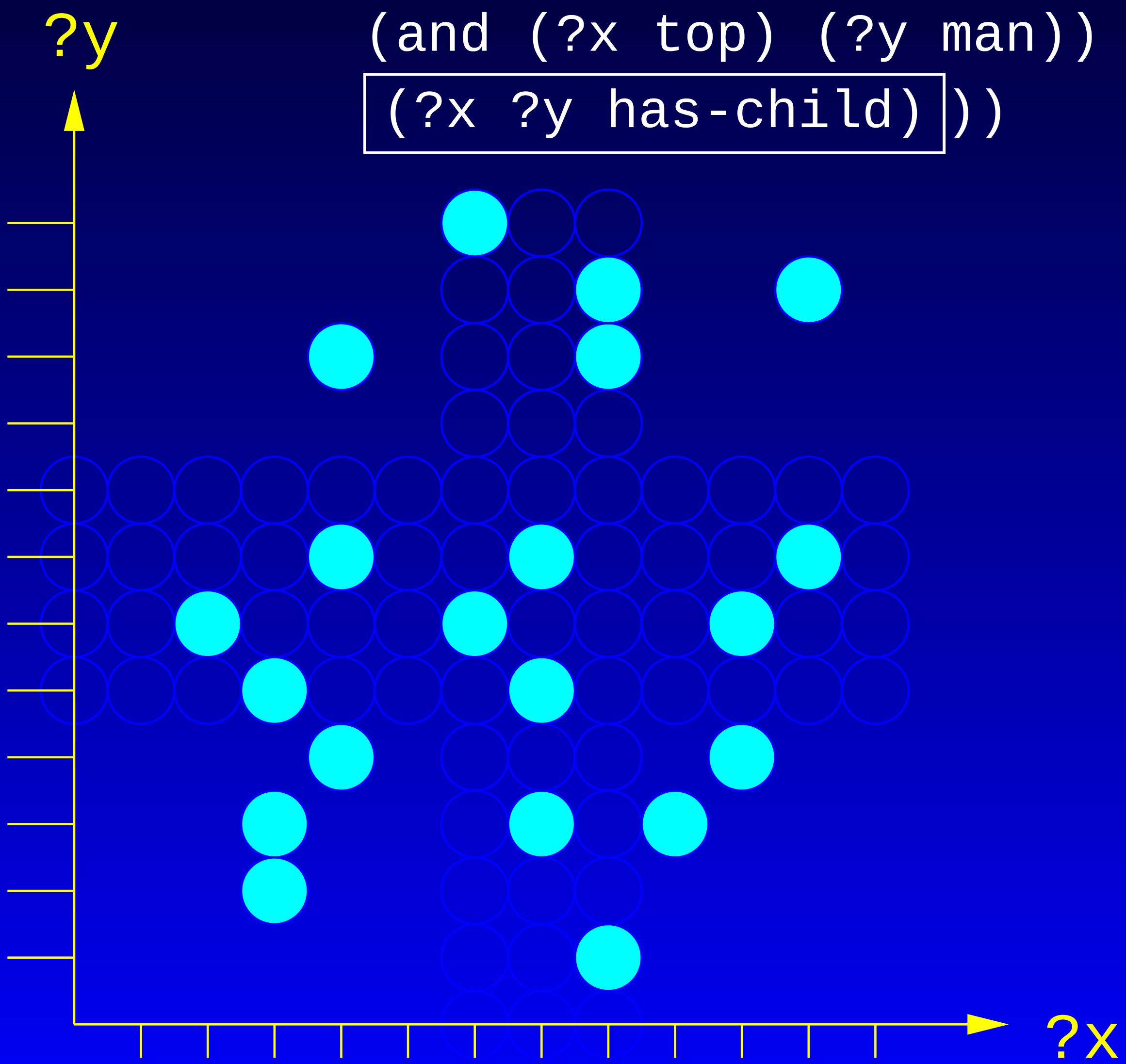


Illustration of Semantics

```
(retrieve (?x)
          ( and  (and (?x woman) (?y top))
                 (and (?x top) (?y man))
                 (?x ?y has-child)))
```

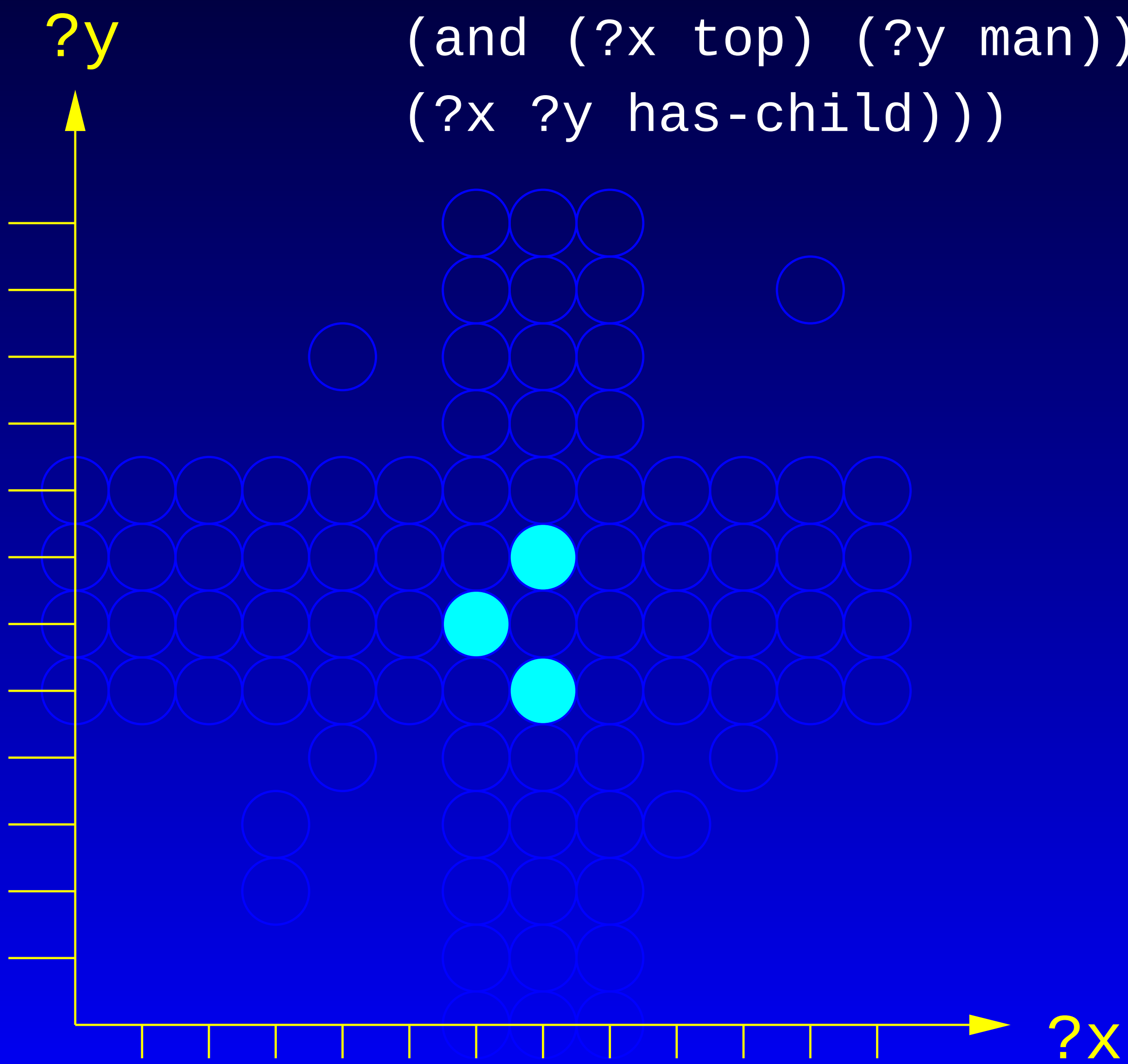
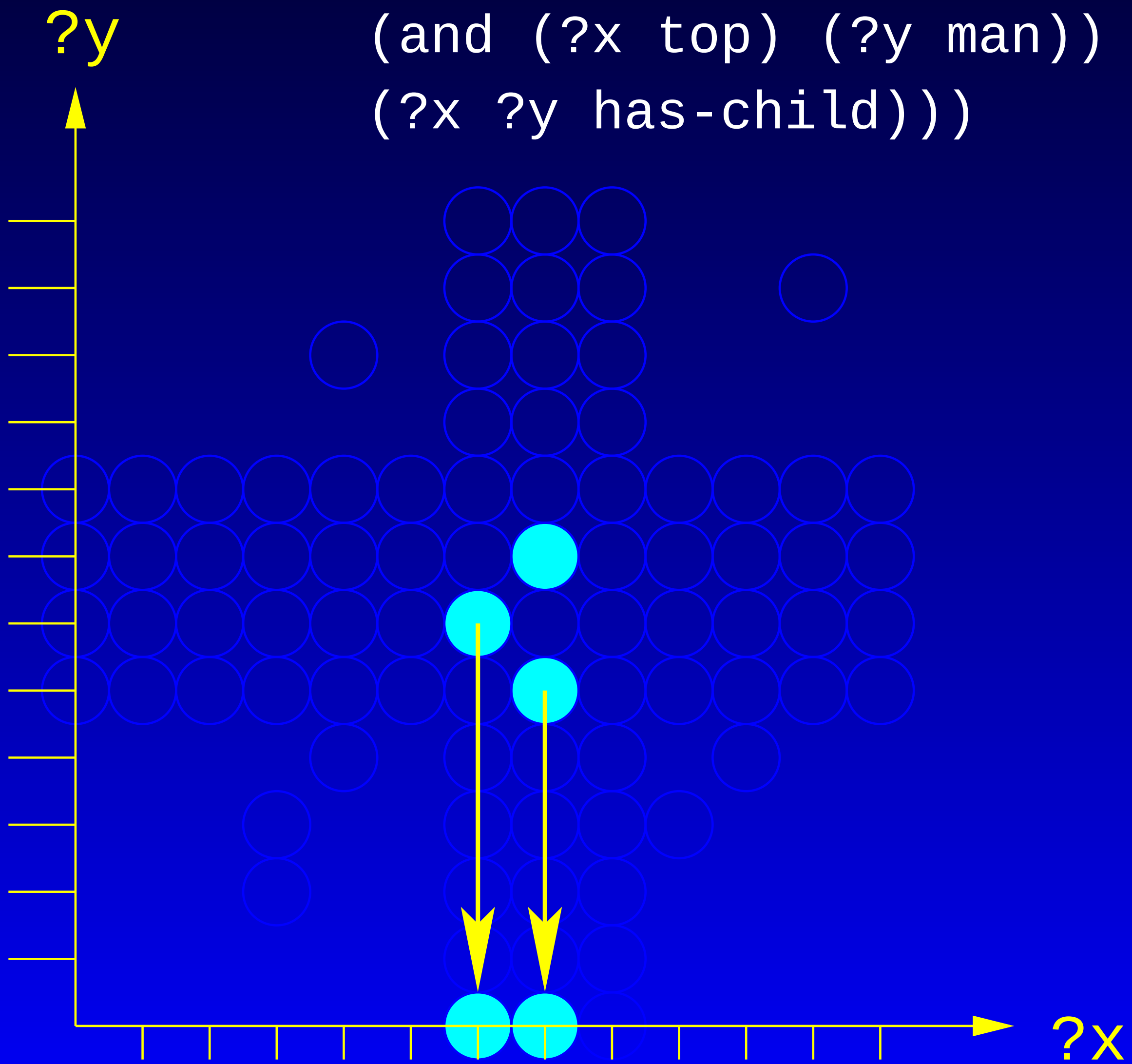


Illustration of Semantics

```
(retrieve (?x)
          (and (and (?x woman) (?y top))
               (and (?x top) (?y man))
               (?x ?y has-child)))
```



Negation in nRQL

- nRQL offers NAF with `neg`
- Semantics: simple set complement
  - well-defined for concept and role query atoms
  - well-defined for compound queries (DeMorgan etc.)
  - some "tricks" are needed for `same-as` and `constraint` query atoms
- Classical DL-like negation
  - obviously, in concept query atoms
  - but also in role query atoms `(?x ?y (not has-father))`

Illustration of `neg`

```
(retrieve (?x)
          (and (?x woman)
               (?y man)
               (?x ?y has-child)))
```

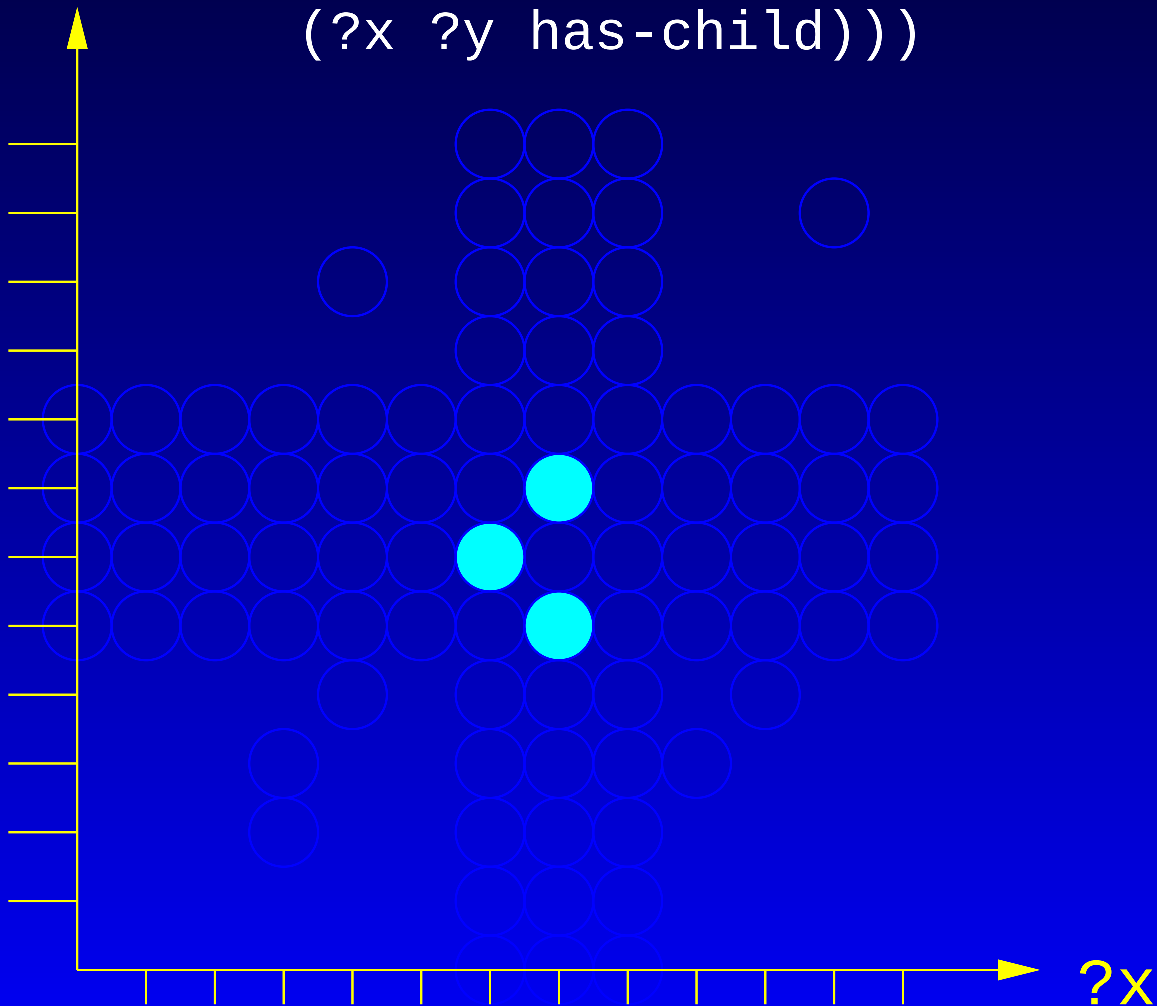


Illustration of neg

```
(retrieve (?x)
          (neg (and (?x woman)
                    (?y man)
                    (?x ?y has-child))))
```

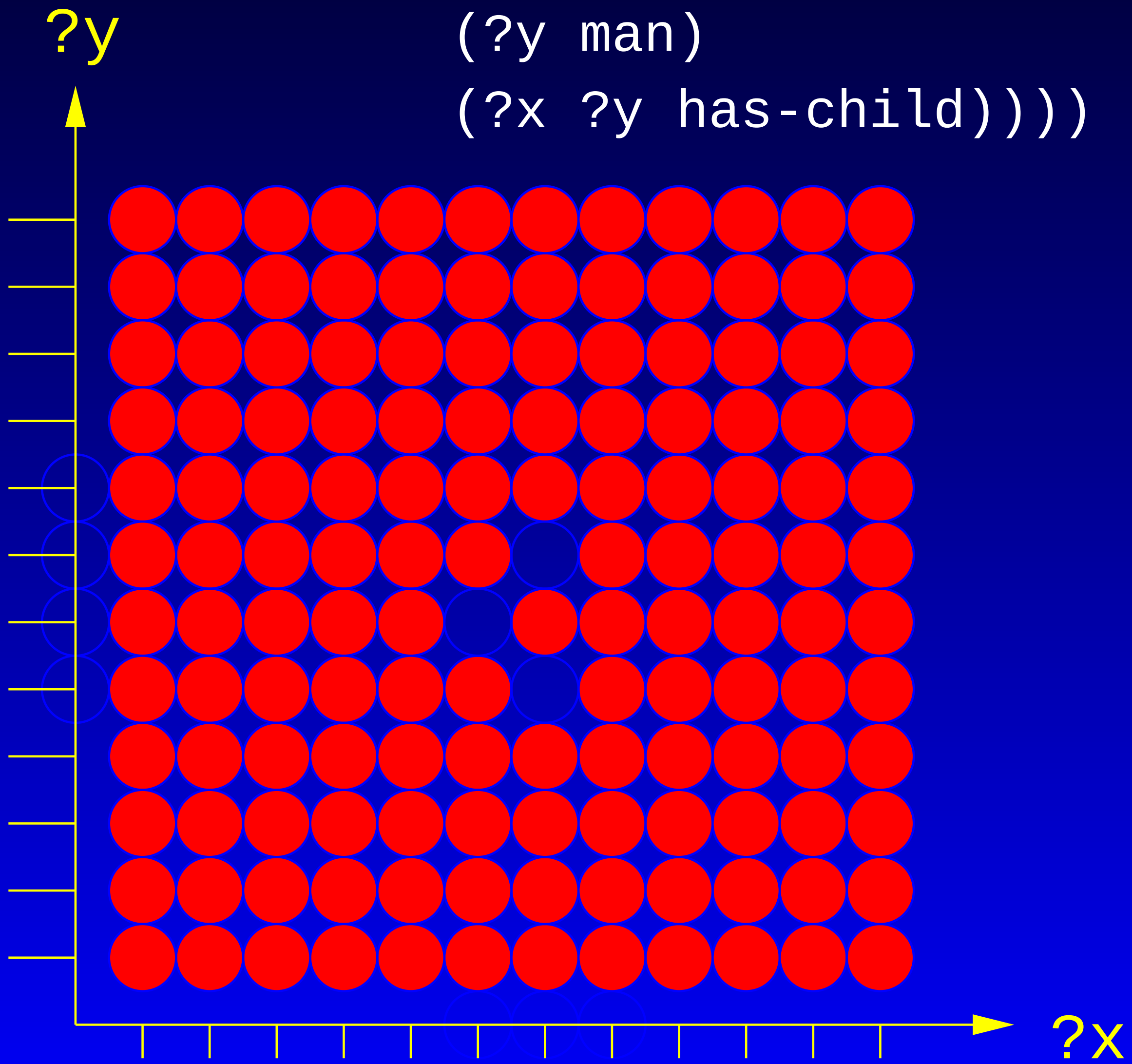


Illustration of neg

```
(retrieve (?x)
          (union (neg (?x woman))
                 (neg (?y man))
                 (neg (?x ?y has-child))))
```

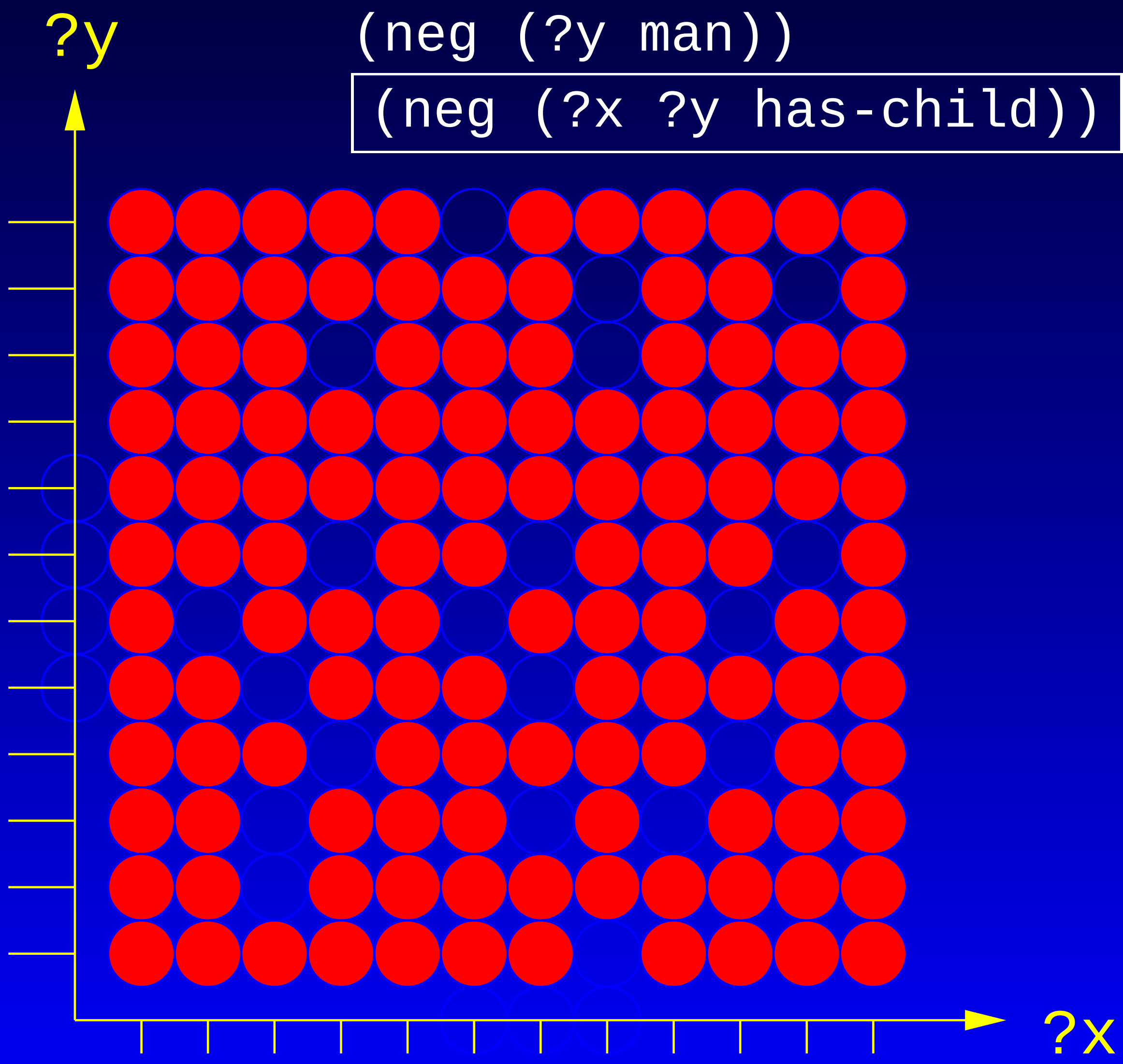


Illustration of `neg`

```
(retrieve (?x)
          (union (neg (?x woman))
                 (neg (?y man))
                 (neg (?x ?y has-child))))
```

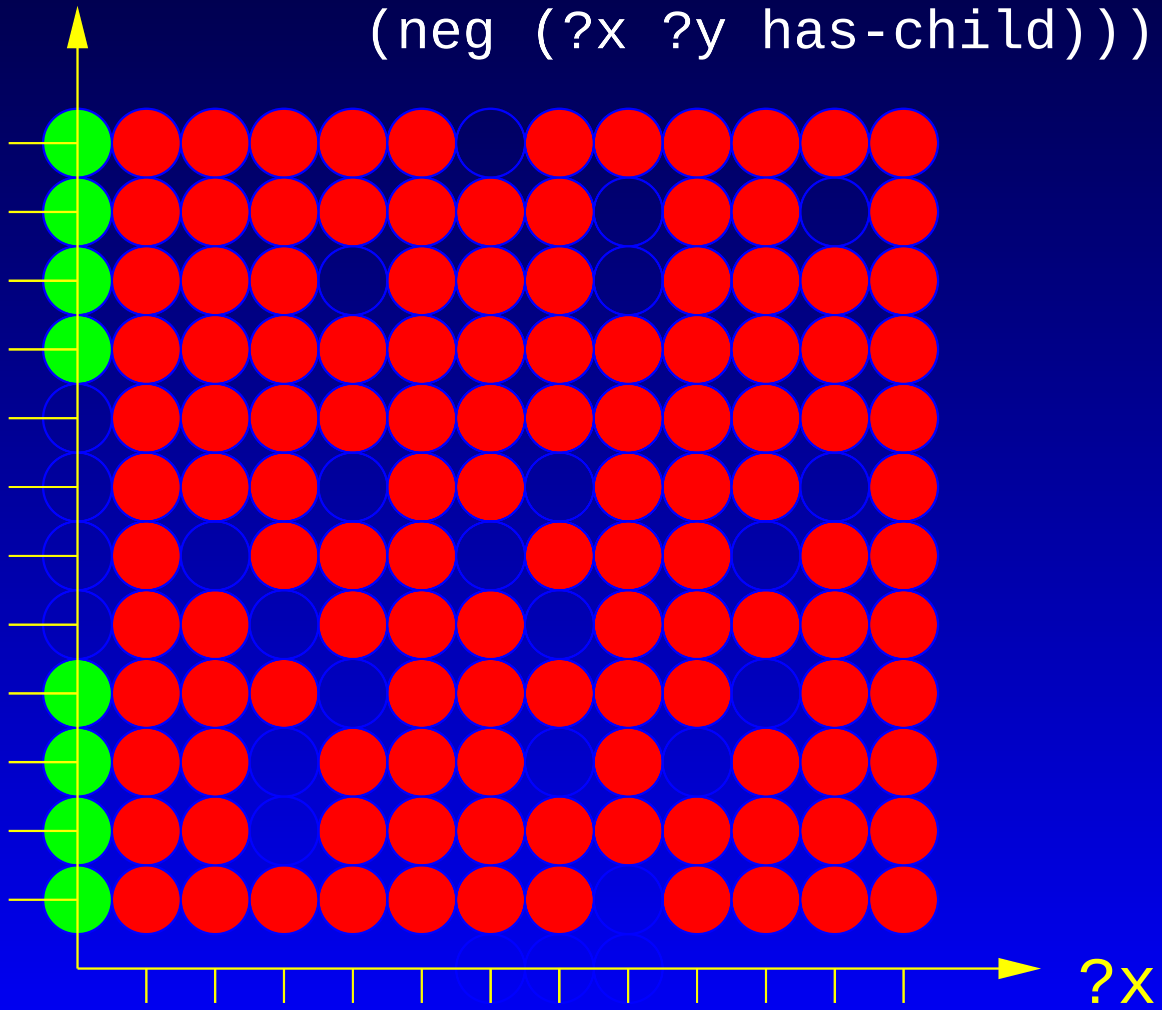
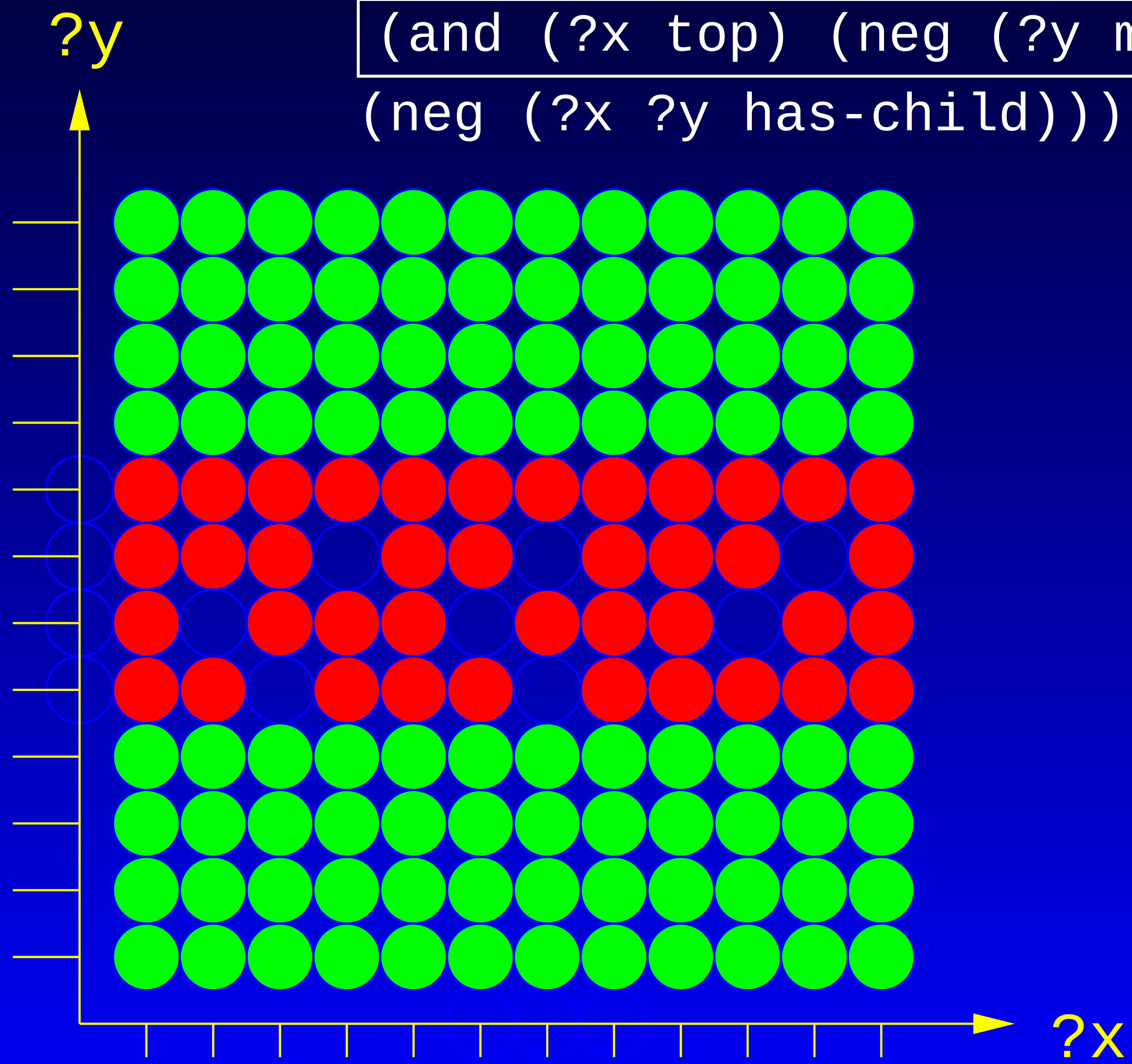


Illustration of neg

```
(retrieve (?x)
          (union (neg (?x woman))
                 (and (?x top) (neg (?y man)))
                 (neg (?x ?y has-child))))
```



TUHH Technische Universität Hamburg-Harburg

Illustration of `neg`

```
(retrieve (?x)
          (union (?x woman)
                 (and (?x top) (neg (?y man)))
                 (neg (?x ?y has-child))))
```

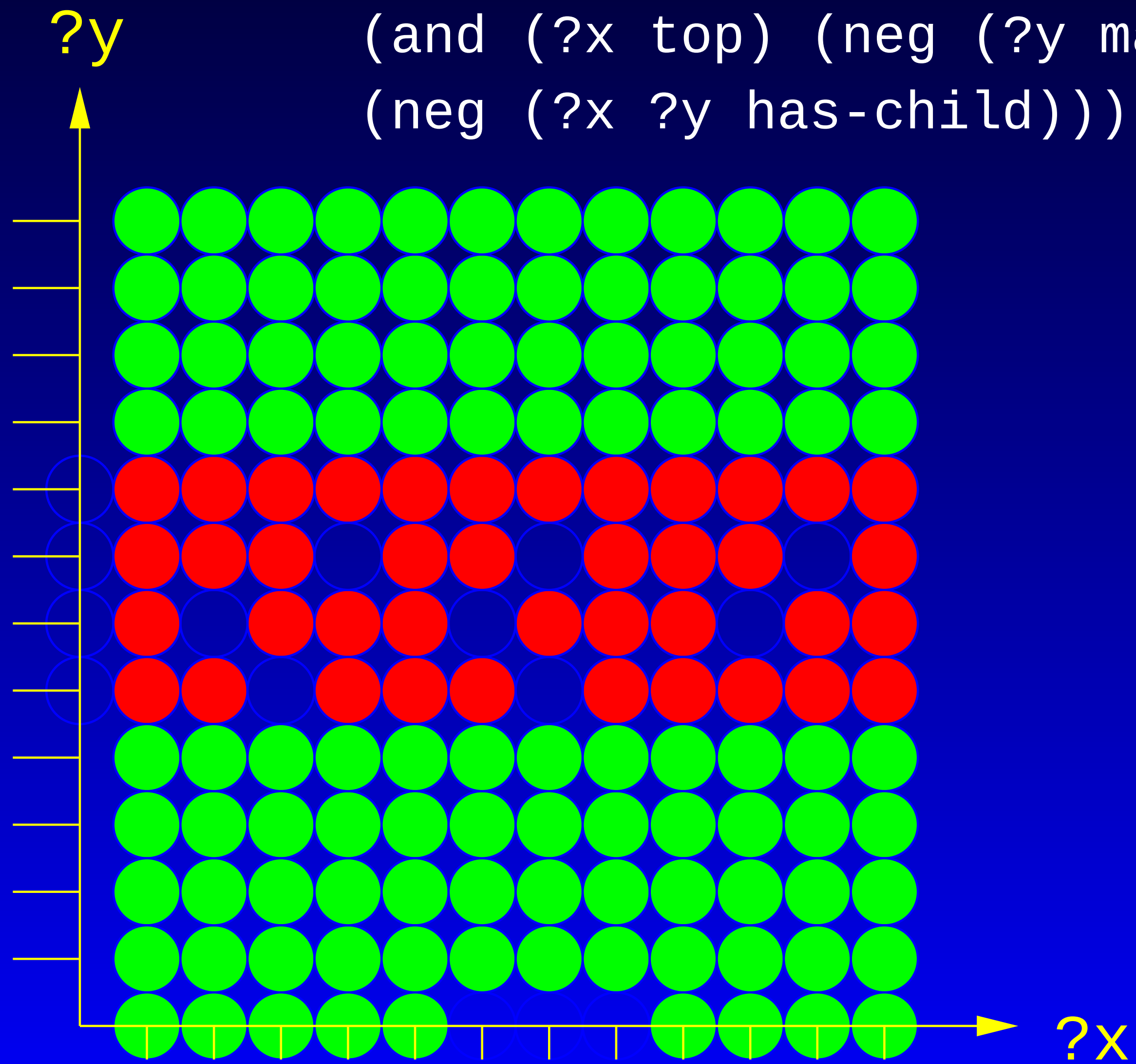


Illustration of neg

```
(retrieve (?x)
          (union (and (neg (?x woman) (?y top)))
                 (and (?x top) (neg (?y man)))
                 (neg (?x ?y has-child))))
```

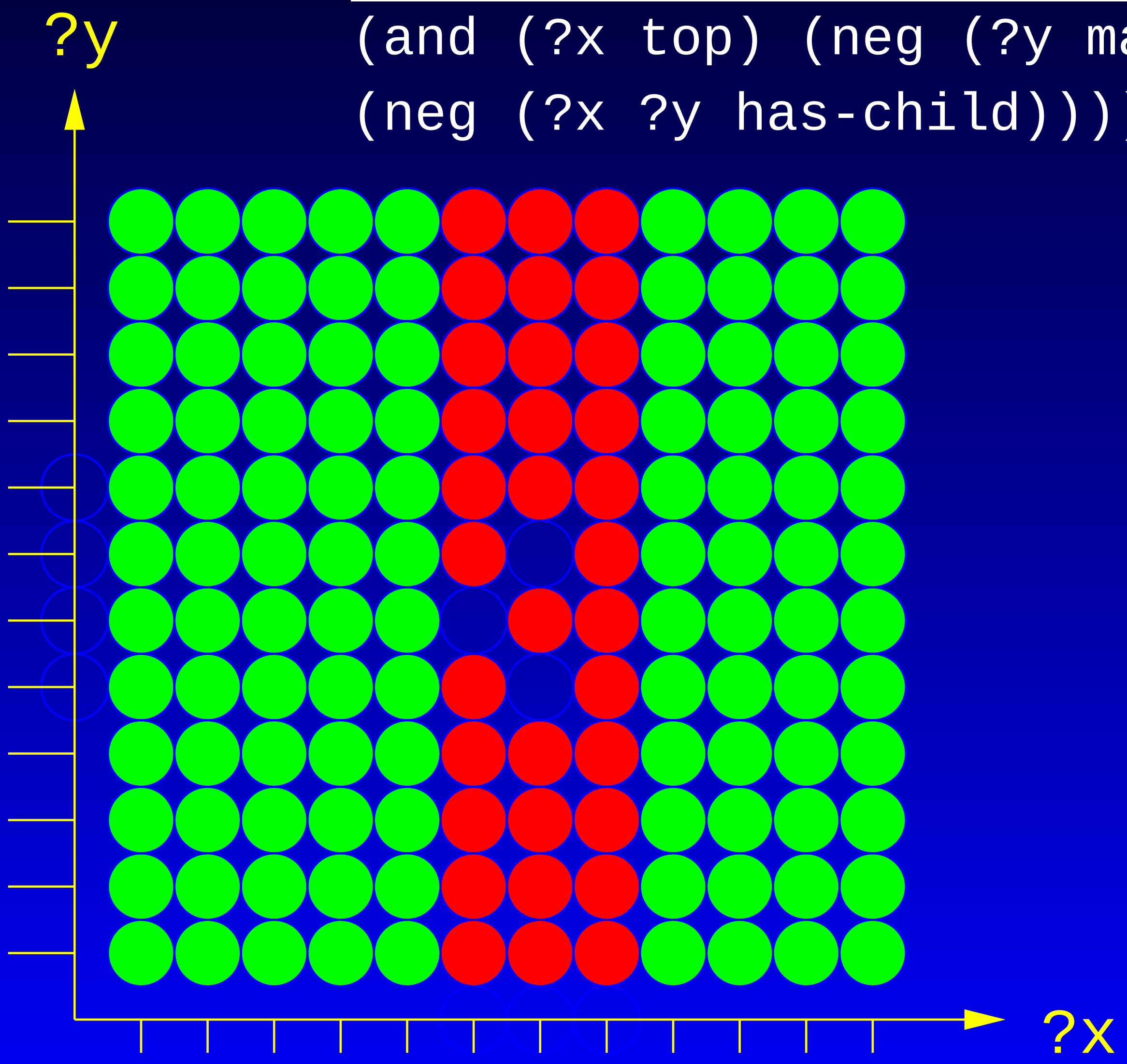


Illustration of neg

```
(retrieve (?x)
          (union (and (neg (?x woman) (?y top)))
                 (and (?x top) (neg (?y man)))
                 (neg (?x ?y has-child))))
```

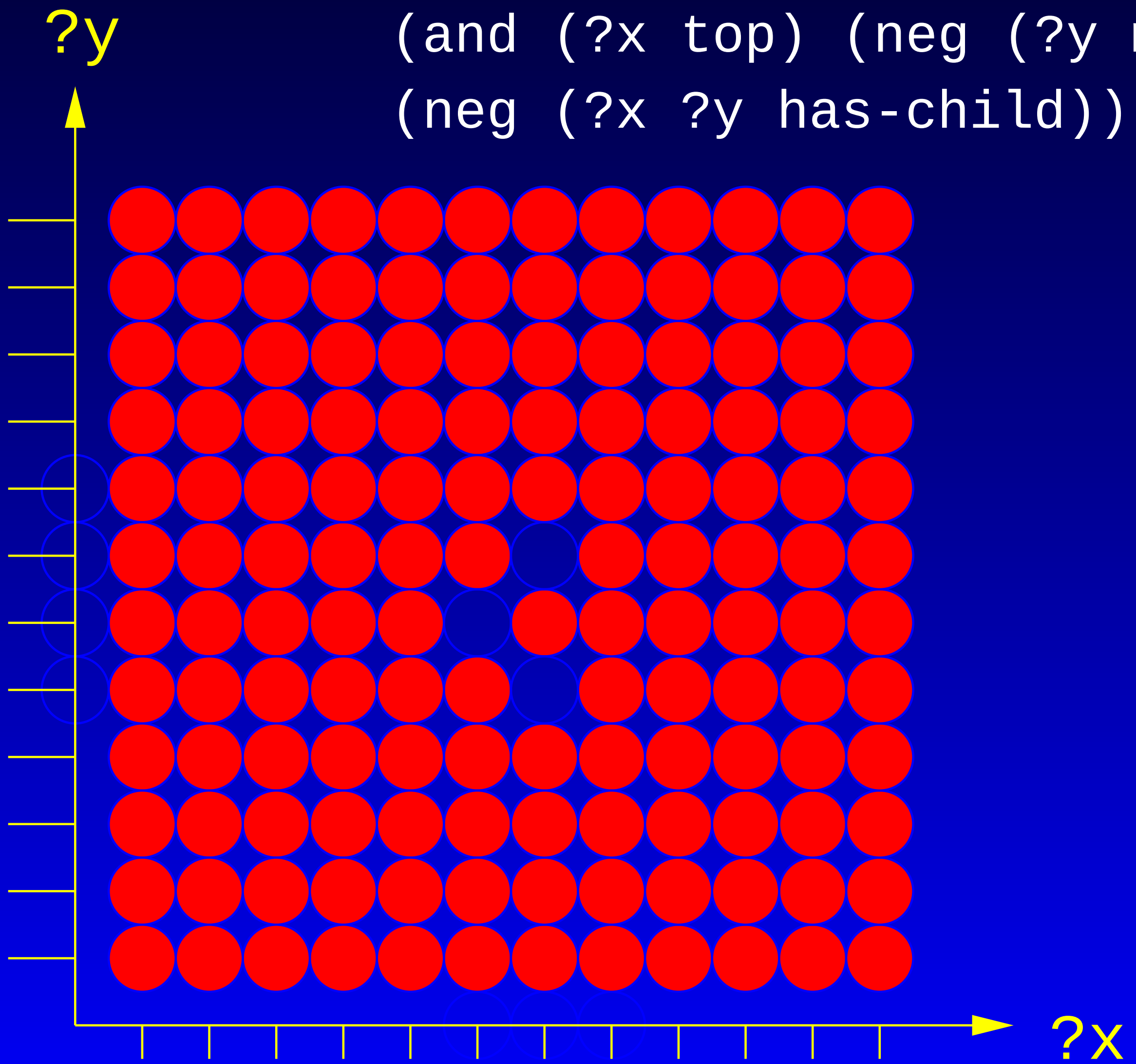


Illustration of neg

```
(retrieve (?x)
          (and (?x woman)
               (?y man)
               (?x ?y has-child)))
```

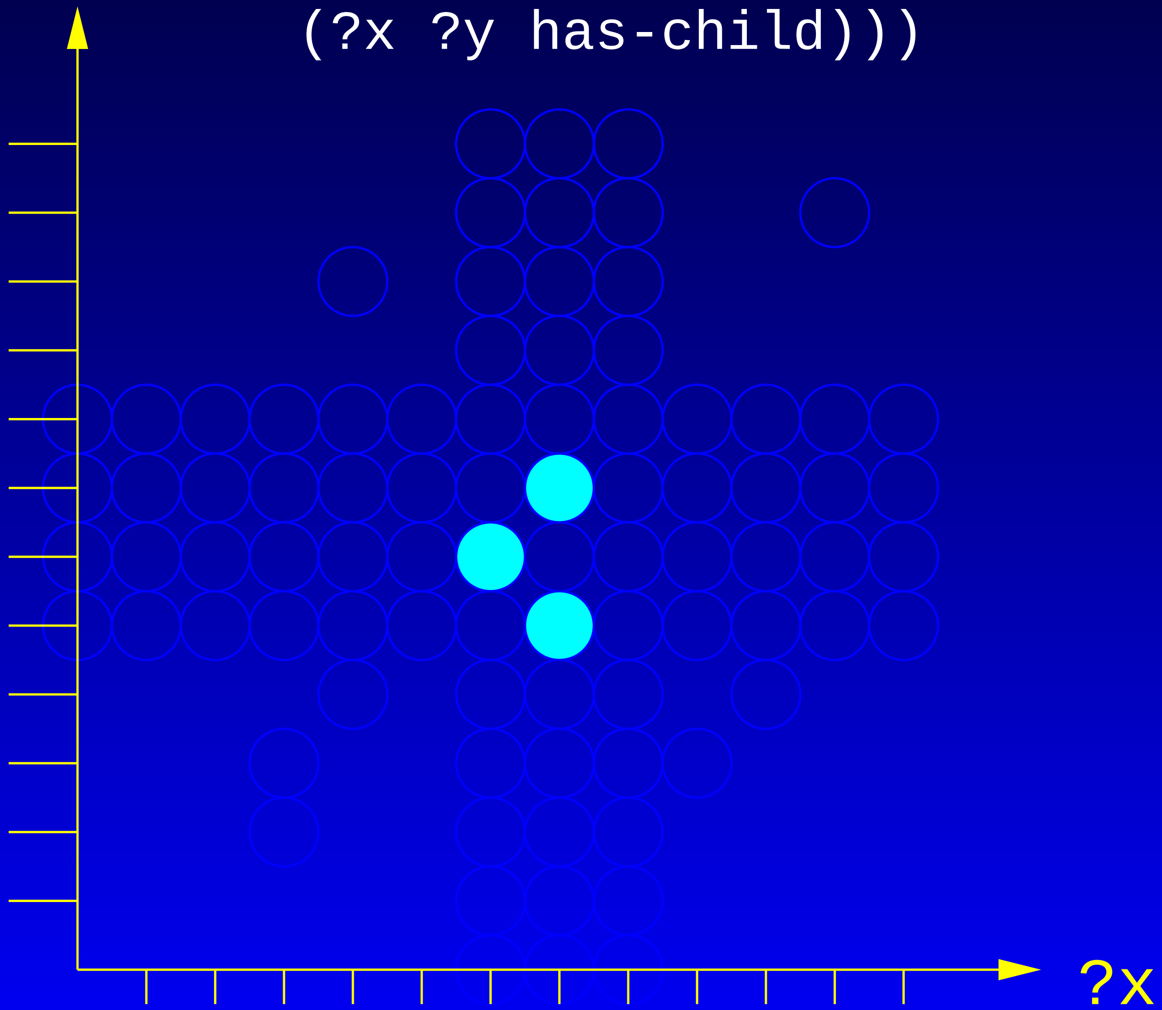


Illustration of neg

```
(retrieve (?x)
          (neg (and (?x woman)
                    (?y man)
                    (?x ?y has-child))))
```

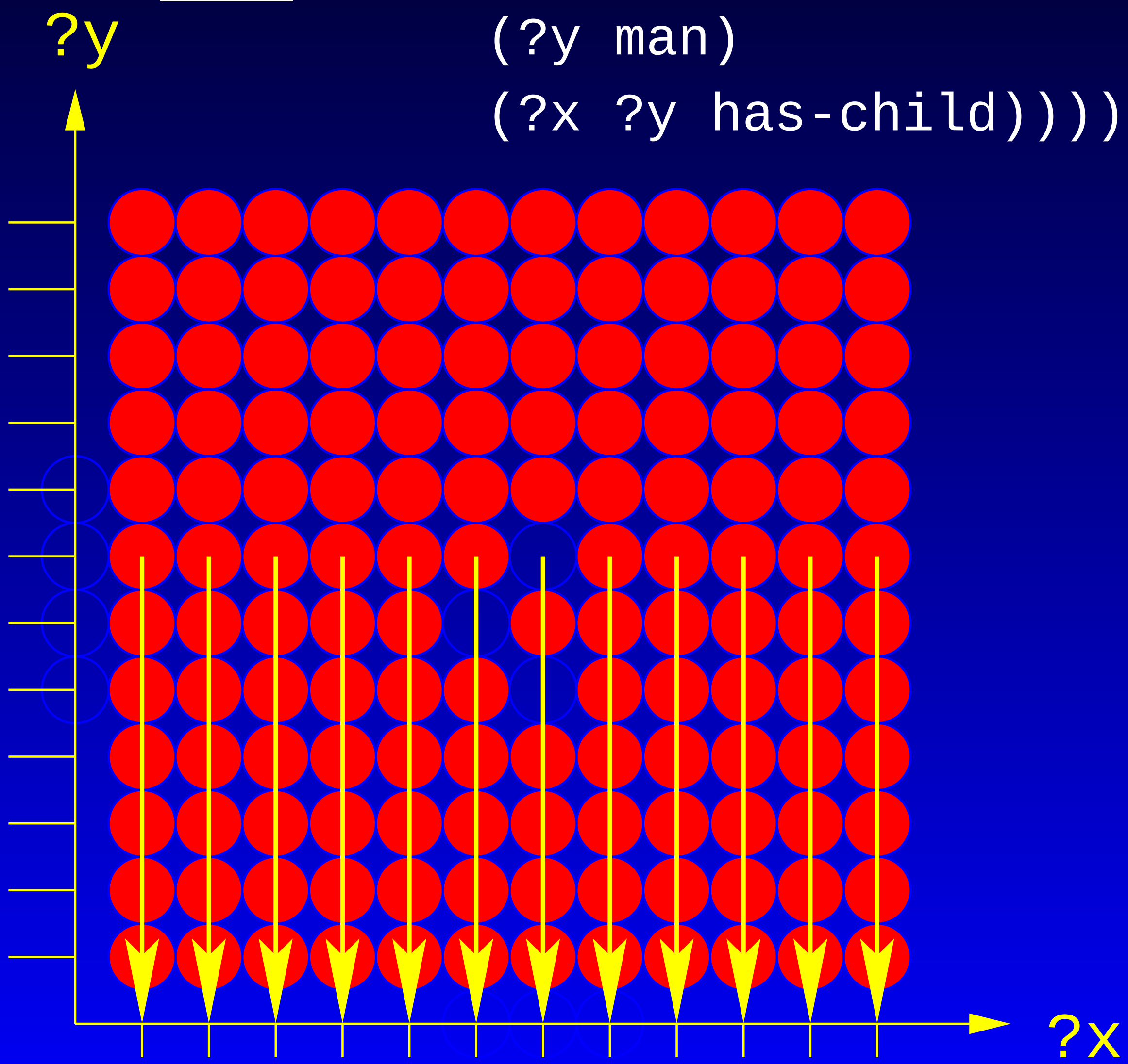


Illustration of neg

```
(retrieve (?x)
          (neg (and (?x woman)
                    (?y man)
                    (?x ?y has-child))))
```

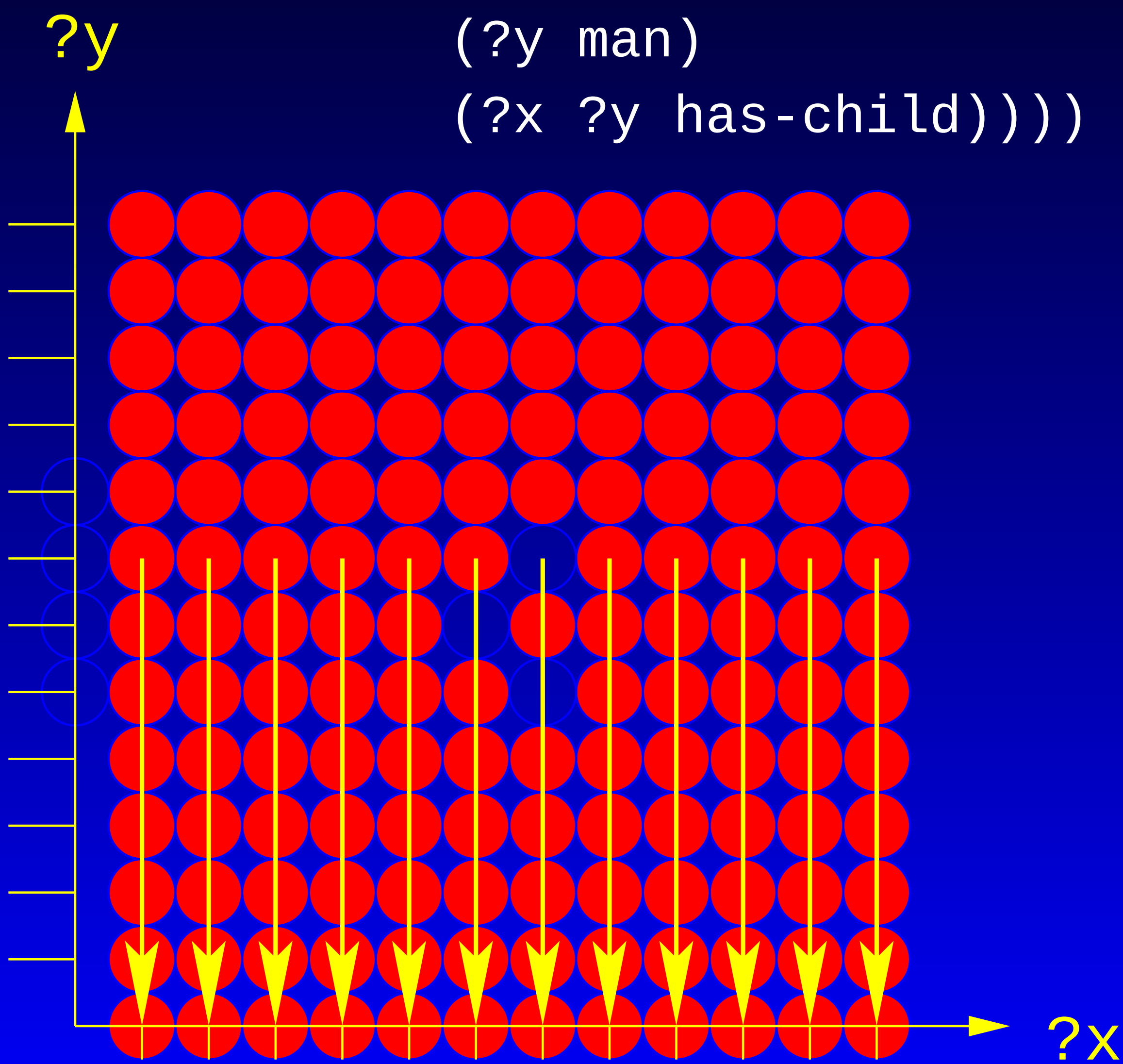


Illustration of neg

```
(retrieve (?x)
          (and (?x woman)
               (?y man)
               (?x ?y has-child)))
```

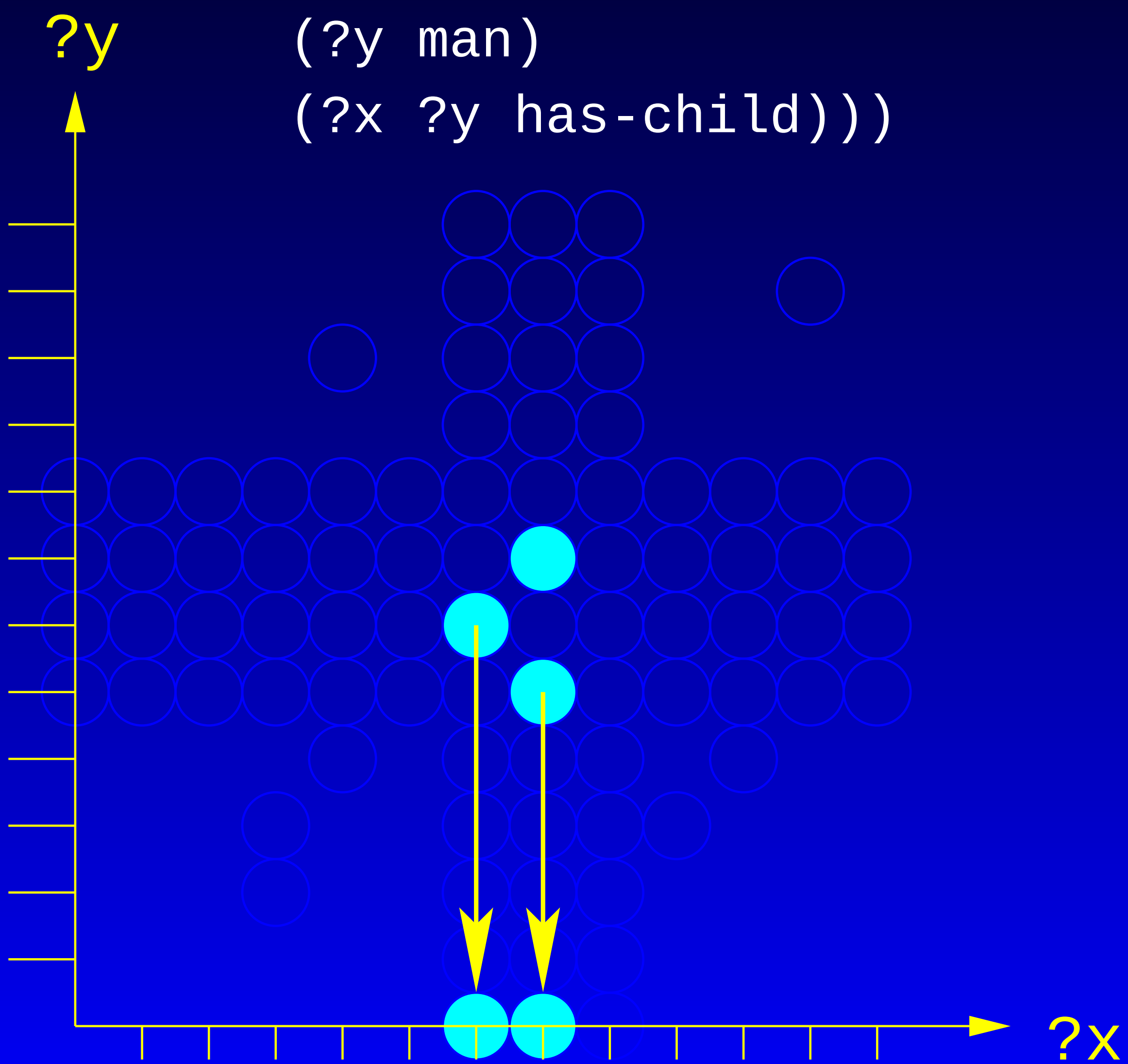


Illustration of neg

```
( NEG (retrieve (?x)
                (and (?x woman)
                     (?y man)
                     (?x ?y has-child))))
```

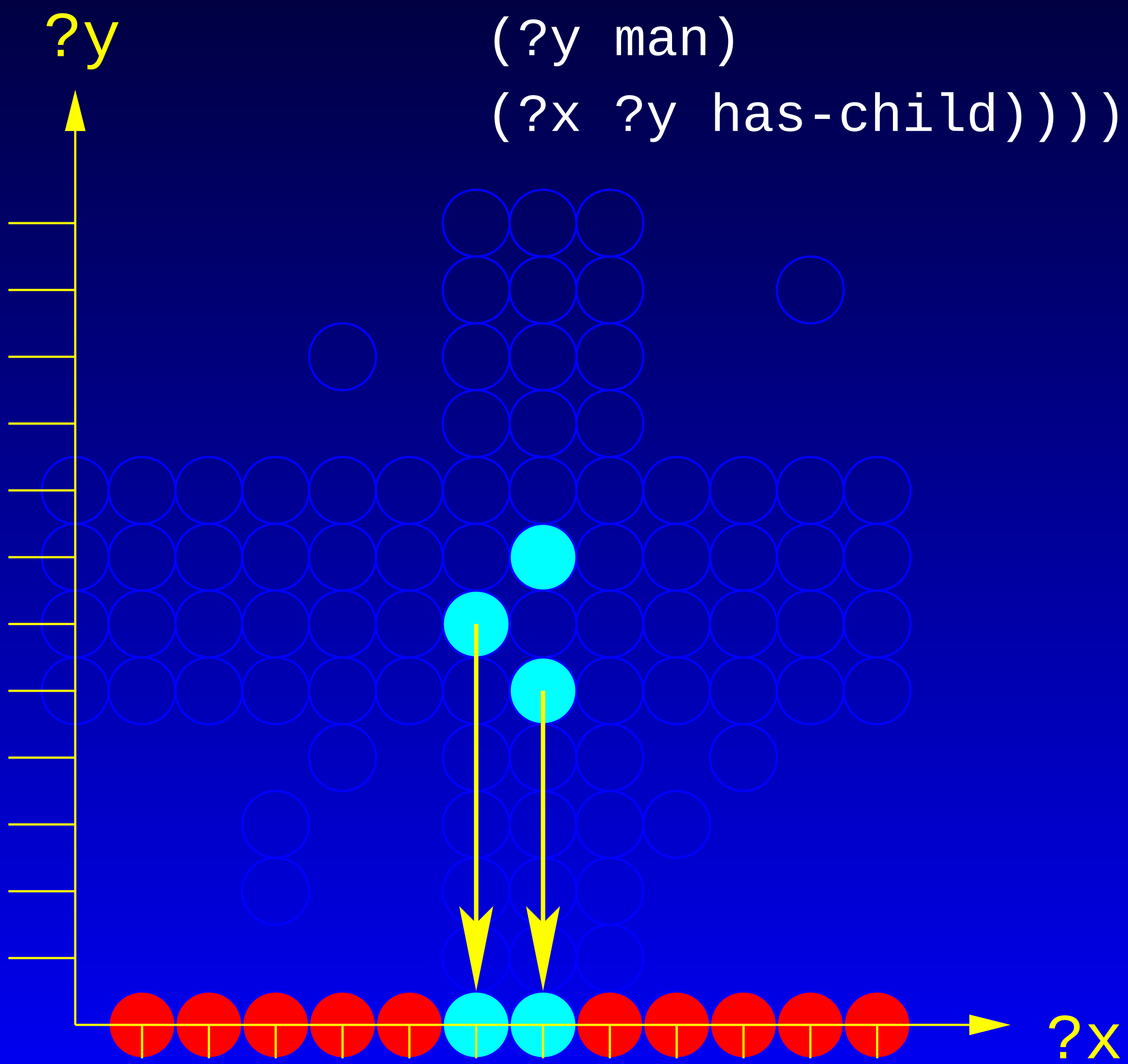
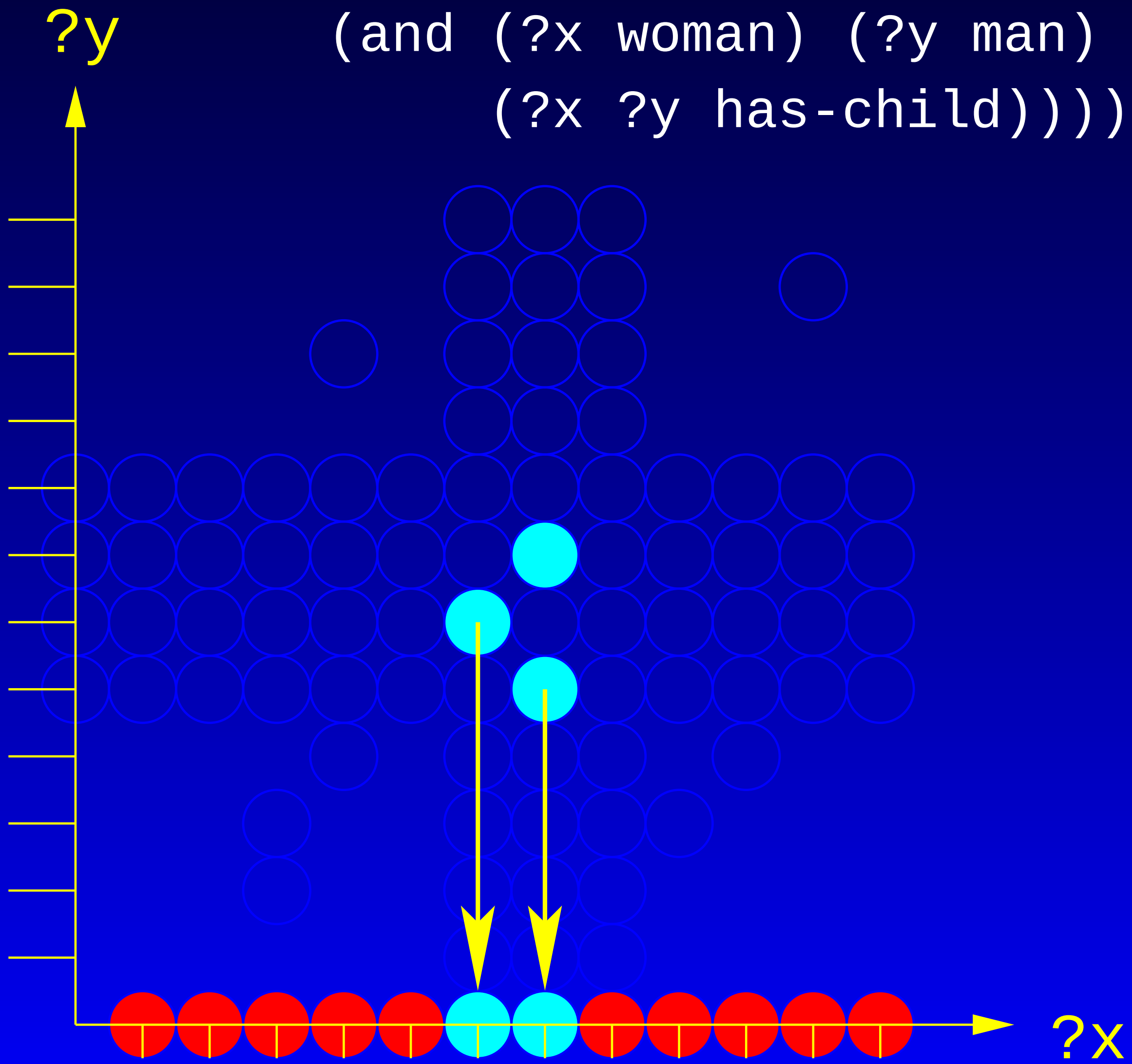


Illustration of neg

```
(retrieve (?x)
          (neg (project-to (?x)
                   (and (?x woman) (?y man)
                        (?x ?y has-child))))
```



Queries with Individuals

```
? (retrieve (betty)
            (betty woman))
```

```
> ((($?BETTY BETTY)))
```

- Explanation: query is rewritten into

```
(AND (SAME-AS $?BETTY BETTY)
     ($?BETTY WOMAN))
```

- `$?BETTY` is a variable that does not obey the unique name assumption for variables
- `(SAME-AS $?BETTY BETTY)` enforces binding of `$?BETTY` to `BETTY`

Semantic Consequences

- “NAF” for atoms with individuals can be tricky

```
(retrieve (betty)
          (neg (betty woman)))
=
(retrieve ($?betty)
    (neg (and ($?betty woman)
              (same-as $?betty betty))))
=
(retrieve ($?betty)
    (UNION (neg ($?betty woman))
           (neg (same-as $?betty betty))))
```

- one must define the semantics in such a way if the orthogonality of the language shall be preserved!

The Projection Operator

- We can retrieve all woman having children with

```
(retrieve (?x)
   (and (?x woman) (?x ?y has-child)))
```

- How can we retrieve woman which have no (known) children?

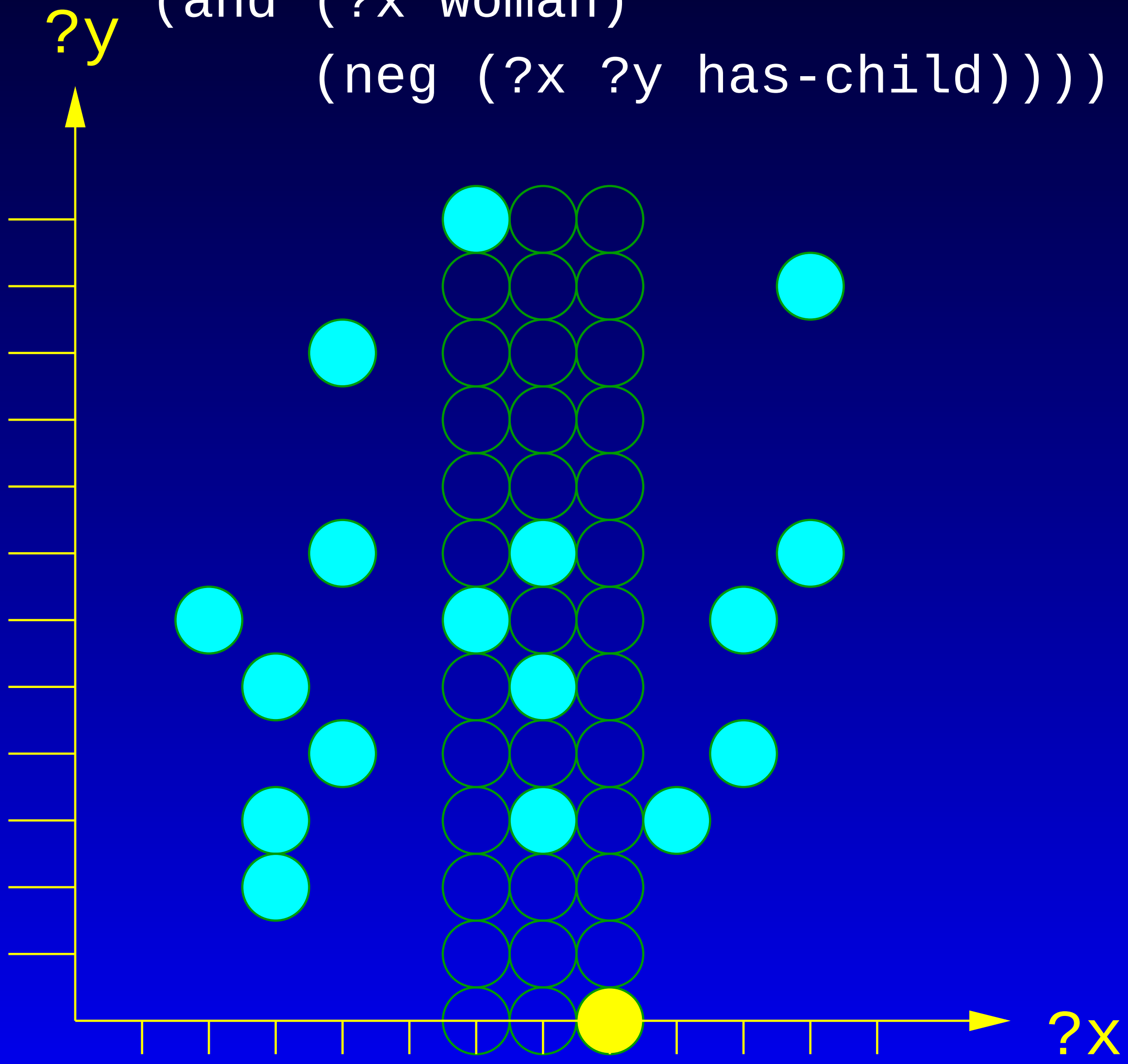
```
? (retrieve (?x)
     (?x (and woman (all has-child bottom))))

? (retrieve (?x)
     (and (?x woman)
          (neg (?x ?y has-child))))

? (retrieve (?x)
     (neg (and (?x woman) (?x ?y has-child))))
```

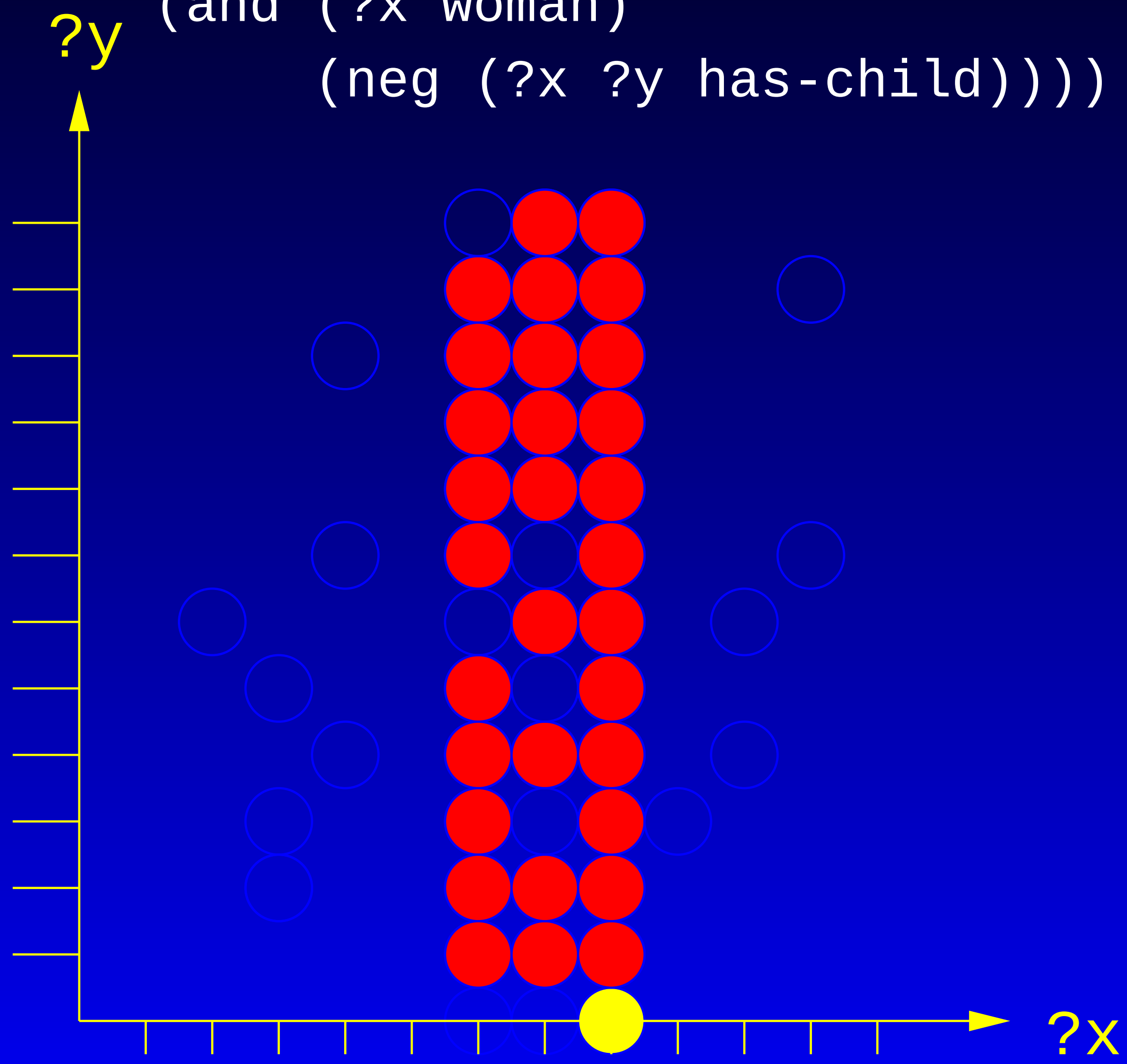
The Projection Operator (2)

```
Q1: (retrieve (?x)
              (and (?x woman)
                   (neg (?x ?y has-child))))
```



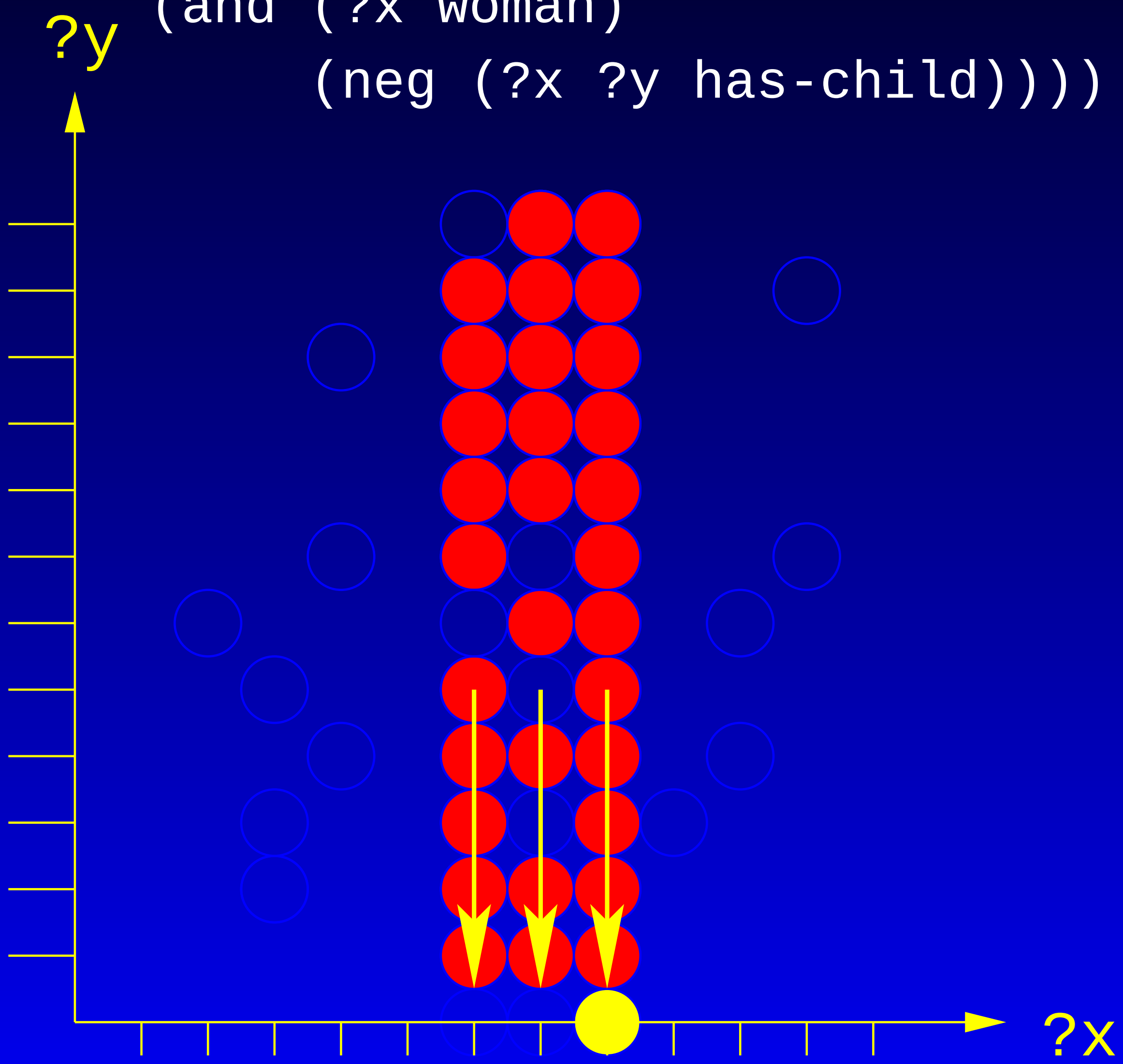
The Projection Operator (2)

```
Q1: (retrieve (?x)
              (and (?x woman)
                   (neg (?x ?y has-child))))
```



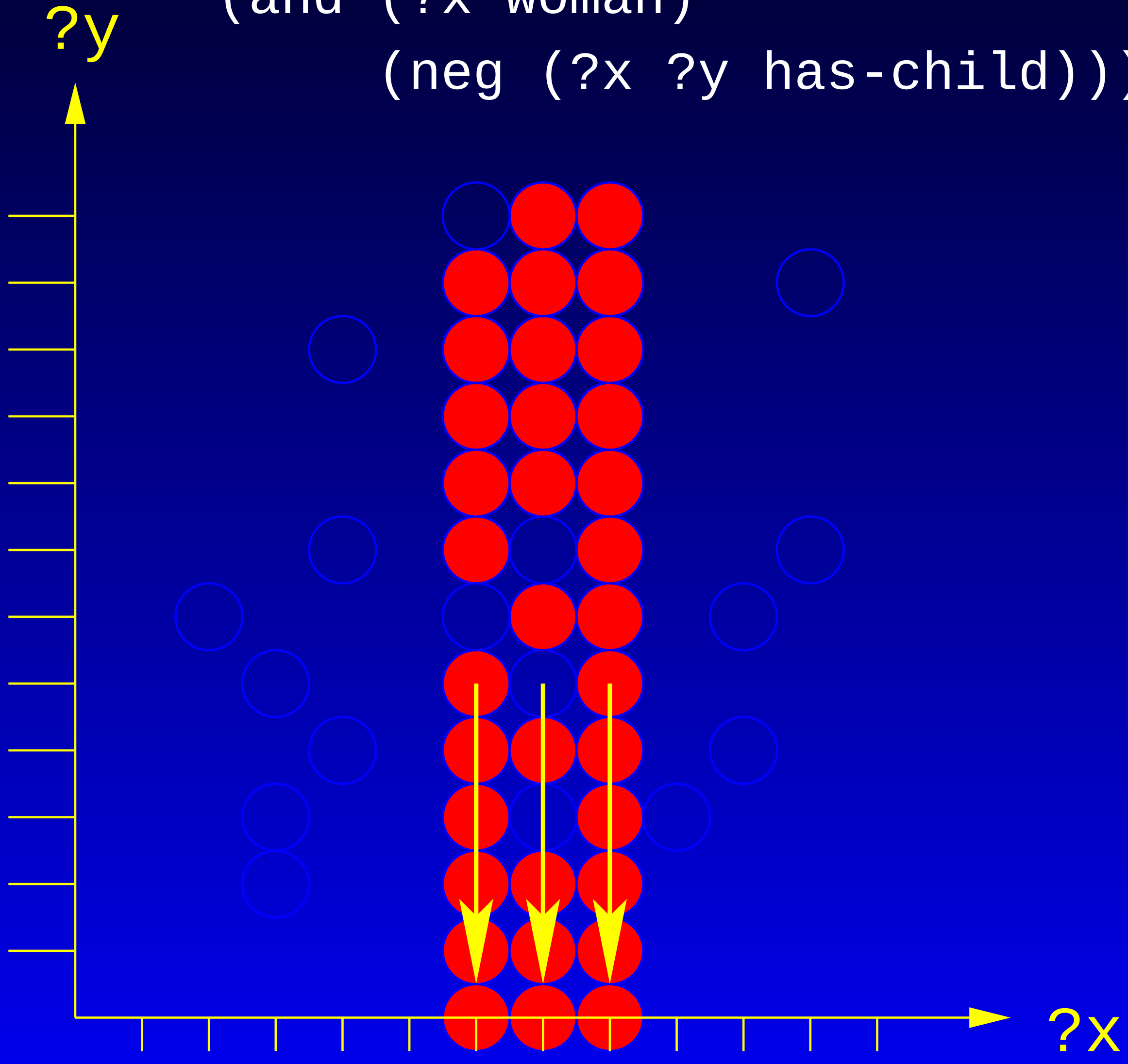
The Projection Operator (2)

```
Q1: (retrieve (?x)
              (and (?x woman)
                   (neg (?x ?y has-child))))
```



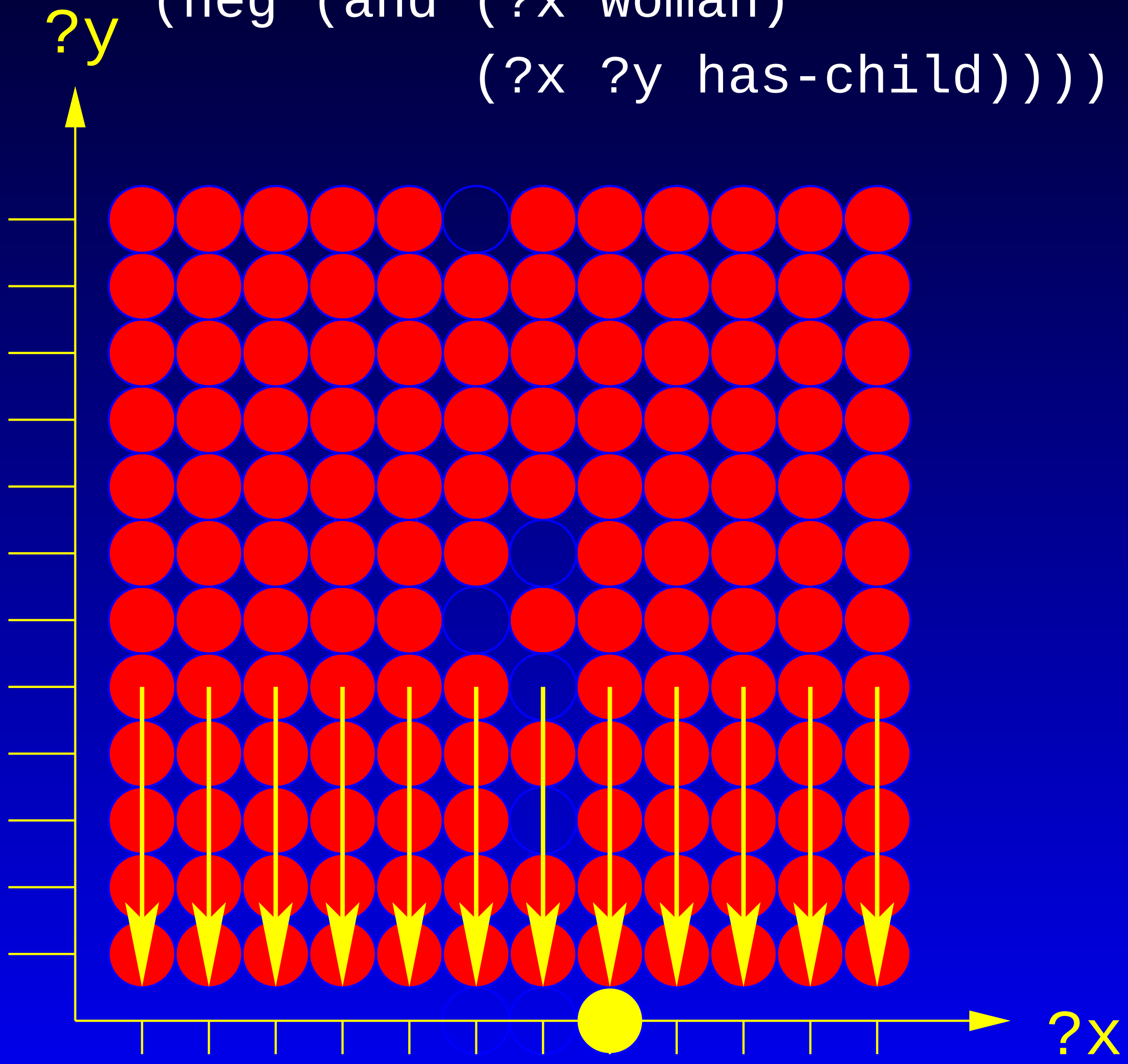
The Projection Operator (2)

```
Q1: (retrieve (?x)
        (and (?x woman)
             (neg (?x ?y has-child))))
```



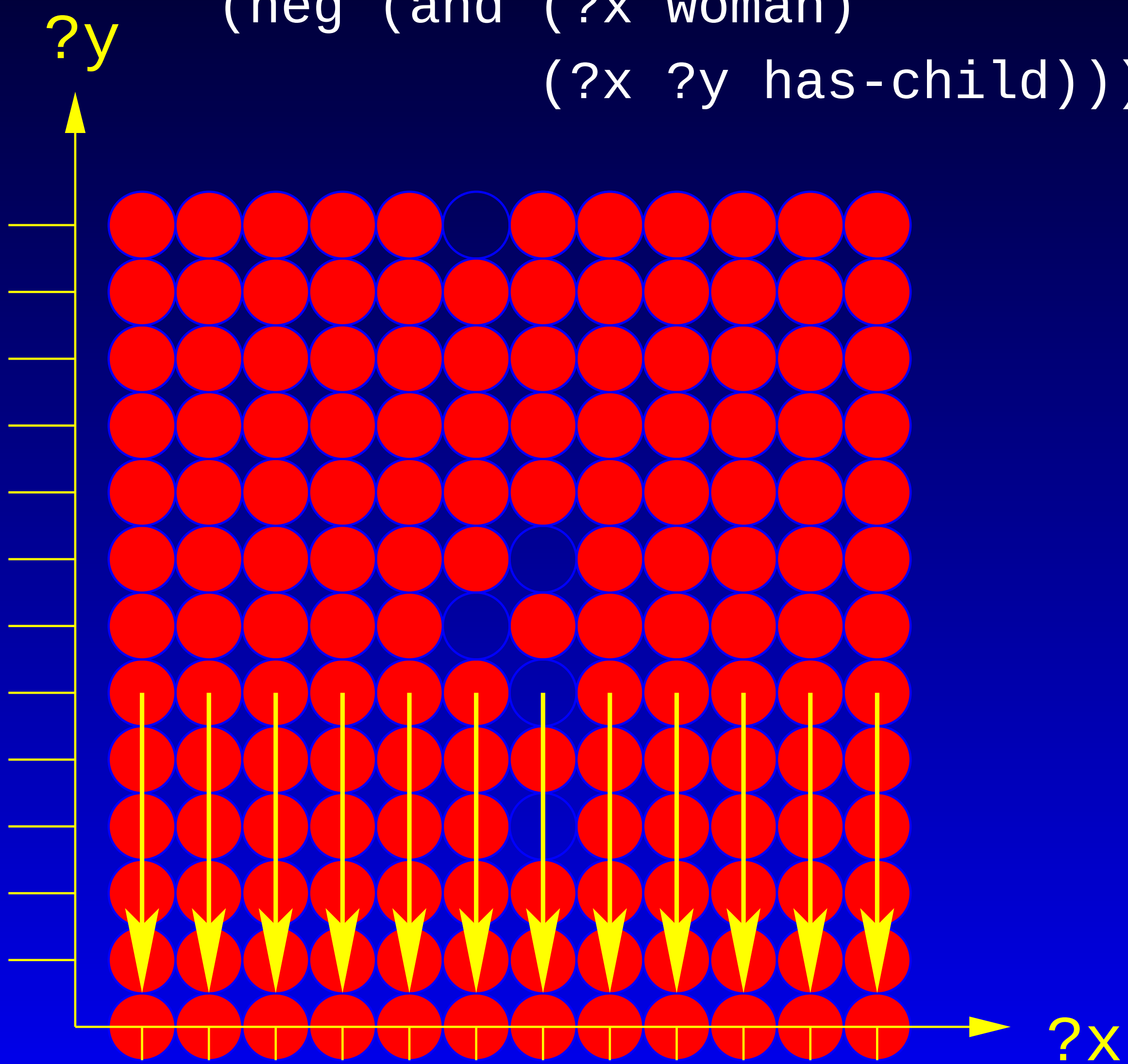
The Projection Operator (2)

```
Q2: (retrieve (?x)
              (neg (and (?x woman)
                        (?x ?y has-child))))
```



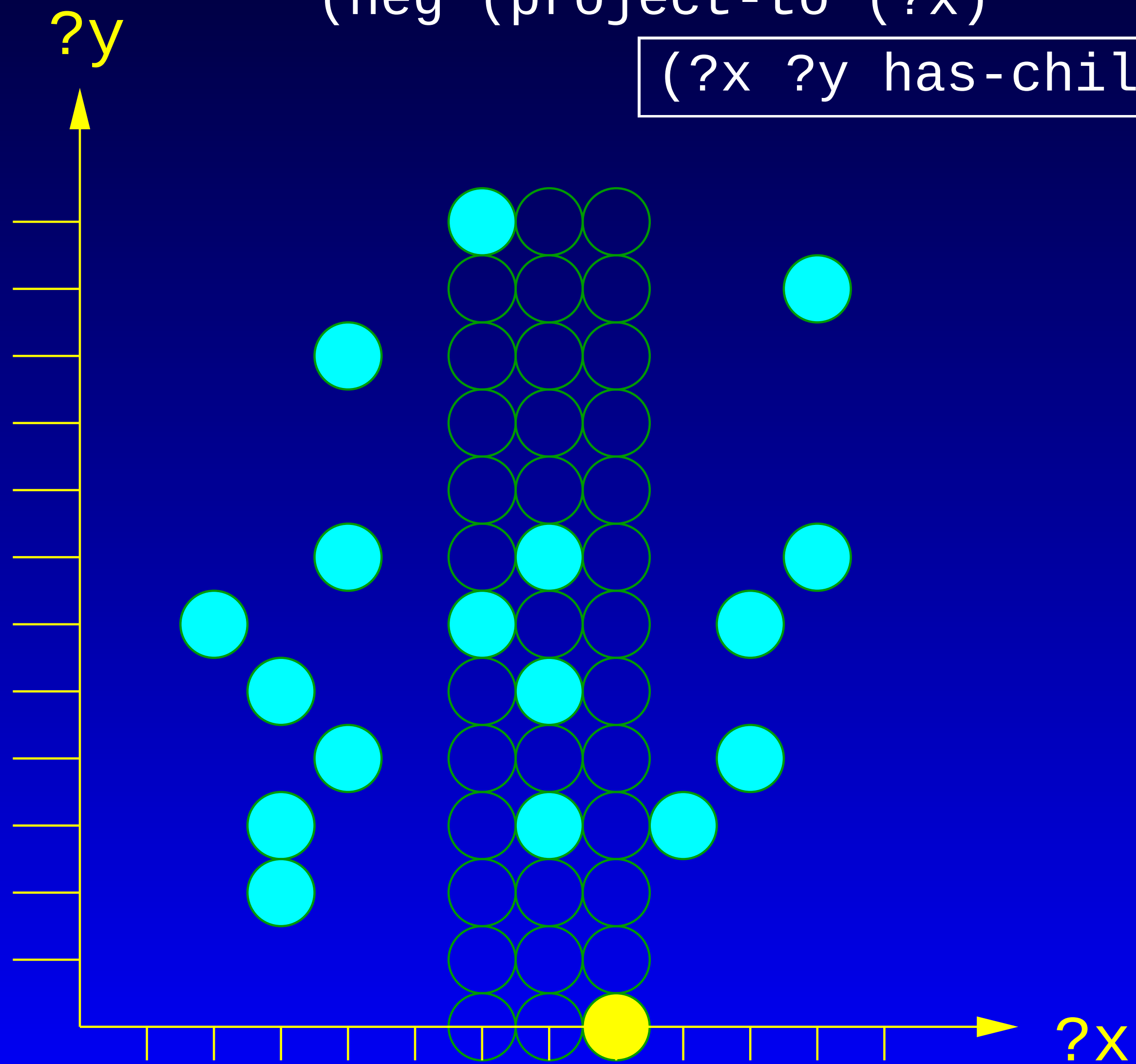
The Projection Operator (2)

- Q2: (retrieve $\boxed{\texttt{(?x)}}$
  (neg (and (?x woman)
  (?x ?y has-child))))



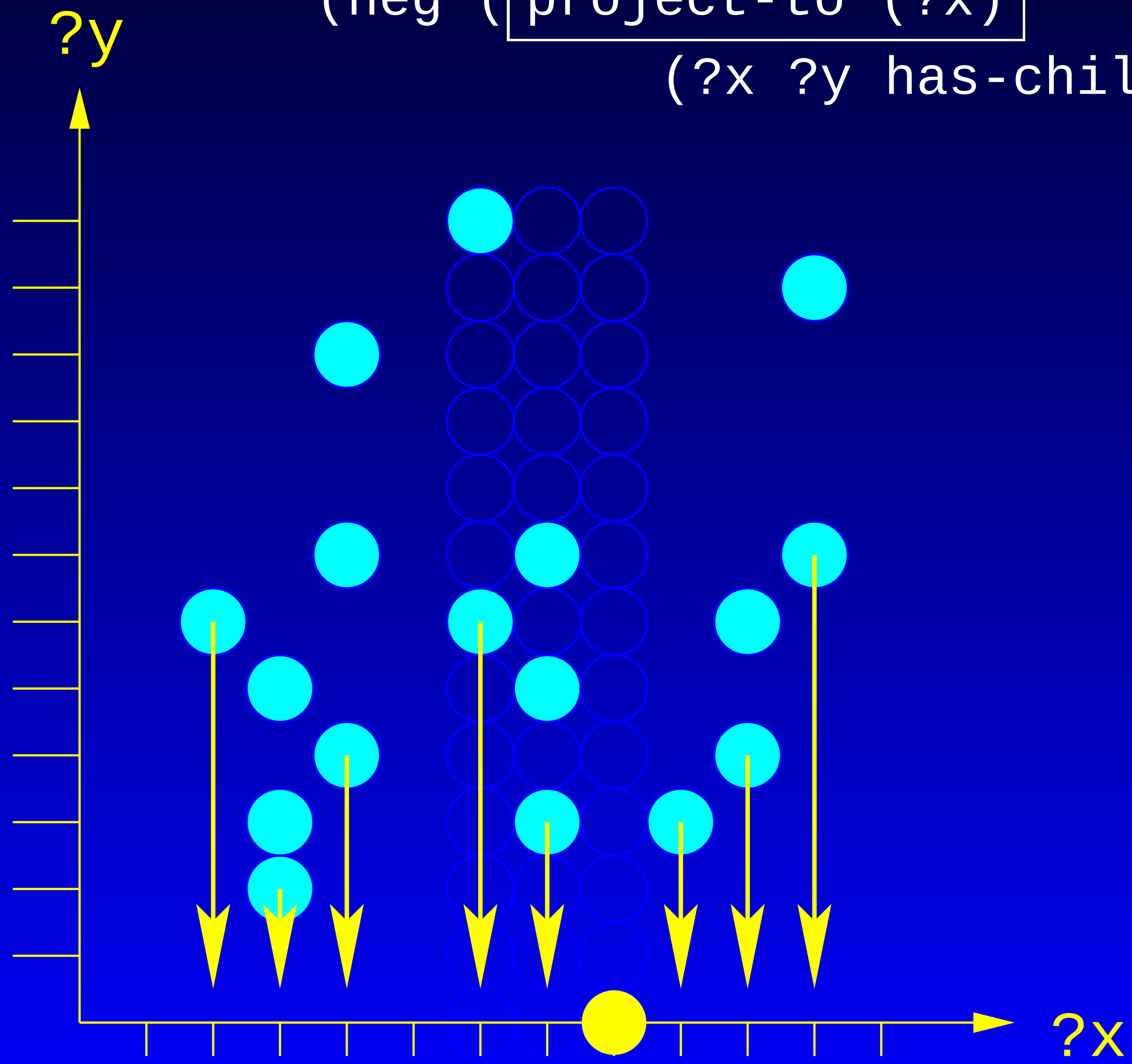
The Projection Operator (2)

```
Q3: (retrieve (?x)
              (and (?x woman)
                   (neg (project-to (?x)
                             (?x ?y has-child))))
```



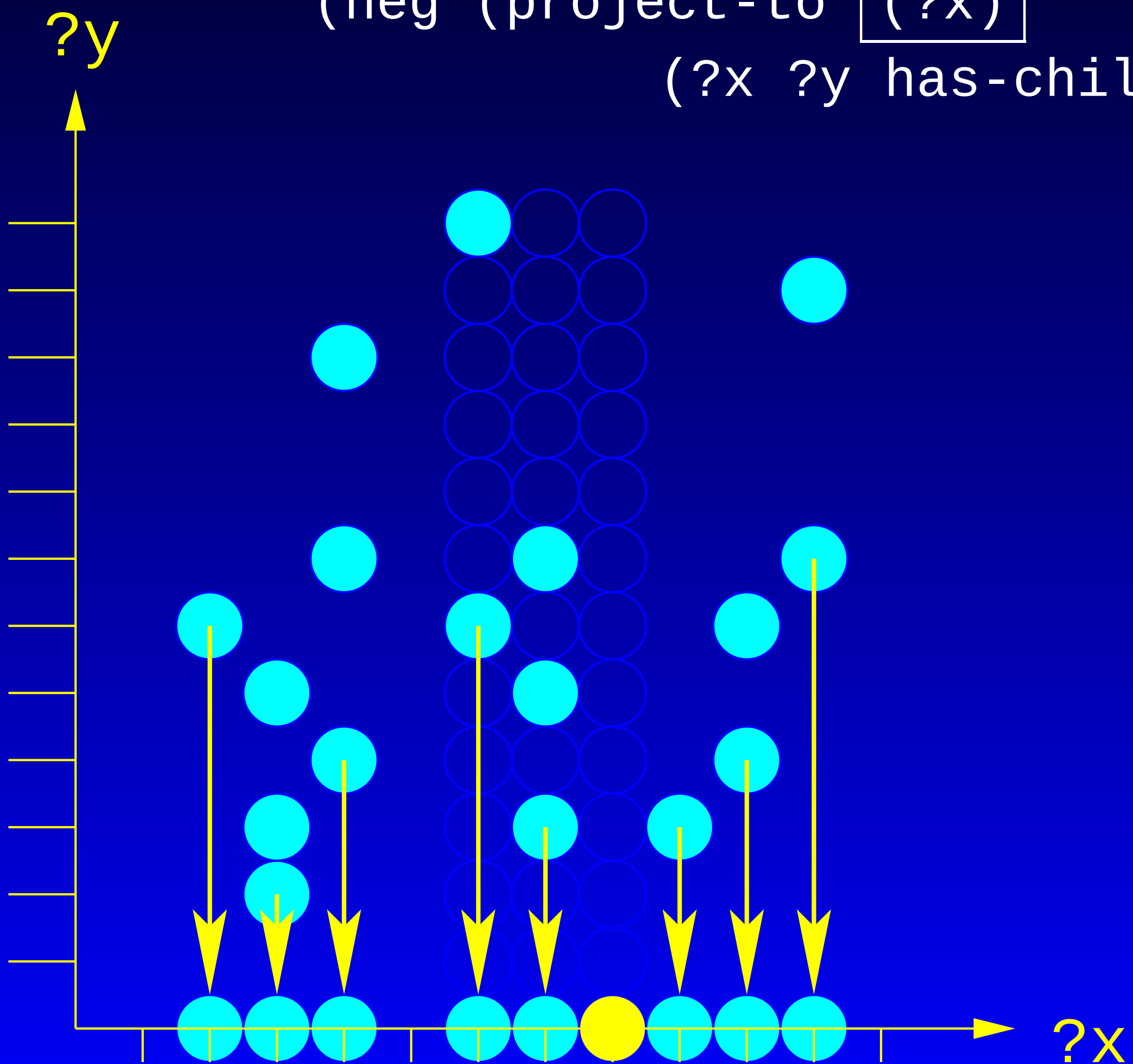
The Projection Operator (2)

```
Q3: (retrieve (?x)
              (and (?x woman)
                   (neg (project-to (?x)
                                (?x ?y has-child)))))
```



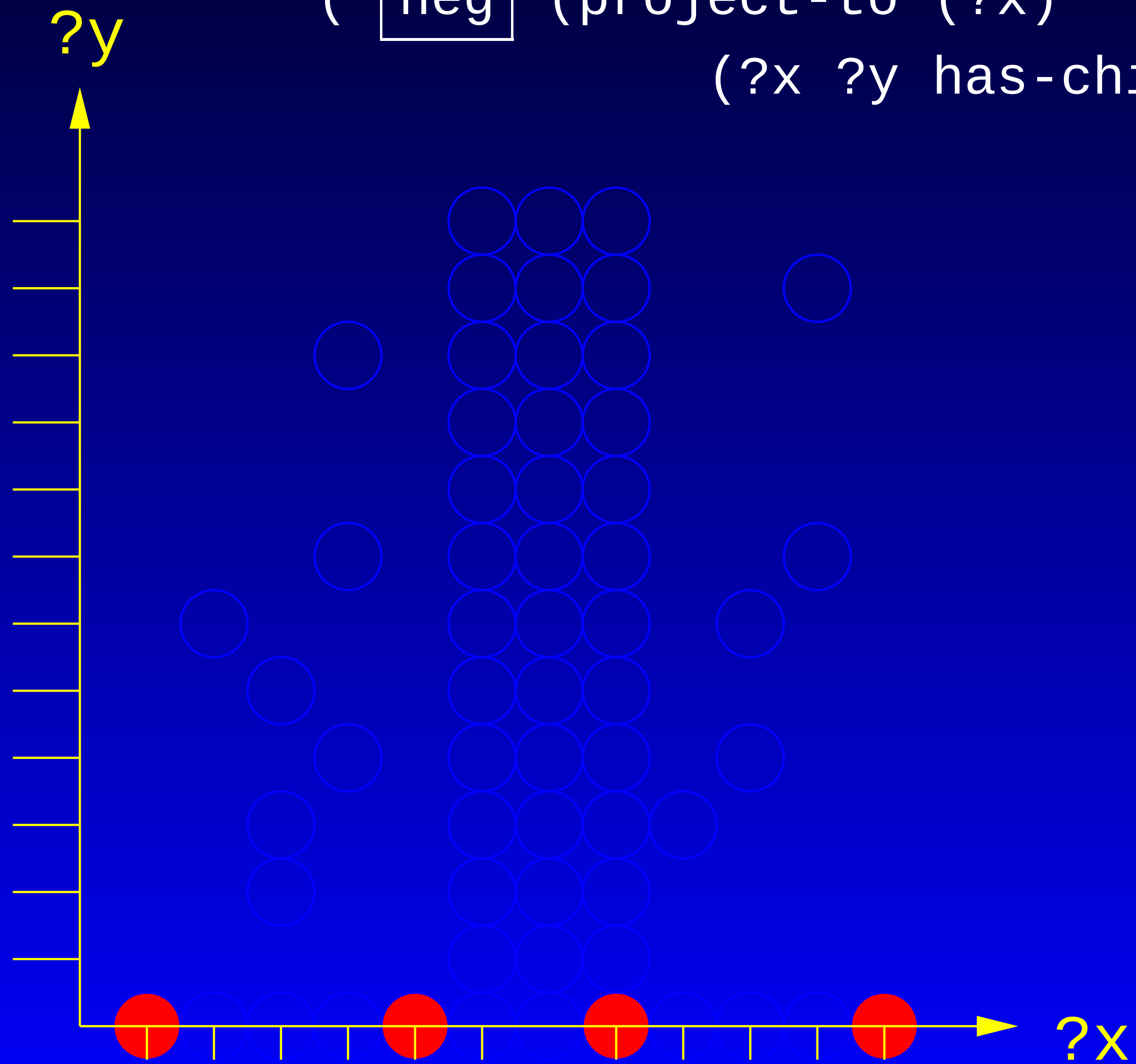
The Projection Operator (2)

```
Q3: (retrieve (?x)
              (and (?x woman)
                   (neg (project-to (?x)
                              (?x ?y has-child)))))
```



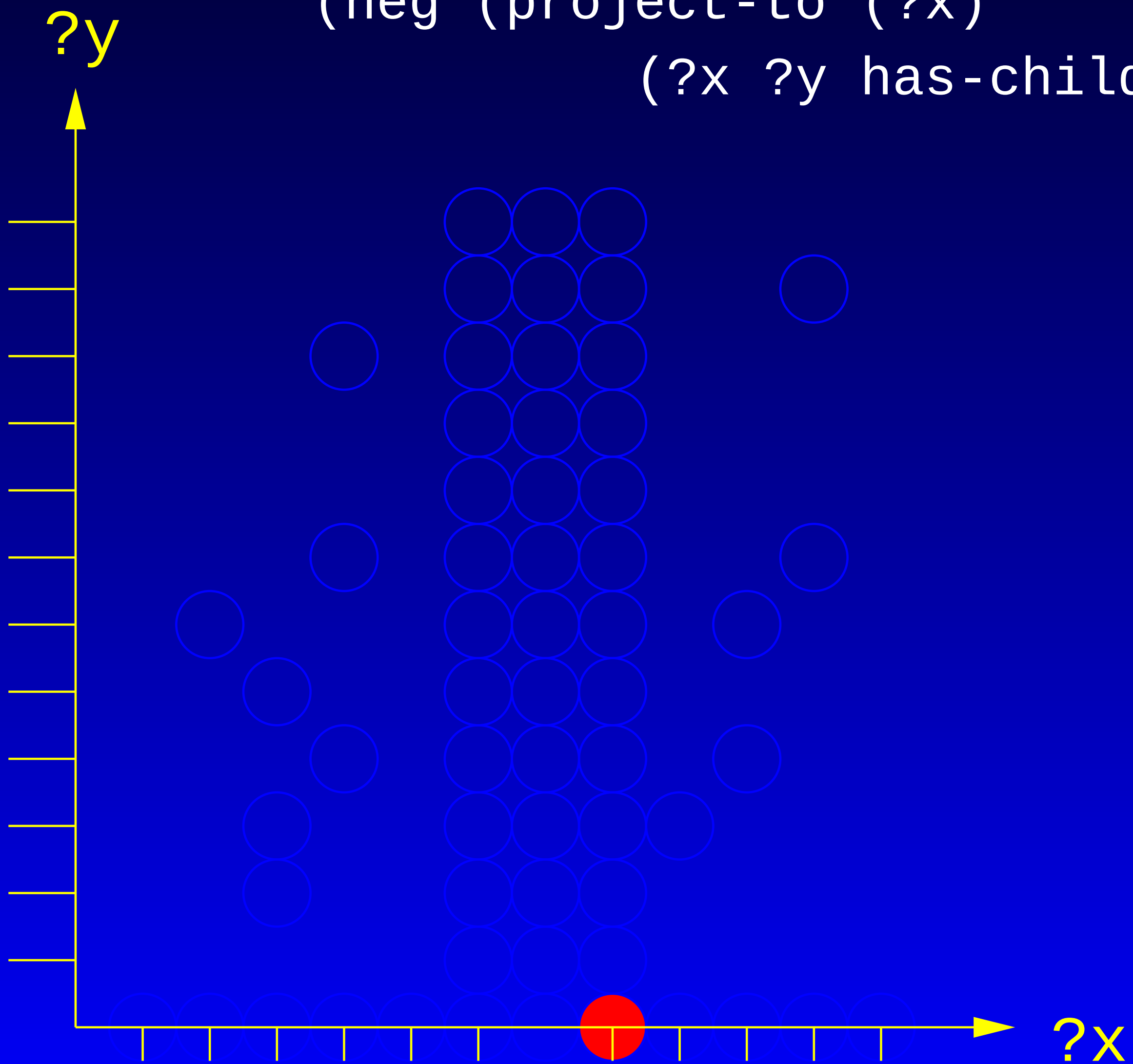
The Projection Operator (2)

```
Q3: (retrieve (?x)
              (and (?x woman)
                   ( neg (project-to (?x)
                               (?x ?y has-child)))))
```



The Projection Operator (2)

```
Q3: (retrieve (?x)
              (and (?x woman)
                   (neg (project-to (?x)
                             (?x ?y has-child)))))
```
✓



. . . Some Syntactic Sugar

- Due to the new projection operator, some “special syntax” from the older nRQL (DL ’04) can now be expressed
- `(?x (has-known-successor has-child))`
  `= (project-to (?x) (?x ?y has-child))`
- `(?x NIL has-child)` (borrowed from LOOM)
  `= (neg`
  `(?x (has-known-successor has-child)))`
  `= (neg`
  `(project-to (?x) (?x ?y has-child)))`
- now expressible in terms of `project-to`

Querying OWL KBs

- OWL datatype properties:

```
<owl:Class rdf:ID="Person">
  <rdfs:label>person</rdfs:label>
</owl:Class>

<owl:DatatypeProperty rdf:ID="age">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource=
    "http://www.w3.org/2001/XMLSchema#integer" />
</owl:DatatypeProperty>

<Person rdf:about="http://www.test.com/michael">
   <age>34</age>
</Person>
```

nRQL & Datatype Properties

- Idea: handle OWL DTP like concrete domain attributes

```
? (retrieve
    (?x
      (datatype-fillers
         (|http://www.test.com/test.owl#age| ?x)))
    (?x (some |http://www.test.com/test.owl#age|
              (and (min 30) (max 35)))))

> (((?X |http://www.test.com/michael|)
    ((:TOLD-VALUE
     (|http://www.test.com/test.owl#age| ?X)) (34))))
```

- Extended Racer concept syntax (expressions like `(and (min 30) (max 35))` only recognized by nRQL)

nRQL & Annotation Properties

```
<owl:AnnotationProperty rdf:ID="my-comment">
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#person"/>
</owl:AnnotationProperty>

<person rdf:ID="i">
  <my-comment rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#string"
  >My comment</my-comment>
</person>
```

- A special head projection operator `annotations (told-value)` is provided by nRQL
- Similar to querying for datatype properties

Expressivity Problems

- Access to “data values” in OWL docs (fillers of datatype/annotation properties) is restricted
- from the DL perspective, only the (extended) Racer concept expression language can be used
- How to retrieve all individuals which have (CD attribute or DTP) fillers containing substring x?
- Solution: maintain a data substrate in parallel to an ABox
- the data substrate is used to automatically “mirror” the ABox
- offer query access to this substrate by means of a hybrid query language - nRQL

Hybrid Queries

```
(retrieve (?x ?*name ?*age)
   (and (?x (and |http://...#person|
                 (an |http://...#age|)))
        (?*x ?*name |http://...#name|)
        (?*name ( (:predicate (search "wessel"))
                  ((:predicate (search "michael"))
                   (:predicate (search "achim")))))
        (?*x ?*age |http://...#age|)
        (?*age ((:predicate (< 40))))))
```

- New sort of variables: `*?x (*$?x)`, ranging over data nodes
- Data nodes can also be data values in OWL documents
- Data nodes/edges have descriptive labels: kind, role, property, …
- Notion of entailment for labels of nodes/edges
- Data query atoms are in pos. CNF & contain literals and predicates.

Formal Semantics - Auxiliaries

- The projection
  $\mathcal{T}' =_{def} \{ \langle t_{i_1}, \ldots, t_{i_m} \rangle \mid \langle t_1, \ldots, t_n \rangle \in \mathcal{T} \} = \pi_{\langle i_1, \ldots, i_m \rangle}(\mathcal{T})$ of $\mathcal{T}$ to the components mentioned in the index vector $\langle i_1, \ldots, i_m \rangle$.
  Example:
  $\pi_{\langle 1,3 \rangle}\{\langle 1,2,3 \rangle, \langle 2,3,4 \rangle\} = \{\langle 1,3 \rangle, \langle 2,4 \rangle\}$.
- If $\vec{b}$ is a bit vector which contains exactly m ones, and $\mathcal{B}$ is a set, $\mathcal{T}$ is a set of m-ary tuples, then
  the n-dimensional extension $\mathcal{T}'$ of $\mathcal{T}$ w.r.t. $\mathcal{B}$ and $\vec{b}$ is defined as $\mathcal{T}' =_{def}$
  $\{ \langle i_1, \ldots, i_n \rangle \mid \langle j_1, \ldots, j_m \rangle \in \mathcal{T}, 1 \leq l \leq m, 1 \leq k \leq n$.
  with $i_k = j_l$ if $b_k = 1$, and b_k is the lth one (1) in $\vec{b}$,
  otherwise, $i_k \in \mathcal{B}$ $\}$
  and denoted by $\chi_{\mathcal{B}, \langle b_1, \ldots, b_n \rangle}(\mathcal{T})$.
  Example:
  $\chi_{\{a,b\},\langle 0,1,0,1 \rangle}(\{\langle x,y \rangle\}) =$
  $\{\langle a,x,a,y \rangle, \langle a,x,b,y \rangle, \langle b,x,a,y \rangle, \langle b,x,b,y \rangle\}$.

Formal Semantics - Atoms

$$(q'_{x_i}\ concept\_expr)^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{concept\_instances}(\mathcal{A}, concept\_expr))$$

$$(q'_{x_i}\ q'_{x_j}\ rolen\_expr)^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{role\_pairs}(\mathcal{A}, role\_expr)), \text{ if } i \neq j$$

$$(q'_{x_i}\ q'_{x_i}\ role\_expr)^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{role\_pairs}(\mathcal{A}, role\_expr) \cap \mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}})$$

$$(\texttt{same-as}\ \ q'_{x_i}\ ind)^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i\}}}(\{ind\})$$

$$(q'_{x_i}\ q'_{x_j}\ (\texttt{constraint}\ \ attrib_1\ attrib_2\ P))^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i,j\}}}(\mathsf{predicate\_pairs}(\mathcal{A}, attrib_1, attrib_2, P)), \text{ if } i \neq j$$

$$(q'_{x_i}\ q'_{x_i}\ (\texttt{constraint}\ \ attrib_1\ attrib_2\ P))^{\mathcal{E}} =_{def}$$

$$\chi_{\mathsf{Inds}_{\mathcal{A}}, \vec{\mathbf{1}}_{n,\{i\}}}(\mathsf{predicate\_pairs}(\mathcal{A}, attrib_1, attrib_2, P) \cap \mathcal{ID}_{2,\mathsf{Inds}_{\mathcal{A}}})$$

Formal Semantics - Bridge2DL

- Semantics of DL standard ABox retrieval functions ("Bridge to Racer's basic ABox retrieval functions")

$$\mathsf{concept\_instances}(\mathcal{A}, concept\_expr) =_{def}$$
$$\{\, i \mid \ i \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models concept\_expr(i)\,\}$$

$$\mathsf{role\_pairs}(\mathcal{A}, role\_expr) =_{def}$$
$$\{\, \langle i, j \rangle \mid \ i, j \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models role\_expr(i, j)\,\}$$

$$\mathsf{predicate\_pairs}(\mathcal{A}, attrib_1, attrib_2, P) =_{def}$$
$$\{\, \langle i, j \rangle \mid \ i, j \in \mathsf{Inds}_{\mathcal{A}}, (\mathcal{A}, \mathcal{T}_{\mathcal{A}}) \models$$
$$\exists x, y : attrib_1(i, x) \wedge attrib_2(j, y) \wedge P(x, y)\,\}$$

Formal Semantics - Bodies

$$
\begin{array}{rcl}
(\texttt{and} \;\; q_1 \ldots q_i)^{\mathcal{E}} & =_{def} & \bigcap_{1 \leq j \leq i} q_j^{\mathcal{E}} \\
(\texttt{union} \;\; q_1 \ldots q_i)^{\mathcal{E}} & =_{def} & \bigcup_{1 \leq j \leq i} q_j^{\mathcal{E}} \\
(\texttt{neg} \;\; q)^{\mathcal{E}} & =_{def} & (\mathsf{Inds}_{\mathcal{A}})^n \setminus q^{\mathcal{E}} \\
(\texttt{inv} \;\; q)^{\mathcal{E}} & =_{def} & \mathsf{inv}(q^{\mathcal{E}}), \text{ where } \mathsf{inv} \\
 & & \text{reverses all tuples} \\
(\texttt{project-to} \;\; \texttt{(}x_{i_1,q} \ldots x_{i_k,q}\texttt{)} \;\; q)^{\mathcal{E}} & =_{def} & \pi_{\langle i_1, \ldots, i_k \rangle}(q^{\mathcal{E}})
\end{array}
$$

- Claim: the given semantics is easy to catch
  - only basic set-theory required
  - easy to visualize

$\Rightarrow$ good for users

Features of the nRQL Engine

- Integral part of RacerPro

⇒ no communication overhead with Racer (an "external" query answering engine would have to communicate a lot with Racer, performance comparable to nRQL's performance would be unachievable)

- "Multi-query" answering (multi-threaded)
- Different query processing modes
- Degree of completeness configurable
- Non-recursive defined queries (macro queries)
- Simple rule engine
- Semantic & cost-based Query Optimizer

Query Processing Modes

- Set-at-a-time mode
  - synchronous mode of interaction, call to `retrieve` blocks until answer is computed, returned as a bunch
- Tuple-at-a-time mode
  - asynchronous mode of interaction, call to `retrieve` returns immediately with query identifier
  - query thread works in the background
  - `get-next-tuple <id>` returns next tuple of query `<id>`
  - Lazy: compute next tuple if requested
  - Eager: precompute next tuple(s)

Degree of Completeness

- Mode 0: syntactic told information is used for query answering
- Mode 1: Mode 0 + exploited TBox information
- Mode 3: complete Racer ABox retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)
- Even more modes: “two-phase query processing”
  - Phase 1: deliver cheap tuples (incomplete)
  - Warn user; then, if next tuple requested, start
  - Phase 2: use full ABox reasoning to deliver remaining tuples (complete)

Two-Phase Query Processing

TBox:

$$
\begin{aligned}
person &\sqsubseteq \top \\
man &\sqsubseteq person \\
woman &\sqsubseteq person \\
spouse &\doteq woman \sqcap \\
&\quad (\exists married\_to.man)
\end{aligned}
$$

ABox :

$$
\begin{aligned}
&spouse(doris) \\
&spouse(betty) \\
&man(adam) \\
&woman(eve) \\
&maried\_to(eve, adam)
\end{aligned}
$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `((?X DORIS))`

Two-Phase Query Processing (2)

TBox:

$$\begin{aligned} person &\sqsubseteq \top \\ man &\sqsubseteq person \\ woman &\sqsubseteq person \\ spouse &\doteq woman \sqcap \\ & \quad (\exists married\_to.man) \end{aligned}$$

ABox :

$$\begin{aligned} & spouse(doris) \\ & spouse(betty) \\ & man(adam) \\ & woman(eve) \\ & maried\_to(eve, adam) \end{aligned}$$

- `(retrieve (?x) (?x spouse))`

⇒ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

⇒ `((?X BETTY))`

Two-Phase Query Processing (3)

TBox:

$$\begin{aligned} person &\sqsubseteq \top \\ man &\sqsubseteq person \\ woman &\sqsubseteq person \\ spouse &\doteq woman \sqcap \\ &\quad (\exists married\_to.man) \end{aligned}$$

ABox :

$$\begin{aligned} &spouse(doris) \\ &spouse(betty) \\ &man(adam) \\ &woman(eve) \\ &maried\_to(eve, adam) \end{aligned}$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `:WARNING-EXPENSIVE-PHASE-TWO-STARTS`

Two-Phase Query Processing (4)

TBox:			**ABox :**
$person$	$\sqsubseteq$	$\top$	$spouse(doris)$
man	$\sqsubseteq$	$person$	$spouse(betty)$
$woman$	$\sqsubseteq$	$person$	$man(adam)$
$spouse$	$\doteq$	$woman \sqcap$	$woman(eve)$
		$(\exists married\_to.man)$	$maried\_to(eve, adam)$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `((?X EVE))`

Two-Phase Query Processing (5)

TBox:			**ABox :**
$person$	$\sqsubseteq$	$\top$	$spouse(doris)$
man	$\sqsubseteq$	$person$	$spouse(betty)$
$woman$	$\sqsubseteq$	$person$	$man(adam)$
$spouse$	$\doteq$	$woman \sqcap$	$woman(eve)$
		$(\exists married\_to.man)$	$maried\_to(eve, adam)$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `:EXHAUSTED`

Two-Phase Query Processing (6)

TBox:			ABox :
$person$	$\sqsubseteq$	$\top$	$spouse(doris)$
man	$\sqsubseteq$	$person$	$spouse(betty)$
$woman$	$\sqsubseteq$	$person$	$man(adam)$
$spouse$	$\doteq$	$woman \sqcap$	$woman(eve)$
		$(\exists married\_to.man)$	$maried\_to(eve, adam)$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-answer :query-1)`

$\Rightarrow$ `(((?X DORIS)) ((?X BETTY)) ((?X EVE)))`

Optimization & Caching

- Caching of Racer results (cache consistency, . . .)
- Lots of index structures (must be maintained, . . .)
- Cost-based optimizer (reordering of conjuncts and marking variables as non-generators, e.g. `?y` in `(retrieve (?x) (?x ?y has-child))`)
- Reasoning with Queries (optional, incomplete)
  - Query consistency check
  - Query entailment check (subsumption)

  $\Rightarrow$ maintenance of a “Query repository” DAG (similar to a taxonomy)
  - Query “realization” (adds implied conjuncts to enhance informdness of backtracking search)

Defined Queries

- nRQL offers a simple macro-mechanism

```
(defquery mother-of-child
          (?x ?y)
          (and (?x woman)
               (?x ?y has-child)))

(defquery mother-of-son
          (?x ?child)
          (and (?x ?child mother-of-child)
               (?child man)))
```

- no cyclic definitions allowed

Simple Rules

- nRQL offers a simple rule mechanism

```
(defrule
   (and (?x woman) (?y man) (?x ?y married))
        (neg (?x (has-known-successor has-child)))
   ((instance (new-ind child-of ?x ?y) human)
    (instance ?x mother)
    (instance ?y father)
    (related (new-ind child-of ?x ?y) ?x
             has-mother)
    (related (new-ind child-of ?x ?y) ?y
             has-father)))
```

- Rule antecedence is a query body; consequence is a list of <u>generalized ABox assertions</u>
- rules must be fired manually

Complex TBox Queries

- "What are the child (parent, descendant, ancestor) concepts of the concept `woman`"?
- Idea: view the taxonomy of a TBox as a relational structure (a DAG), stored as a "data substrate"
- use nRQL to query this structure with `tbox-retrieve`:

```
? (tbox-retrieve (?x ?y)
        (and (?x woman)
             (?x ?y has-child)))

> (((?X WOMAN) (?Y SISTER))
   ((?X WOMAN) (?Y AUNT))
   ((?X WOMAN) (?Y *BOTTOM*))
   ((?X WOMAN) (?Y MOTHER))
   ((?X WOMAN) (?Y GRANDMOTHER)))
```

Query Processing

- In principle, nRQL uses top-down query evaluation strategy:
  - each query evaluation plan determines the role of an atom as a generator or tester
  - optimizer: try to minimize required generators
- in the presence of `project-to`, this becomes more involved
  - sub-queries must be evaluated first, e.g., in case of `(neg (project-to ...))`

⇒ bottom-up-top-down mixture of query evaluation

- continuation-style implementation, can be compiled (see implementations of Prolog in Lisp)

Related Activities & Conclusion

- Benchmarking nRQL: wait for Ralf's talk
- nRQL as a basis for a subset of OWL-QL: Atila's & Jan's Poster
- "nRQL tab" for Protégé (Kruthi & Volker)
- RacerPorter supports life-cycle management and inspection of nRQL queries and rules
- only nRQL implementation: 29.553 LOC
- Future plans
  - rolling-up (support OWL-QLs "do-not-bind variables" for acyclic conjunctive queries)
  - better index structures for data substrate layer
  - database access

Thanks

for your

attention!